**Università degli Studi di Modena e Reggio Emilia**

Dipartimento di Ingegneria "Enzo Ferrari"
Corso di Laurea Magistrale in Ingegneria Informatica

# Tecniche di Continual Learning applicate a serie temporali finanziarie

**Candidato:**
Alberto Zurli

**Relatore:**
Prof. Simone Calderara

**Correlatori:**
Dott.ssa Alessia Bertugli
Dott. Jacopo Credi

Anno Accademico 2020/2021

# Ringraziamenti

Al termine di questo percorso è necessario ringraziare alcune persone per il sostegno ricevuto e l'aiuto nei momenti di difficoltà:

- i miei compagni di corso, con cui spesso mi sono confrontato e ho collaborato, e senza i quali mi sarei trovato molto in difficoltà, soprattutto quelli che oggi posso chiamare amici;
- i miei amici, diverse volte in grado di farmi vivere momenti indimenticabili al di fuori dell'università;
- i miei genitori, punto di riferimento nel mio percorso e sempre pronti a sostenermi, e mia sorella Elena;
- Arianna, per avermi sempre supportato ed incoraggiato, soprattutto in quest'ultima fase del mio percorso;
- i miei relatori prof. Simone Calderara, i dottori Alessia Bertugli e Jacopo Credi e tutti i colleghi ad Axyon, sempre disponibili a rispondere ad ogni mio quesito ed essenziali per lo svolgimento di questa tesi.

# Abstract

Il Continual Learning è stato oggetto di grande interesse negli ultimi anni, fornendo importanti risultati nel campo della classificazione di immagini. In questa tesi si studiano e applicano i principali algoritmi di continual learning per valutare la loro efficacia su serie temporali finanziarie. In particolare, ci si focalizzerà sulla capacità di salvare i principali pattern del mercato per confrontarli con l'andamento corrente di esso al fine di valutarne l'andamento futuro.

# Indice

# Elenco delle figure

# Elenco delle tabelle

# Capitolo 1

# Introduzione

Il mercato finanziario è un luogo prettamente virtuale, all'interno del quale vengono scambiati strumenti finanziari di varia natura, come azioni, contratti o moneta. Esso è tra i più attivi al mondo, aperto 24 ore su 24 per 5 giorni a settimana e in grado di muovere miliardi di euro al giorno coinvolgendo differenti tipologie di partecipanti, pubblici e privati. In passato gli investimenti erano effettuati via telefono da una cerchia ristretta di persone con a disposizione un numero limitato di informazioni, mentre allo stato attuale gli attori coinvolti sono cambiati radicalmente. La forte disponibilità di dati in tempo reale tramite la diffusione a livello mondiale della rete Internet e le grandi capacità computazionali dei calcolatori hanno fatto sì che la maggior parte degli investimenti non vengano effettuati da trader umani, ma bensì da un numero sempre crescente di macchine dotaste di "intelligenza" in grado di capire le tempistiche migliori per effettuare operazioni finanziarie. Ha preso dunque piede il concetto di *Algorithmic trading*, in cui l'algoritmo è caratterizzato da un set di regole definite per compiere determinate azioni in base allo stato del mercato. Queste regole sono scritte da trader umani che studiano come effettuare particolari scelte di mercato. La naturale evoluzione di questi algoritmi è l'uso di tecniche di *machine learning*, le quali non prevedono più l'intervento umano nella definizione delle regole ma solo a livello di metodo. L'aiuto apportato da queste nuove tecniche è l'estrazione automatica di conoscenza dai dati passati sotto forma di pattern, permettendo di ricavare delle regole più definite e utili nel mercato finanziario. Il problema dell'utilizzo di tecniche di machine e *deep* learning con serie temporali è la necessità di riallenare periodicamente i modelli per permetterne l'aggiornamento e l'acquisizione della conoscenza dei dati più recenti. Questo però porta alla sostituzione delle conoscenze dei dati osservati nei primi step con i dati più recenti. Il Continual Learning si colloca tra le tecniche di deep learning ancora in fase di ricerca ma che promettono risultati di rilievo in ambiti di classificazione di immagini. Esso si pone come soluzione alla necessità di allenare periodicamente i modelli e al contempo permette di apprendere task multipli senza sovrascrivere conoscenze pregresse. Lo studio e l'applicazione di queste tecniche potrebbe risultare rilevante all'interno del mercato finanziario.

## 1.1 Motivazioni

L'idea dell'utilizzo di tecniche di Continual Learning in ambito finanziario nasce in una azienda estremamente innovativa e all'avanguardia nell'Intelligenza Artificiale, Axyon AI. Costantemente alla ricerca di nuove soluzioni che possano migliorare i propri sistemi predittivi, essa si pone l'obiettivo di dedicare gran parte delle risorse a disposizione all'innovazione tecnologica, con lo scopo di portare i propri risultati al più alto livello possibile.

La volontà di esplorare queste tecniche nasce dalla natura ciclica del mercato, in particolare quello azionario e delle *commodities*, ovvero quei beni per cui c'è domanda ma che sono offerti senza differenze qualitative sul mercato e sono fungibili, come materie prime o metalli preziosi. È stato dimostrato di come gli andamenti del mercato nel corso del tempo siano qualitativamente simili, e pertanto apprendere il comportamento della serie temporale nel passato potrebbe essere utile per predirne l'andamento futuro.

Esplorare le potenzialità del Continual Learning è un processo totalmente inedito nell'ambito del mondo finanziario e pertanto non si hanno risorse specifiche o riferimenti ufficiali per la risoluzione di problemi questo ambito.

## 1.2 Obiettivi

Ispirati dal grande successo delle tecniche di deep learning in ambito finanziario nell'ultimo periodo, lo scopo finale della tesi è valutare in maniera approfondita le potenzialità del Continual Learning come metodologia per permettere ad una macchina "intelligente" di operare in maniera autonoma senza la necessità di aggiornamenti da parte dell'uom. Per far ciò risulta necessario l'utilizzo di strumenti all'avanguardia e delle ultime tecnologie disponibili sul mercato, integrandole con la piattaforma aziendale per la costruzione di Deep neural networks. In aggiunta ci si pone l'obiettivo di sviluppare un ambiente di simulazione del mondo finanziario in cui poter liberamente valutare in termini qualitativi e quantitativi i modelli creati nel tempo.

## 1.3 Azienda – Axyon AI

Axyon AI è una azienda di Modena specializzata in soluzioni di intelligenza artificiale per il mondo della finanza. Nata come succursale di DM Digital SRL, Axyon si fonda su tre avantaggi competitivi, permettendole di ricoprire una posizione di forza nel mercato di settore:

- **Tecnologia**. L'azienda possiede una piattaforma proprietaria di Deep Learning basata sul mondo finanziario in grado di sviluppare modelli di predizione ad altissima efficienza.

- **Esperienza**. Dal 2015 l'azienda offre prodotti per il mercato, attualmente ne sono disponibili due mentre un terzo è in fase di sviluppo. L'acquisizione di competenze è migliorata grazie alla possibilità di offrire soluzioni personalizzate in base alle esigenze dei clienti.
- **Skills**. Axyon è composta da un team eterogeneo e supportato da un grande network di consulenti e partners.

Questi tre aspetti sono potenziati anche dai legami con il mondo universitario, consolidati dallo scambio di persone e condivisone in progetti di ricerca. Al contempo Axyon punta a diventare leader di mercato nel settore in rapida crescita della finanza ad alto margine offrendo predizioni altamente accurate basate su modelli di *Deep Learning*. Per conseguire tali obiettivi vengono percorse tre strade:

- Tecnologia: migliorare ed estendere la tecnologia proprietaria per mantenere un vantaggio competitivo nello sviluppo di modelli di AI.
- Soluzioni: offrire i modelli predittivi più completi e accurati a specifici casi d'uso nella finanza ad alto margine.
- Mercato: collaborazione con partner commerciali già presenti nei vari mercati in modo da poter accedere direttamente a livello di *decision making*.

## 1.3 Outline dei contenuti

Di seguito si riporta una sintesi sui contenuti della tesi, divise per capitoli.

- Nel secondo capitolo si offre una panoramica riguardante i concetti fondamentali dell'intelligenza artificiale e sullo stato dell'arte raggiunto.
- Nel terzo capitolo si affrontano nello specifico le scelte implementative con una granularità fine. Si trattano brevemente anche le librerie e framework utilizzati.
- Il quarto capitolo tratta in maniera approfondita le tecniche di Continual Learning utilizzate e non, analizzando analogie e differenze tra di esse.
- Il quinto capitolo riporta la trattazione dei risultati ottenuti da un punto di vista sia qualitativo che quantitativo.
- Infine, nel sesto capitolo vengono discussi gli obiettivi raggiunti durante lo sviluppo del progetto, portando infine ad una breve analisi di quali potrebbero essere le possibili aree d'azione in futuro per quanto riguarda tutte le fasi di sviluppo.

# Capitolo 2

# Stato dell'arte

Di seguito vengono presentate e discusse le tecnologie più diffuse nei settori dei problemi affrontati in questa tesi.

## 2.1 Machine Learning

Il Machine Learning è una disciplina appartenente all'ambito dell'*Artificial Intelligence* che studia come progettare e realizzare sistemi software in grado di offrire prestazioni che, ad un osservatore esterno, sarebbero attribuibili soltanto all'uomo. Un generico algoritmo di machine learning viene addestrato per risolvere un determinato problema estraendo conoscenza dai dati ricevuti in input per produrre un output in maniera del tutto autonoma, senza quindi seguire routine o istruzioni codificate a mano. Osservando la curva dell'*Hype Cycle* di Gartner (fig. 2.1), che mira a rappresentare le fasi di sviluppo, maturità e adozione di nuove tecnologie, si può notare come il Machine Learning abbia da poco superato il picco di massimo interesse. Ciò non significa che la tecnologia sia obsoleta ma che non è applicabile ad ogni ambito e nel futuro andrà in contro ad una settorializzazione dove sarà, ancora per lungo tempo, la tecnologia di riferimento. Oggi il Machine Learning è estremamente diffuso in moltissimi ambiti non solo in informatica, ma anche in medicina e industria solo per citarne alcuni.

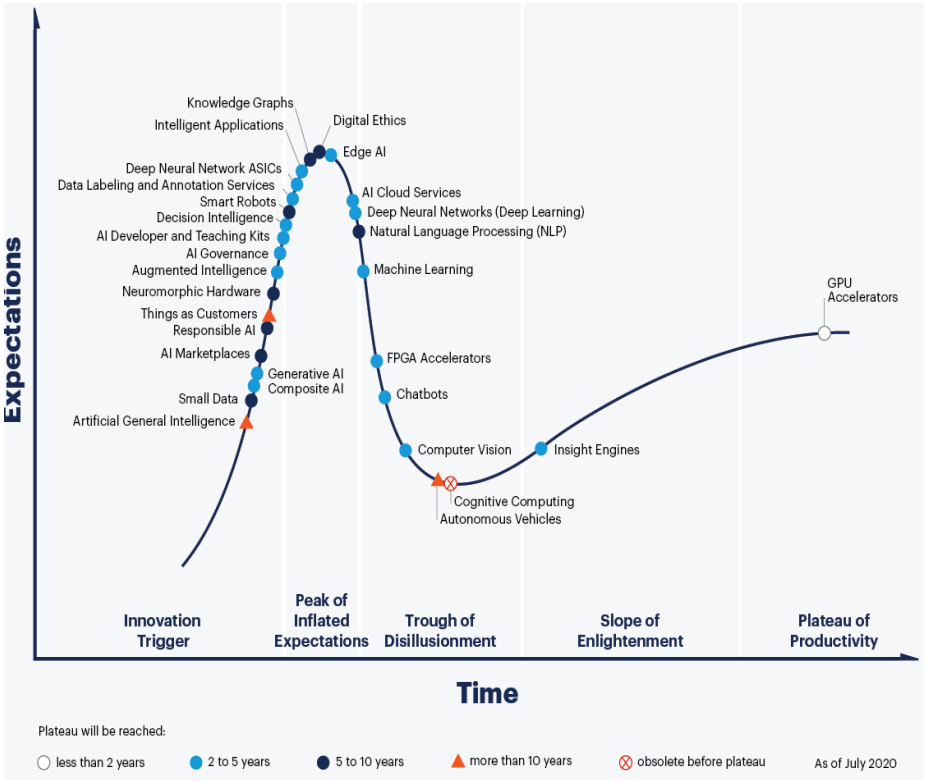



Figura 2.1: Gartner's 2020 Hype Cycle

### 2.1.1 Supervised e unsupervised learning

L'apprendimento di un algoritmo di Machine Learning può essere diviso in due principali categorie: apprendimento supervisionato (supervised learning) e apprendimento non supervisionato (unsupervised learning).

L'apprendimento supervisionato necessita di osservazioni costituite da coppie di dati $(x, y)$ con $x \in X$ e $y \in Y$. $X$ corrisponde all'insieme degli esempi e $Y$ corrisponde all'insieme delle classi (o label) associabili agli esempi. Ogni dato di input è caratterizzato da valori, tipicamente numerici o booleani (features), che offrono una descrizione quantitativa del dato in esempio. Assumendo che esista una funzione $f$ ignota tale che $y = f$, lo scopo di questi algoritmi è quello di apprendere una funzione (o modello) $y\tilde{y} = gf(x)$ che rappresenti la miglior approssimazione possibile della funzione ignota $f$, ovvero tale che $\tilde{y} \cong y$. Per raggiungere lo scopo, il problema di apprendimento viene trasformato nel problema di minimizzazione di una funzione di costo $L(y, \tilde{y})$ che quantifica la differenza tra il risultato dell'algoritmo $\tilde{y}$ e la label $y$. Esistono funzioni di costo diverse in base al problema, come classificazione o regressione. L'obiettivo della fase di *training* è minimizzare la funzione di costo (*loss* function). Una volta concluso, è possibile applicare il modello su dati mai visti (dati di test) e valutare, tramite apposite metriche, la bontà complessiva della funzione $gf(x)$. Il learning supervisionato si può suddividere a sua volta in due macrocategorie: problemi di regressione, nei quali l'obiettivo è predire valori appartenenti ad uno spettro continuo, e problemi di classificazione, in cui lo scopo è predire la corretta classe e quindi valori discreti.

L'apprendimento non supervisionato invece, si pone un problema differente. Forniti in input una serie di dati $x \in X$ e una funzione di costo, questa dipendente dal tipo di problema da affrontare, l'obiettivo è estrarre una struttura o caratteristiche (features) generalmente nascoste all'interno dei dati. Un esempio di questo tipo di learning è il *clustering*, dove l'output è il raggruppamento secondo caratteristiche, non note a priori e alle volte nemmeno una volta concluso il learning, dei dati in input.

## 2.2 Deep Learning

Con il termine *Deep learning* ci si riferisce ad una branca del Machine Learning che utilizza modelli matematici chiamati Reti Neurali. Questi modelli hanno le stesse finalità degli algoritmi di machine learning, ma hanno la peculiarità che sono modellate secondo il cervello umano.

### 2.2.1 Percettrone

Contrariamente a quanto si possa pensare, i primi studi su reti neurali in grado di emulare il cervello umano risalgono agli anni '40 grazie ai neuropsicologi McCulloch e Pitts [1]. Nel 1958 Frank Rosenblatt creò la prima formulazione matematica di un neurone o *perceptron* [2]. Il percettrone nasce come un classificatore binario che associa un dato in input ad un valore di output tramite una semplice funzione lineare (fig. 2.2). In seguito alla creazione del percettrone si andò incontro ad una fase di stallo dovuta principalmente all'impossibilità di unire tra loro più neuroni per creare reti complesse in grado di risolvere problemi di tipo non lineare. Soltanto nel 1974 venne sviluppato il metodo di allenamento chiamato *backpropagation* e, grazie ad esso, si riuscì a combinare più neuroni assieme dando vita reti più complesse chiamate *multi-layer perceptron* o MLP. Dagli anni '80 ai giorni nostri la ricerca è avanzata sempre di più grazie allo sviluppo di tecnologie in grado di offrire grandi quantità di potenza computazionale, come le GPU, per sviluppare reti di dimensioni e complessità sempre più crescenti, in grado di risolvere problemi di grandi dimensioni elaborando pattern impossibili da individuare dagli esseri umani.

### 2.2.2 Struttura e reti complesse

L'unità atomica di una rete neurale è il neurone artificiale. Esso riceve in ingresso un numero $n$ di input $x$, cui vengono uniti ai pesi $w$ del neurone e a cui viene sommato un bias $b$. Segue l'applicazione di una funzione di attivazione $\varphi$ in grado di mappare in uscita il corrispondente valore $y$, come illustrato attraverso la seguente formula:

$$y(x_i) = \varphi(\sum_i^n x_{i^*} w_i + b)$$

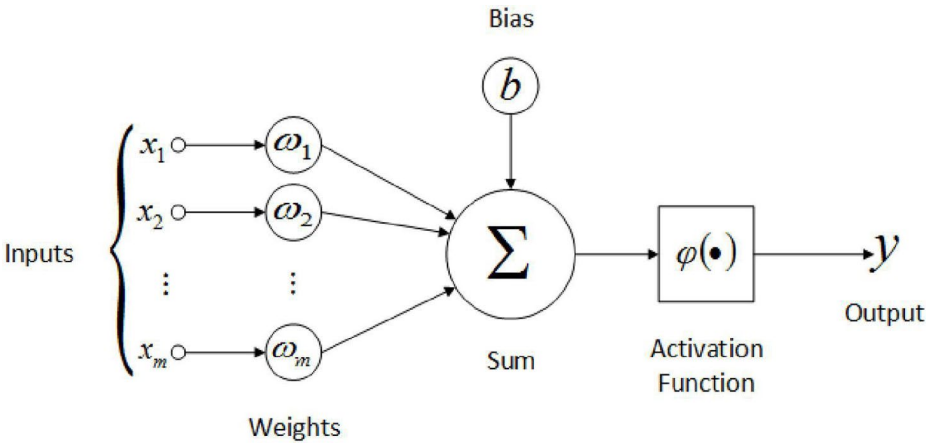



Figura 2.2: Percettrone

Al variare della funzione di attivazione $\varphi$ varia anche l'output corrispondente. Possono essere usate moltissime attivazioni, lineari e non lineari, a seconda del problema da risolvere e della forma degli input. Le più comuni in reti *deep* sono: sigmoide, tangente iperbolica, ReLu e tutte le sue varianti.

L'insieme di più perceptron in parallelo forma un layer. Esso corrisponde ad una funzione $\Phi(x)$ avente in ingresso $n$ input e in uscita un numero di output pari al numero di neuroni che compongono il layer successivo. I layer possono essere combinati per costruire una rete neurale completa, avente $n$ input a formare l'*input layer*, $k$ *hidden layer* formati da un numero variabile di neuroni per ogni layer e un *output layer* dove il numero di output è determinato dal numero di neuroni nel layer. La rete si dice profonda o *deep* all'aumentare del numero di hidden layer $k$. Reti di questo tipo, come i MLP, sono detti *fully connected* nel momento in cui ogni neurone è connesso a tutte le unità dei due layer adiacenti, ad eccezione di input e output layer. In reti con più layer diventano necessarie funzioni di attivazione non lineari come illustrate sopra. Infatti, considerando un neurone come una funzione lineare, la composizione a cascata di funzioni lineari è a sua volta lineare e quindi tutta la rete collasserebbe ad un singolo layer.

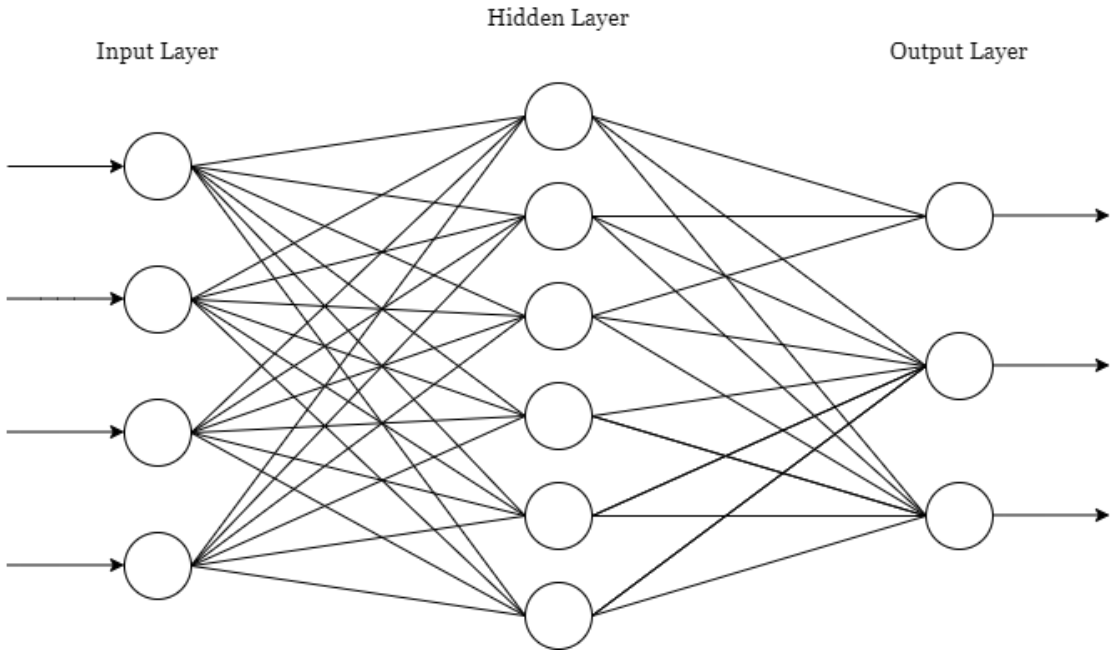



Figura 2.3: Fully connected network

Ciò che rende molto interessanti le reti neurali con almeno un hidden layer è la capacità di fungere da approssimatore universale [3], infatti ogni funzione continua $f(x)$ può essere approssimata come una precisione arbitraria $\varepsilon > 0$ da una rete neurale $g(x)$ secondo la seguente relazione:

$$\forall x,\ |f(x) - g(x)| < \varepsilon.$$

Inoltre, reti con più layer sono in grado di estrarre feature a diversi livelli di astrazione, combinandole tra loro e ricavando importanti informazioni per risolvere il problema assegnatoli. Ad ogni rete viene associata una *loss function* o funzione di costo, simile alle funzioni del machine learning, con la differenza che le reti deep la sfruttano per ottimizzare i propri parametri, vale a dire pesi e bias.

I MLP sono stati usati in molti campi, ma con il passare del tempo sono state messe a punto architetture specifiche per determinati problemi e dati in input. Quando l'input è un'immagine, un dato ad alta dimensionalità, è richiesto un grande numero di neuroni e grande capacità computazionale. Per risolvere questo problema è stato progettato nuovamente il neurone, rendendolo più piccolo e con un numero inferiore di pesi, e con una nuova operazione in sostituzione della somma pesata del percettrone classico. Essa consiste nello scorrimento di una matrice lungo una o più dimensioni dell'input e questo permette di vedere l'intero input senza avere un peso specifico per ogni feature. Le reti che effettuano questo tipo di operazione, definita convoluzione, sono dette *convolutional neural network* o CNN [4]. Queste reti sono storicamente applicate a problemi dove l'input sono immagini o video ma, recentemente, sono state applicate anche a problemi riguardanti le time series facendo scorrere il neurone, detto kernel, soltanto lungo una dimensione.

Solitamente i risultati ottenuti da reti deep sono migliori rispetto a modelli di machine learning. Questo è dovuto al fatto che una rete neurale non è legata a features *handcrafted* create dall'uomo, ma è libera di estrarre le informazioni che ritiene più rilevanti senza alcun vincolo. Ciò porta però a non conoscere esattamente il processo di apprendimento di una rete, rendendola di fatto una *black box* dove conosciamo soltanto input e output. Inoltre, un problema tipico delle reti deep è l'*overfitting*. Questo consiste nell'imparare alla perfezione i dati di training, azzerando la loss function, senza però riuscire ad apprendere una relazione generalizzabile a nuovi dati, portando quindi a risultati scadenti sul test set. Da qui si rendono necessari una serie di accorgimenti per "guidare" l'apprendimento di una rete neurale e questi sono dette tecniche di regolarizzazione.

Queste tecniche agiscono su diversi fronti della rete. Generalmente tali tecniche prevedono l'aggiunta di un fattore di penalità, dipendente dai parametri della rete, nella funzione di costo:

$$Loss_{total} = Loss_{original} + \lambda \cdot f_{regularization}$$

Tra le funzioni di regolarizzazione le più famose e utilizzate sono:

- Regolarizzazione L1, che consiste nella somma dei pesi della rete.
- Regolarizzazione L2 o *weight decay*, analoga alla precedente ma con la differenza che vengono presi i quadrati dei pesi.

Questa funzione può essere vista come un compromesso tra la ricerca di parametri piccoli e la minimizzazione della funzione di costo, guidato dal parametro $\lambda$. Un'altra tecnica molto popolare consiste nel modificare la rete stessa, in particolare rimuovendo in maniera casuale alcuni neuroni dagli hidden layer ad ogni epoca di training. Questa tecnica è chiamata *dropout* ed equivale a limitare la potenza computazionale della rete ad ogni epoca in maniera diversa, ottenendo quindi tante configurazioni diverse pari al numero di epoche. Va tenuto presente che questo procedimento è però applicato soltanto in fase di allenamento: durante la valutazione la rete è utilizzata nella sua interezza. Infine, un'ultima tecnica è chiamata *early stopping*: essa consiste nel valutare la loss function su un set di dati chiamato *validation set* al termine di ogni epoca, in modo tale da poter riconoscere quando la rete smette di generalizzare i dati in ingresso e ha inizio l'overfitting.

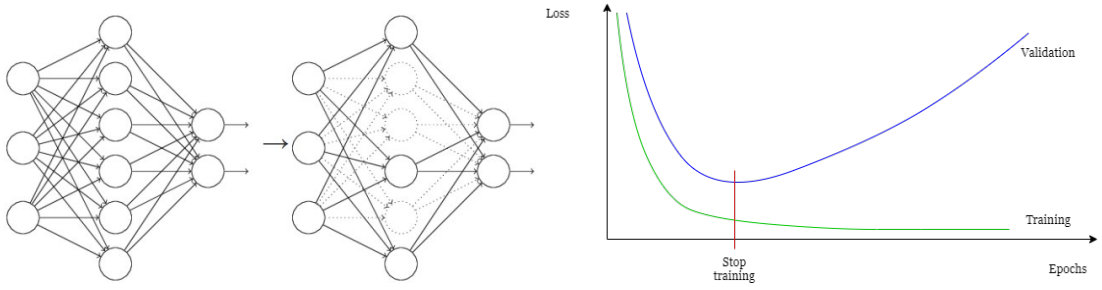



Figure 2.4 e 2.5: A sinistra un esempio di dropout. A destra il funzionamento dell'early stopping

Rimane infine la valutazione dell'apprendimento della rete, trattata successivamente nel capitolo 2.4.

## 2.3 Continual Learning

I sistemi computazionali che operano nel mondo reale sono esposti a flussi di informazioni continue ed è richiesto loro di imparare e ricordare *task* multipli da questi. L'abilità di un sistema di apprendere attraverso il tempo acquisendo nuove conoscenze, senza dimenticare quelle precedenti è nota come *continual* o *lifelong learning*. Questo tipo di problema è storicamente una sfida per sistemi di machine learning e, più in generale, per lo sviluppo di sistemi di intelligenza artificiale [5]. I modelli tradizionali di Machine e Deep Learning non sono adatti ad affrontare tematiche di continual learning a causa della loro natura a dimenticare e sovrascrivere la conoscenza appresa. Infatti, allenando un modello tradizionale con dati provenienti da un task o distribuzione nuova esso sovrascriverà la conoscenza acquisita in precedenza attraverso l'aggiornamento dei parametri della rete. Questo fenomeno è noto come *catastrophic forgetting* [6]. Esso tipicamente conduce ad un repentino crollo delle prestazioni o, nel peggiore dei casi, la totale sovrascrittura del vecchio task nei confronti del nuovo. Se allenate su task sequenziali, le performance delle reti neurali tradizionali decrescono nei task passati all'aumentare del numero di task. Una possibile soluzione, seppur estremamente inefficiente dal punto di vista del tempo e dei costi, è il *re-training* della rete per evitare il forgetting, ma questo inoltre non tiene conto dell'ordine dei task.

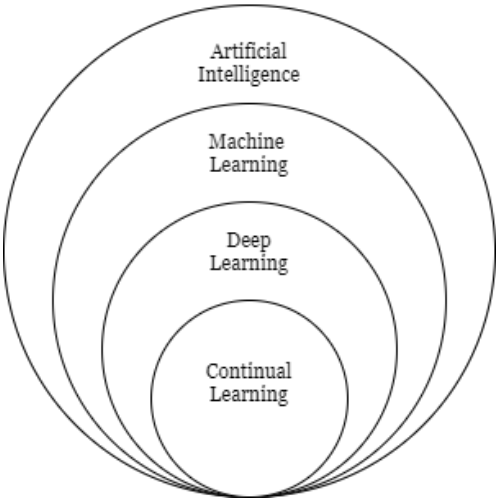


Figura 2.6: Diagramma Eulero-Venn dell'AI

Per non manifestare forgetting, i modelli di deep learning devono soddisfare tre caratteristiche fondamentali: la capacità di acquisire conoscenza da nuovi task, perfezionare la conoscenza acquisita nei task precedenti e prevenire che nuove conoscenze sovrascrivano quelle pregresse. La dualità tra conoscenze nuove e vecchie è nota in letteratura come *stability-plasticity dilemma* [7]. Infatti, un modello deve esibire plasticità nell'acquisire nuove conoscenze e al contempo deve dimostrare stabilità riguardo quelle dei task passati.

In un contesto di continual learning è importante anche definire con precisione il concetto di task, cioè una isolata fase di training di una batch di dati, appartenenti ad un gruppo di classi, dominio o spazio di output. La tipologia di learning continuo varia in base alle distribuzioni $P(Y^t)$ e $P(X^t)$ del task *t*, generalmente con $P(X^t) \neq P(X^{t+1})$:

- *Task incremental learning*: $\{Y^t\} \neq \{Y^{t+1}\}$ con $P(Y^t) = P(Y^{t+1})$
- *Domain incremental learning:* $\{Y^t\} = \{Y^{t+1}\}$ con $P(Y^t) = P(Y^{t+1})$
- *Class incremental learning:* $\{Y^t\} \subset \{Y^{t+1}\}$ con $P(Y^t) \neq P(Y^{t+1})$

Di conseguenza varia anche il task da risolvere, con incremento crescente di difficoltà (Tabella 2.1).

| ***Scenario*** | ***Task*** |
|---|---|
| Task-IL | Risoluzione task, task-ID fornito |
| Domain-IL | Risoluzione task, task-ID non fornito |
| Class-IL | Risoluzione task *e* inferenza su task-ID |

Tabella 2.1: Tre scenari per il Continual Learning [8]

### 2.3.1 Paradigmi di learning correlati

La caratteristica chiave del continual learning, cioè il mantenimento di conoscenze pregresse e capacità di espandere la conoscenza per imparare nuovi task, non è sua esclusiva. Esistono altri paradigmi di learning che condividono, totalmente o in maniera parziale, caratteristiche e obiettivi. Tra i più affini possiamo citare *transfer learning, multi-task learning, online learning, reinforcement learning* e *meta-learning*. I primi due sicuramente mostrano più punti in comune siccome si basano sul trasferimento di learning tra task, ma non si occupano di mantenere esplicitamente le conoscenze pregresse. Online e reinforcement learning invece, manifestano un apprendimento continuo anche se spesso inerente ad uno stesso task che si evolve nel tempo. Di seguito una breve rassegna di questi metodi renderà la comprensione più semplice.

**Transfer Learning**. Paradigma molto popolare, specialmente in ambito deep learning. Solitamente coinvolge due domini, source e *target*. Il dominio di tipo source possiede un grande quantitativo di dati di training mentre il dominio target ne possiede in quantità ridotta o non ne possiede affatto. Lo scopo del transfer learning è usare un modello pre-allenato sul dominio source per favorire il learning sul dominio target. In questi casi nulla vieta di utilizzare anche dati di source per il target. La principale differenza rispetto al continual risiede nel fatto che il dominio source non viene mai testato e non viene richiesto di mantenere conoscenza su di esso.

**Multi-task Learning**. Come suggerito dal nome, in questo contesto si apprendono $N$ task contemporaneamente. Questo permette di aumentare le performance usando le informazioni rilevanti condivise tra i task. In fase di test vengono richiesti tutti i task simultaneamente. Viene anche scongiurato il forgetting su singoli task e viene favorita la generalizzazione. Rispetto al continual learning viene meno l'aspetto temporale e la sequenzialità dei task, così come l'esigenza di non sovrascrivere conoscenza acquisite in passato, siccome avviene tutto in contemporanea.

**Online Learning**. In questo paradigma si ha un solo dominio e i dati di training vengono processati uno alla volta. Questo consente un continuo aggiornamento del modello. L'obiettivo è quello del learning classico, cioè massimizzare le performance sul test di set. Tuttavia, l'online learning non vieta che dati visti in precedenza possano essere riutilizzati, mentre in un ambiente di continual l'ordine temporale, anche interno al task, è rilevante.

**Reinforcement Learning**. Sono problemi tipici di agenti autonomi in grado di muoversi liberamente all'interno di un ambiente dinamico per imparare un task. Ad ogni step l'agente riceve in input lo stato corrente ed un possibile *reward* riguardante la prossima azione tra un pool di possibili azioni da compiere. Ogni azione cambia l'ambiente e in questo modo l'agente impara una traiettoria verso l'obiettivo. Obiettivo che consiste nell'imparare una *optimal policy* che effettua il mapping ambiente-azione e massimizza la *future expected reward* [9].

**Meta-Learning**. Il *meta-learning* è un processo che usa metadati provenienti da esperienze passate in modo tale da migliorare la capacità di learning per nuove esperienze. Come nel continual, esso si basa sull'idea che il processo di learning non è statico ma un processo continuo in evoluzione. Ciò nonostante, la ricerca si è quasi totalmente focalizzata sul migliorare le performance in tasks già definiti ed oggi è usato come fase di pre-learning per avere un migliore punto di partenza nel processo di training.

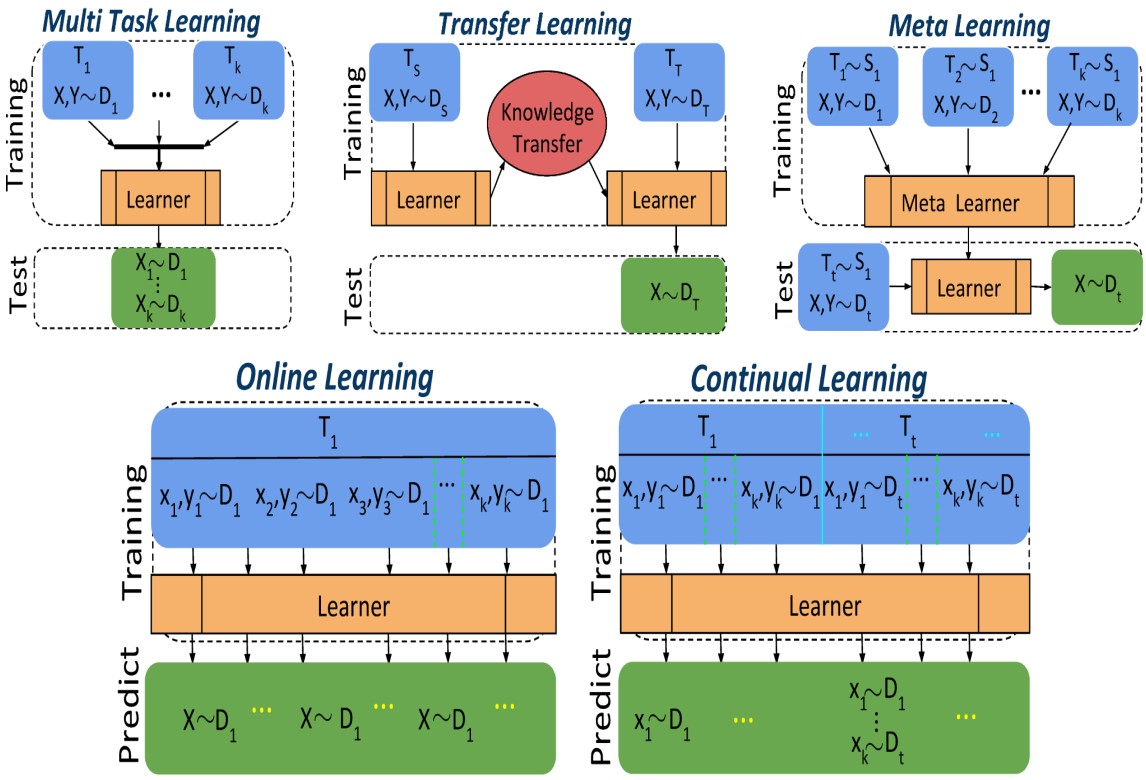



Figura 2.7: Principali paradigmi di learning [10].

### 2.3.2 Strategie

In questa sezione vengono brevemente illustrate le principali strategie per affrontare problemi di continual learning. I singoli approcci testati verranno affrontati all'interno del capitolo 4.

È necessario innanzitutto precisare che non esiste un algoritmo di continual learning assoluto valido per tutti i problemi, ma in ambito di ricerca sono stati presentati metodi che, al momento della stesura di questo elaborato, rappresentano lo stato dell'arte per le tematiche affrontate nella ricerca, vale a dire classificazione di immagini in dataset noti e usati come benchmark. Inoltre, ogni "famiglia" possiede requisiti di memoria, potenza computazionale e scalabilità che le rendono più o meno applicabili in base al problema da affrontare. Possiamo dunque distinguerne tre:

- Metodi *replay-based*
- Metodi *regularization-based*
- Metodi di *parameter isolation*

I metodi *replay-based*, noti anche come *rehearsal*, salvano esempi provenienti dai task precedenti. Questi esempi vengono utilizzati come input durante il training del task corrente, e possono concorrere o meno alla loss function per prevenire forgetting e interferenza. I metodi di rehearsal non usano tutti i dati dei task precedenti, ma soltanto un subset limitato tipicamente dalla dimensione della memoria usata per salvare gli esempi. Memoria che gioca un ruolo molto importante in quanto essa non deve essere troppo piccola per non mitigare il forgetting ma al contempo non deve eccedere per sfociare in uno pseudo multi-task learning. In assenza di dati di task precedenti le tecniche di *pseudo-rehearsal* raccolgono informazioni riguardante la distribuzione dei dati e, tramite un modello generativo, generano dati fittizi ma affini alla distribuzione del dataset originale. Questi metodi generalmente scalano molto bene ma, per un alto numero di task, possono incorrere in problemi di saturazione della memoria disponibile, se questa di piccole dimensioni.

I metodi *regularization-based* evitano di salvare dati, riducendo i requisiti di memoria richiesti. In compenso, introducono un ulteriore termine alla loss function, opportunamente pesato:

$$L_{global} = L_{task} + \lambda * penalty_{regularization}$$

Tipicamente la componente di regolarizzazione agisce sui parametri $\theta$ del modello, scoraggiando l'aggiornamento di neuroni o layer ritenuti rilevanti per i singoli task. I vari metodi differiscono sul tipo di *penalty* da applicare e come questa viene calcolata. Questo permette di consolidare la conoscenza dei task precedenti, lasciando al modello la possibilità di osservare e apprendere dati provenienti dal task corrente. I metodi di regolarizzazione, salvo rare eccezioni, non sono particolarmente onerosi dal punto di vista della potenza computazionale richiesta ma peccano in scalabilità. Ciò è dovuto al

"congelamento" dei pesi, rendendo impossibile, con un alto numero di task, apprendere correttamente gli ultimi task in ordine temporale.

I metodi di *parameter isolation* o architetturali, infine, dedicano parti del modello o copie di esse in modo esclusivo ad ogni task. In assenza di vincoli sull'architettura, dunque, viene istanziato un nuovo ramo del modello per ogni nuovo task, in modo che ognuno di essi abbia parametri esclusivi. Possono essere esclusivi o condivisi i layer finali del modello coinvolto nella classificazione. Come è facile intuire, i metodi architetturali risultano estremamente costosi dal punto di vista della memoria e potenza computazionale richieste in quanto si hanno $N$ modelli per $N$ task, ma hanno una scalabilità potenzialmente infinita.

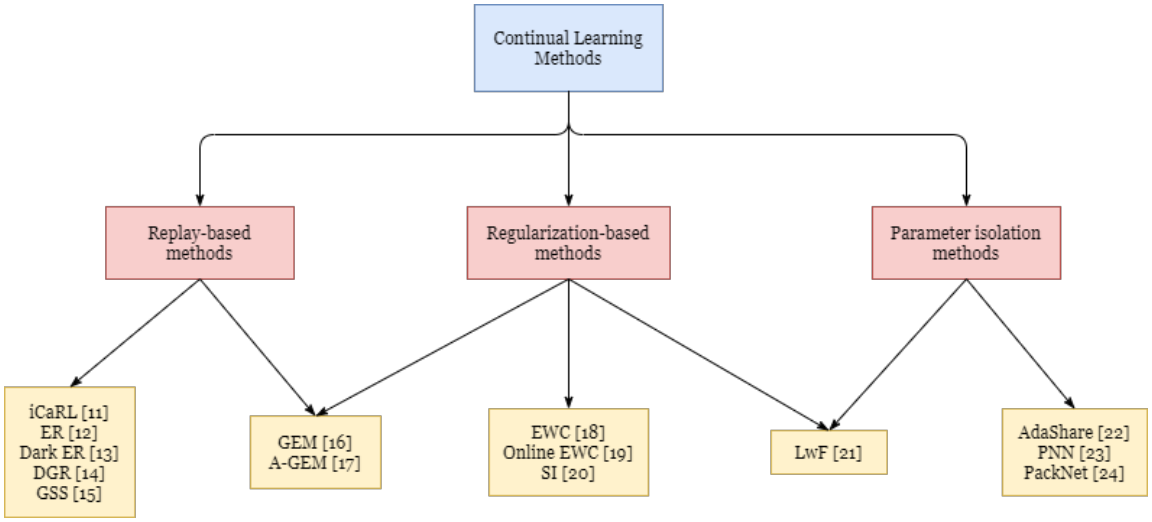



Figura 2.8: Famiglie di continual learning e metodi.

## 2.4 Metriche di valutazione

Durante la fase di testing, per valutare correttamente la qualità del learning, è necessario valutare sia i singoli task che l'intero processo. Mentre per i task presi singolarmente possiamo usare le metriche standard del machine learning, come l'accuracy, per quanto riguarda le metriche di continual è necessario definire metriche ad hoc, in grado di quantificare il forgetting e il transfer learning. Lopez-Paz e Ranzato [16] hanno espanso il concetto di transfer learning con backward transfer (BWT) e forward transfer (FWT). Il primo indica come un task $t$ influenza le performance di un task precedente $k \preccurlyeq t$. Possiamo definire un positive backward se il learning del task t porta ad un incremento delle performance del task k. Al contempo, si osserva un negative backward transfer in caso di peggioramento delle performance. Quando il negative backward transfer assume un valore molto alto allora ci troviamo in una situazione di catastrophic forgetting. Il secondo invece, indica come un task t influenza le performance

di un task k successivo con $k \succ t$. In particolare, un positive forward transfer è possibile quando il modello è capace di effettuare uno "zero-shot" learning.

Definiamo invece forgetting la quantità di conoscenza "persa" a causa dell'insorgere di nuovi task da apprendere in un contesto di sequential learning. Applicato ad un problema di continual learning, forgetting e backward transfer si equivalgono. Possiamo calcolare queste metriche come segue:

**Average Accuracy**: $\text{ACC} = \frac{1}{T}\sum_{i=1}^{T} R_{T,i}$

**Backward Transfer**: $\text{BWT} = \frac{1}{T-1}\sum_{i=1}^{T-1} R_{T,i} - R_{i,i}$

**Forward Transfer**: $\text{FWT} = \frac{1}{T-1}\sum_{i=2}^{T} R_{i-1,i} - \underline{b}_i$

con la matrice $R \in R^{T\,x\,T}$, dove $R_{i,j}$ indica l'accuracy sul test set del modello sul task $t_j$ dopo averlo allenato sul task $t_i$, e il vettore $\underline{b}$ è ottenuto dall'accuracy di ogni task con una randomizzazione iniziale, vale a dire ottenuto all'inizializzazione della rete. Il FWT è calcolato come la differenza tra accuracy prima di allenare un task e l'accuracy calcolata con un'inizializzazione casuale della rete, e successivamente viene effettuata la media tra tutti i task. Se da un lato il forward learning di un task può essere utile, dall'altro in un contesto di continuous learning si ha la certezza che prima o poi tutti i task verranno allenati e quindi è più significativa l'accuracy misurata al termine del training di ogni task. Da considerare, inoltre, che l'accuracy calcolata con pesi casuali dipende fortemente dal tipo di inizializzazione scelta. BWT e forgetting invece calcolano la differenza tra l'accuracy al termine del task corrente e quella migliore per quel task (presumibilmente ottenuta al termine del training dello stesso). È piuttosto semplice ottenere un BWT alto ma bassa accuracy finale al termine del training; ciò non è raro che si verifichi specie in strategie di regolarizzazione. Infatti, in questa famiglia di strategie, è plausibile che un eccessivo fattore di regolarizzazione impedisca il corretto training dei task futuri. L'assunzione che l'accuracy massima di un task si ottenga al termine del training di esso non è sempre verificata. Ciò accade soprattutto in strategie di rehearsal dove gli esempi di task passati sono disponibili anche in quelli successivi, non fermando mai di fatto il learning di task passati. Per concludere, una buona strategia di continuous learning dovrebbe prediligere un BWT il più possibile vicino allo zero o positivo, anche a discapito di un FWT non molto soddisfacente.

## 2.5 Changepoint detection

Cambiamenti repentini nel comportamento di una serie temporale sono spesso segnali d'allarme importanti in quanto potrebbero portare ad una variazione nella distribuzione che genera i dati. Questi momenti sono noti come *changepoints*, e la loro rilevazione è importante in molti ambiti, dall'ingegneria alla medicina. Effettuare changepoint detection può essere visto come partizionare una sequenza temporale in intervalli che presentano caratteristiche comuni, solitamente dal punto di vista statistico.

I changepoint possono avere natura molto diversa tra loro, così come la fonte che li genera. Infatti, essi possono essere generati da eventi interni (rilevabili all'interno della distribuzione dei dati) oppure da eventi esterni. Gli algoritmi di changepoint detection sono classificati in base alla quantità dei dati che analizzano per ogni step, vale a dire alla dimensione della finestra d'osservazione. Gli algoritmi offline possiedono una finestra di dimensioni uguali all'intera serie temporale. Questi sono liberi di scorrere la serie in entrambe le direzioni e sono trattati come problemi di learning supervisionato in quanto conoscere l'andamento nel futuro della serie equivale a conoscere la label in un classico problema di classificazione. Dall'altre parte gli algoritmi online hanno una finestra di dimensione finita e quindi possiede una conoscenza limitata della serie temporale. È necessario quindi che la finestra scorra lungo l'asse temporale, solo in avanti, per rilevare correttamente i changepoint. Changepoint che risultano meno accurati negli algoritmi online in quanto è necessario analizzare anche N datapoint successivi al changepoint per avere la certezza di quest'ultimo. Con queste basi possiamo definire gli algoritmi anche in base al numero di datapoint necessari per confermare la presenza di un changepoint. Un algoritmo offline è considerabile ∞-real time in quanto osserva tutti i datapoint successivi al changepoint mentre uno online, dal punto di vista teorico è 1-real time [25]. Ma, come detto poc'anzi, nessun algoritmo è in grado di rilevare istantaneamente una variazione nella distribuzione della serie temporale; pertanto, possiamo generalizzare definendo gli algoritmi online $\varepsilon$-real time. Valori piccoli di $\varepsilon$ renderanno gli algoritmi più robusti e soprattutto responsivi ai cambiamenti. Questa quantità dipende dalla natura dell'algoritmo e dal tipo di valutazione effettuata per la detection.

In base a ciò esistono molte categorie di algoritmi di changepoint, ma non è scopo di questa sezione illustrarle tutte. Questa tesi si focalizza sugli algoritmi probabilistici, che dal punto di vista teorico offrono la reattività maggiore nella detection tra quelli online, come illustrato nella figura 2.7.

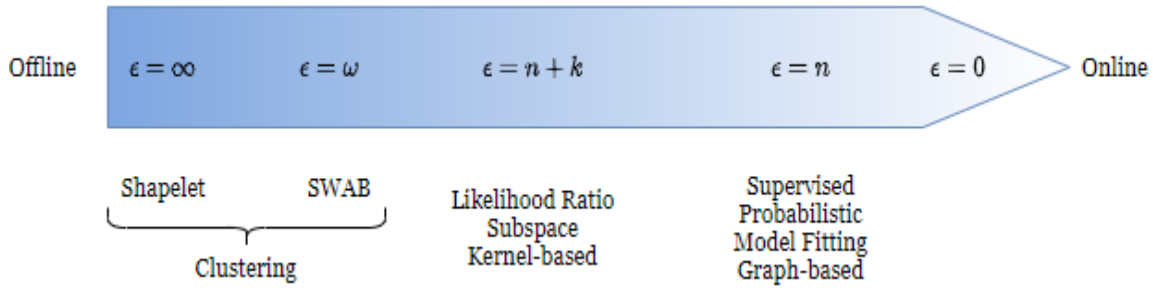



Figura 2.9: Algoritmi di changepoint detection

# Capitolo 3

# Continual Learning e Financial Time Series

Con la metodologia seguente si vuole allenare un modello di Continual Learning (CL) ad apprendere *N* task di classificazione per predire l'andamento di mercato di una serie storica finanziaria. Per fare ciò il modello riceve in input i dati storici del mercato, rappresentati dai dati grezzi contenenti solo il prezzo della *stock* o *commodity* e successivamente ingegnerizzati per ottenere nuove features. Queste sono degli indicatori finanziari comunemente usati nel mercato e nulla vieta di usarne altri o generarne di nuovi. Successivamente, grazie ad un algoritmo di online changepoint detection, verranno definiti dei regimi ed essi comporranno i task del problema di CL. Questa operazione permette di rilevare gli andamenti storici del mercato in modo tale che, nel caso in futuro si ripresenti un pattern già visto, il modello sarà in grado di effettuare correttamente le predizioni. L'utilizzo di pattern passati per predire l'andamento futuro del mercato è ad oggi ancora un topic di ricerca e sono state proposte molte metodologie di deep learning per affrontare il problema. Le *recurrent neural netowrk*, o RNN, sono tra le reti più utilizzate [30] in quanto, grazie ad un hidden state capace di salvare *T* stati passati al suo interno, mantengono memoria di *T* timestep o input passati. Esse inoltre aggiornano lo stato interno basandosi sull'input corrente e lo stato al timestep precedente e calcolano l'output basandosi esclusivamente sullo stato corrente, come illustrato di seguito:

$$h_t = f(U_{xh} * x_t + V_{hh} * h_{t-1})$$
$$y_t = W_{hy} * h_t$$

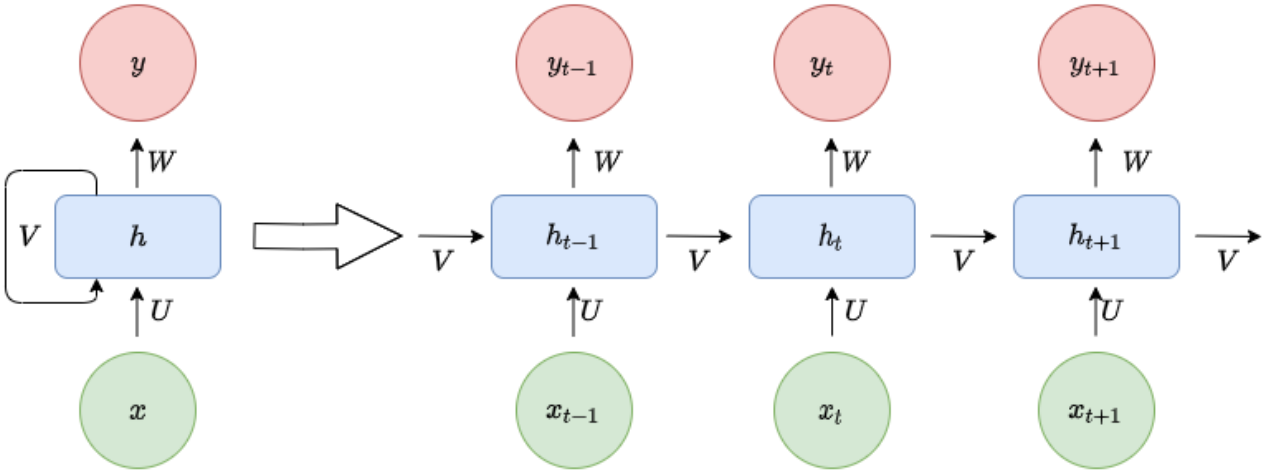



Figura 3.1: Schema di una Vanilla RNN

La loro capacità di memorizzare il passato è dettata dalla dimensione dell'hidden state e, mantenere memoria di pattern mille o diecimila timestep nel passato comporterebbe una memoria e capacità computazionale impensabili e non efficienti.

L'uso di tecniche di continual learning applicate a time series finanziarie è un topic mai affrontato in letteratura e pertanto non si ha prova della loro efficacia in tale ambito. In questo capitolo si esaminano a fondo le tecniche utilizzate durante lo svolgimento della tesi, l'ambiente in cui essa è stata svolta e i dati a disposizione. Verranno inoltre illustrate le problematiche affrontate e le possibili tecniche di risoluzione implementate per queste ultime.

## 3.1 Definizione del problema

I classici problemi di continual learning non trattano time series, ma bensì immagini. Di conseguenza il primo passo da effettuare è stato capire quali algoritmi potessero essere applicati, tenendo conto che dati in input di natura diversa portano ad assunzioni differenti. La prima, fondamentale, è l'aspetto temporale della serie finanziaria. In questo modo l'ordine dei task è obbligato e le performance di essi sono necessariamente influenzate. Nonostante ciò, l'impalcatura fondamentale dei problemi di continual learning rimane valida e quindi si è potuto procedere verso un migliore studio del problema.

Come illustrato nella sezione 2.3, il continual learning è definito per diversi scenari (task, domain e class-IL) e lo step successivo è stato quello di individuare in quale di questi si trovi il problema trattato in questa tesi. Siccome tutti i task provengono dalla medesima serie storica anche la distribuzione dei dati di ogni task sarà uguale; pertanto, possiamo scartare lo scenario di *class-IL*. Come detto nelle sezioni precedenti, ci troviamo davanti un problema di predizione del mercato tramite classificazione. Per ogni task il modello riceve in input una sequenza pari ad un mese di dati aggiornati ogni giorno alla chiusura dei mercati, per una lunghezza della sequenza equivalente a 20 giorni. Il motivo di questa scelta è dettato innanzitutto dall'apertura e chiusura del mercato a livello mondiale. Esso, infatti, apre il lunedì e chiude il venerdì della stessa

timezone, a meno di festività. Il secondo motivo della scelta di una finestra di queste dimensioni è dato dall'assunzione che in un mese di osservazioni giornaliere è possibile raccogliere una quantità di dati sufficienti a fare un'analisi attendibile. Essendo learning di tipo supervisionato si ha anche una label associata ad ogni sequenza. Si avrà *y=1* se il valore della serie temporale 20 giorni dopo il termine della sequenza sarà maggiore dell'ultimo dato della stessa, zero altrimenti. Questo corrisponde ad un mese di osservazione e predizione il mese successivo. Così facendo per ogni task i possibili output apparterranno allo stesso dominio e di conseguenza ci troviamo dinanzi ad uno scenario di *domain-IL*. In uno scenario di questo tipo non si hanno informazioni riguardanti un identificativo del task che si sta affrontando e questo ha portato a scartare tutta la famiglia di metodi architetturali o di parameter isolation. Nel problema affrontato, siccome tutti i task avranno ID progressivi per comodità, non avremo la possibilità di verificare che due pattern che identificano due task nella realtà coincidano, se non attraverso le metriche di valutazione e il ragionamento, strumenti non sufficienti per fare una analisi quantitativa accurata.

## 3.2 Librerie e frameworks

In questa sezione sono presentate brevemente le librerie e i *frameworks* usati durante lo svolgimento della tesi, focalizzandosi sulle caratteristiche e motivazioni che hanno portato alla loro adozione.

### 3.2.1 PyTorch

PyTorch [26] è una libreria Open Source per la progettazione e realizzazione di reti neurali scritta nei linguaggi C++ e Python. Essa è stata sviluppata dall'AI Research lab di Facebook nel 2016. Sebbene sia più giovane rispetto ad altre librerie dello stesso ambito, vengono rilasciate versioni nuove periodicamente e viene usata da alcune tra le più grandi società come Tesla o Uber per svolgere compiti di *computer vision* e *natural language processing*. PyTorch è scritta in modo da essere *user-friendly*, modulare e facilmente comprensibile grazie ad un'esaustiva documentazione. La possibilità di effettuare operazioni ad alte prestazioni con i tensori grazie all'uso di GPU e l'automatica creazione di un albero dei gradienti per le reti deep sono solo alcune tra le funzionalità ad alto livello che la libreria offre. Quest'ultima garantisce di risparmiare tempo nella computazione in ogni epoca al momento del forward pass per una migliore efficienza e uso delle risorse a disposizione.

### 3.2.2 Weights & Biases

W&B [27] è una piattaforma dedicata a sviluppatori di Machine e Deep Learning scritta in linguaggio Python. Essa offre un ecosistema leggero, interoperabile e performante per una vasta gamma di operazioni che coinvolgono reti neurali. Facilmente integrabile con PyTorch, permette di tracciare live le fasi di training e visualizzare

mediante grafici e tabelle i layer della rete, oltre che le metriche del modello come loss o accuracy. Permette inoltre di modificare dinamicamente alcuni iper-parametri ed effettuare test rapidi per l'ottimizzazione di essi.

### 3.2.3 TA-Lib

TA-Lib [28] è oggi la libreria di riferimento per l'analisi tecnica dei mercati. Data una serie temporale è possibile estrarre da essa oltre 200 indicatori finanziari e statistici. Offre anche uno strumento di pattern recognition, sempre in ambito finanziario. Scritta in linguaggio C/C++ dispone di wrapper per il linguaggio Python che consente un suo facile utilizzo.

### 3.2.4 Altre librerie

Nel corso della realizzazione della tesi si è reso necessario l'uso di svariate librerie:

**Pandas**. Libreria Open Source scritta in linguaggio Python per la manipolazione e analisi dei dati. In particolare, offre strutture dati e operazioni per manipolare tabelle numeriche e serie temporali.

**NumPy/SciPy**. Librerie Open Source di strumenti matematici e algoritmi in linguaggio Python. In particolare, NumPy aggiunge supporto a matrici e array di grandi dimensioni e ciò la rende molto utile come strumento di integrazione a PyTorch.

**Scikit-learn**. Libreria Open Source di apprendimento automatico. Contiene una vasta gamma di algoritmi di Machine Learning ed è progettata per operare in simbiosi con NumPy e SciPy.

**Matplotlib**. Libreria per la creazione di grafici. Progettata per essere simile a MATLAB, oggi è preferita ad essa in quanto usabile con linguaggio Python e soprattutto gratuita.

## 3.3 Dati e indicatori

Un grande vantaggio del trading mediante algoritmi è la possibilità di processare in *real-time* una grande quantità di dati in modo tale da poter estrarre da essi il maggior numero di informazioni possibili per effettuare la scelta di investimento più opportuna. È necessario fornire al modello dati che rispecchiano al meglio lo stato del mercato in ogni *timestep* per poter agire correttamente. In questa sezione verranno presentati i dati, la loro struttura e le features generati da essi.

### 3.3.1 Serie temporali

I dati forniti al modello in input sono formati da serie temporali rappresentanti, per ogni dato, i valori di un mese di osservazione del mercato. In particolare, i dati utilizzati sia come feature non ingegnerizzate (prezzo), sia per indicatori (trattati nel paragrafo successivo), sono ottenuti giorno per giorno alla chiusura del mercato. Per avere una visione più ampia possibile per ogni serie temporale su cui si voglia fare predizione è necessario considerare l'andamento di essa dalla sua creazione in modo da non escludere possibili pattern passati ma con la possibilità che si ripresentino.

I dati presentati sono di lunghezza fissa (20 giorni) con timestep iniziale corrispondente ad un qualsiasi momento all'interno della serie temporale ed ogni *sample* corrisponde ad una sequenza di giorni diversi con *shift* in avanti di un giorno. Vale a dire che due sequenze consecutive all'interno del training set avranno $\frac{19}{20}$ timestep in comune ed esse differiranno nel primo dato per la prima sequenza e l'ultimo dato per quella successiva.

Nella seguente tabella sono rappresentate le serie temporali usate. Ogni serie è stata usata per ottenere le features e le sequenze temporali costruendo un dataset diverso dando vita ad un'analisi di serie storiche multivariate per ogni serie finanziaria.

| Serie storica | Data di inizio | Data di fine | Numero di timestep |
|---|---|---|---|
| Rame | 02-01-1989 | 31-07-2021 | 8 500 |
| Petrolio (WTI) | 02-01-1986 | 31-07-2021 | 9 282 |
| Indice azionario USA (SP500) | 02-01-1970 | 31-07-2021 | 13 455 |
| Indice azionario Giappone (N225) | 05-01-1965 | 31-07-2021 | 14 758 |

Tabella 3.1: Serie storiche finanziarie usate

### 3.3.2 Indicatori finanziari

Nella primissima fase del progetto sono stati usati i dati ottenuti dalle serie storiche senza alcuna modifica o *feature extraction*. Questi si sono rivelati insufficienti ed addirittura controproducenti. Una rete neurale, infatti, possiede la capacità di estrapolare le informazioni che ritiene rilevanti in maniera del tutto autonoma. In questo caso la rete non effettuava alcun tipo di learning ma si limitava a copiare in output la label dell'esempio precedente, ottenendo così un'altissima accuracy in fase di testing, ma osservando i grafici di correlazione degli output era facile notare una dipendenza tra essi dove nella realtà essi non avevano alcuna dipendenza dimostrabile. Di conseguenza si è reso necessario manipolare la serie effettuando diverse operazioni di ingegnerizzazione per ottenere delle features. Il primo passo è stato svolto in direzione dell'eliminazione della copiatura tra gli output, fenomeno non raro nelle serie temporali e che solitamente

viene risolto creando una serie ottenuta come differenza tra datapoints di due istanti consecutivi. Così facendo l'input ha raddoppiato la sua dimensionalità, da un vettore (1,20) si è passati ad una matrice 2x20. L'aumento della dimensionalità è un aspetto di cui tenere conto in un problema di questo tipo: se da un lato solo la serie grezza è insufficiente, dall'altro utilizzare troppe features potrebbe spingere il modello a focalizzarsi su informazioni non necessarie o peggio fuorvianti per il problema da risolvere. Di questo aspetto si è tenuto conto nello scegliere quali indicatori finanziari da usare come features potessero fornire informazioni utili.

Una caratteristica molto interessante degli indicatori scelti è che essi raccolgono informazioni sul passato e le mediano con quelle riguardanti il presente. Questo permette di formare features che rappresentano non solo lo stato attuale, ma anche un insieme di timestep passati utilizzati per costituire la feature stessa. Molti indicatori sono caratterizzati da un periodo di osservazione nel tempo della serie. Quelli elencati di seguito sono stati generati per periodi che variano da 5 a 20 giorni, in modo tale da estrarre contenuto informativo sia nel breve che nel lungo periodo. Periodi più brevi potrebbero cogliere informazioni troppo poco specifiche per la sequenza, così come periodi più lunghi fornirebbero informazioni di una finestra maggiore di quella di osservazione del modello stesso.

**Medie Mobili - Moving Averages**

Grazie alle medie mobili è possibile rimuovere una grande quantità di rumore dalle serie analizzate. Esse, infatti, sono in grado di catturare il vero andamento di mercato rimuovendo "picchi" e oscillazioni poco significative. Spesso, infatti, i picchi rappresentano un discostamento dalla tendenza del mercato ed indicando un'evoluzione non rappresentativa della realtà sul lungo periodo.

Le medie mobili possono essere usate principalmente in due modi:

- Prese singolarmente rispecchiano l'andamento del mercato. Variando l'intervallo di tempo è possibile eliminare il rumore ed evidenziare la tendenza del mercato.
- Medie mobili multiple, basate su differenti finestre temporali, permettono di valutare i punti di *crossover*. Essi sono punti in cui le medie a breve e lungo periodo combaciano, tipico di andamenti con forte pendenza positiva o negativa del mercato.

Le medie mobili usate durante il progetto sono due:

- *Simple moving average*, calcolata come media semplice di datapoints della serie temporale considerando *n* step passati.

$$SMA_i = \frac{\sum_{i=0}^{n} x_i}{n}$$

- *Weighted moving average*, variante della SMA. Assegna un peso ad ogni singolo datapoint. I pesi decadono in modo lineare andando indietro nel tempo. In questo modo gli esempi più recenti avranno un impatto maggiore.

$$WMA_i = \frac{\sum_{i=0}^{n} x_i(n-i)}{\frac{n(n-1)}{2}}$$

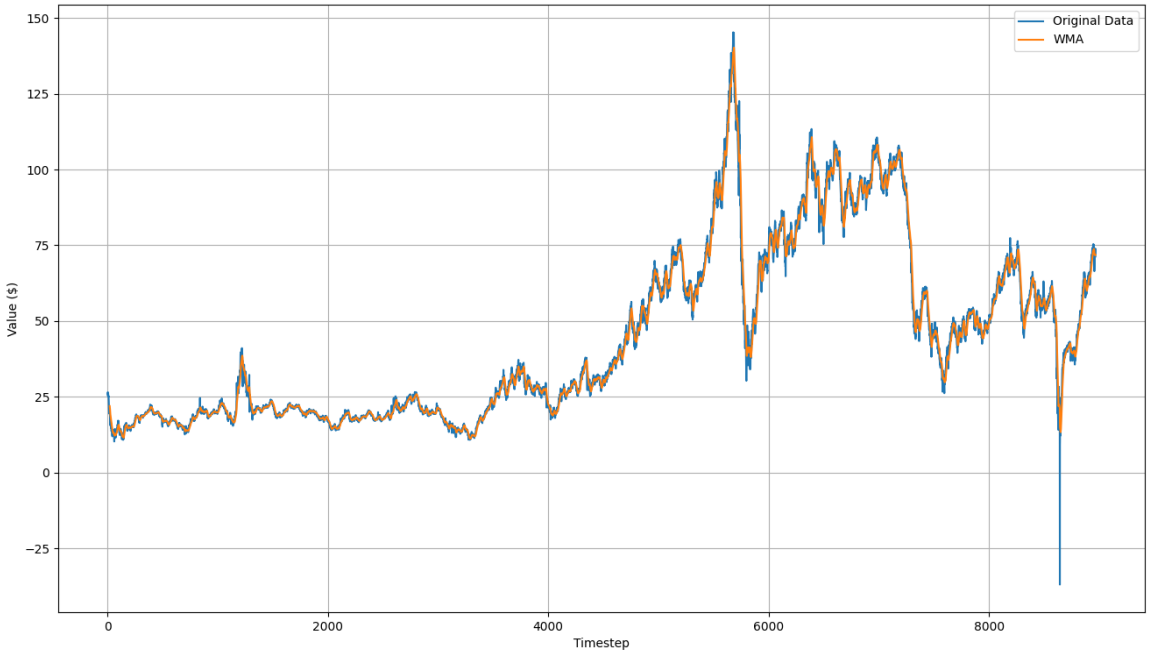



Figura 3.2: Confronto tra serie originale e WMA

Siccome le due features sono simili si è scelto di usarne solo una delle due all'interno del progetto. Il modello è stato testato in entrambe le configurazioni possibili e, in termini di performance, non spicca una configurazione dominante e pertanto ho scelto la WMA come feature di media mobile.

**Rate of Change - RoC**

Il *rate of change* è un indicatore tecnico del momento che misura la percentuale del cambio di prezzo tra il timestep *t* (prezzo attuale) e il prezzo *n* timestep nel passato. Viene calcolato nel seguente modo:

$$ROC_i = 100 \cdot \frac{x_i - x_{i-n}}{x_{i-n}}$$

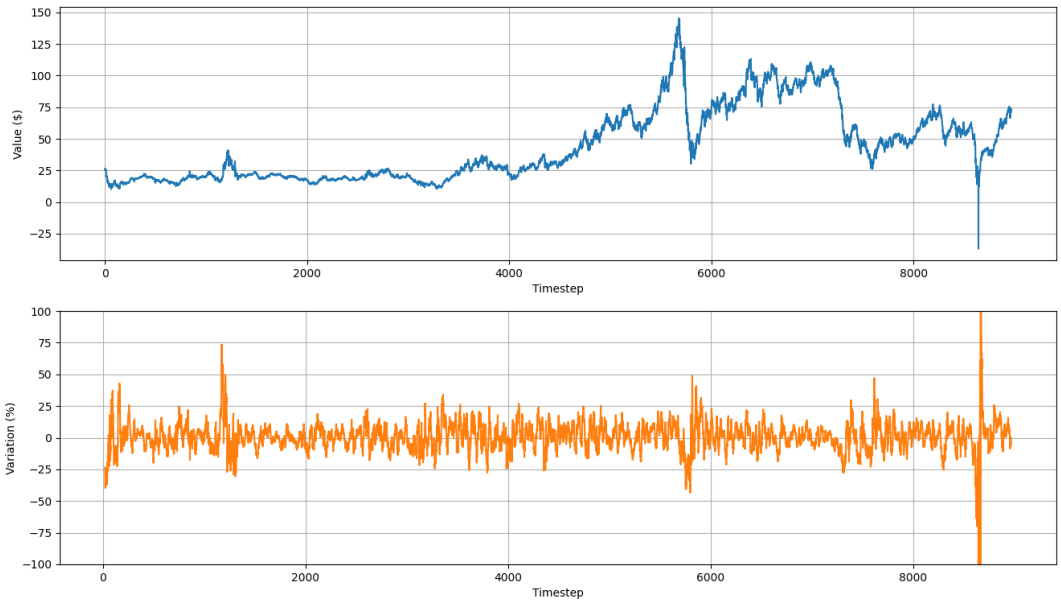



Figura 3.3: Sopra la serie originale, sotto il RoC con la variazione in %

Avendo scelto un intervallo di osservazione molto ampio (tabella 3.2) avremo una alta variazione percentuale molto importante. Questo fatto può essere considerato come una prima analisi qualitativa su possibili changepoint nella serie temporale. Una piccola variazione significa che i dati non si discostano dai precedenti, mentre una variazione importante indica che i dati sono cambiati in maniera significativa rispetto a quelli *n* step nel passato.

### Relative Strength Index - RSI

Il RSI è uno tra gli oscillatori più popolari in finanza. Esso misura la velocità e la magnitudine delle variazioni di prezzo. Stocks con un alto RSI hanno alte variazioni positive nella finestra di osservazione. L'oscillatore è definito in un range tra 0 e 100. In particolare, se una stock ha RSI>70 significa che essa ha subito un rialzo molto rapido ma che rischia di crollare. Al contempo se RSI<30 significa che la stock ha subito un crollo ripido ma che è probabile una sua risalita nel breve periodo. Il valore RSI=70 è detta *overbought line* e suggerisce la vendita mentre RSI=30 è definito *oversold line* e consiglia l'acquisto della stock interessata. Il RSI si basa sulla media delle differenze di chiusura al rialzo e al ribasso dei giorni all'interno della finestra di osservazione e la sua formula è:

$$RSI = 100 - \frac{100}{1 + RS} \quad \text{con} \quad RS = \frac{AVG_{GAIN}}{AVG_{LOSS}}$$

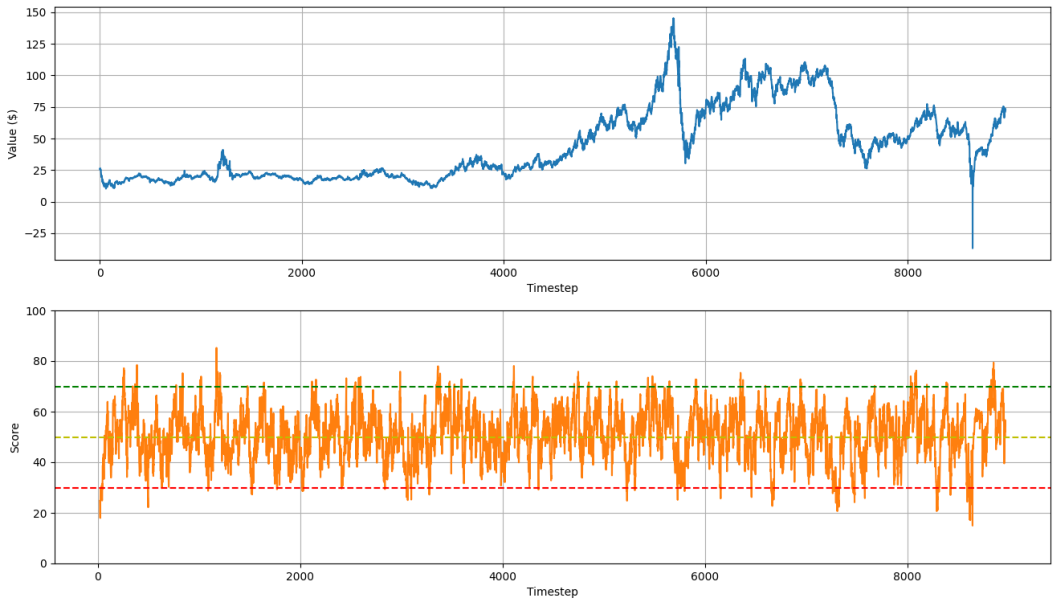



Figura 3.4: RSI con bande di attenzione

I valori dell'indicatore possono essere utili al modello di continual learning. Un valore sottosoglia minima o sopra soglia massima suggerirà al modello che l'andamento della serie non è solido e che molto probabilmente essa cambierà negli step successivi.

### Chande Momentum Oscillator - CMO

Altro famoso oscillatore, derivato dal RSI. La differenza risiede nella sensibilità e precisione, infatti nella stessa finestra di osservazione il CMO individua più facilmente le bande di ipercomprato e ipervenduto. Esso è definito in un range [-100,100] dove l'overbought è fissato a +50 e l'oversold a -50. Il CMO è definito come:

$$CMO = 100 \cdot \frac{AVG_{GAIN} - AVG_{LOSS}}{AVG_{GAIN} + AVG_{LOSS}}$$

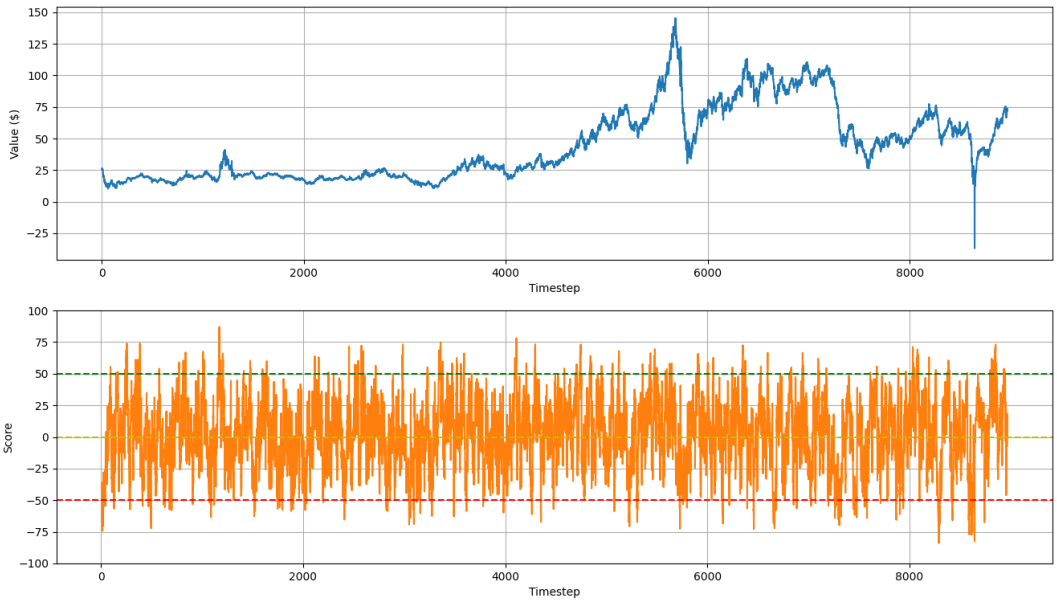



Figura 3.5: CMO con bande di attenzione

Come è possibile vedere osservando le figure 3.3 e 3.4 il CMO è più sensibile rispetto al RSI anche con una finestra di osservazione inferiore. Questo permette di confermare al modello probabili cambiamenti rilevabili con l'indicatore precedente, e al contempo trovarne di nuovi grazie ad una granularità più fine nella valutazione.

**Percentage Price Oscillator - PPO**

Indicatore di momento che illustra la relazione tra due medie mobili con intervalli diversi. Esso viene valutato in termini percentuali nel range [-100,100]. Un valore positivo indica che la media di breve termine si trova al di sopra di quella a lungo termine, esprimendo una configurazione rialzista. Quindi l'algoritmo si sposterà sempre più verso valori rialzisti quanto più la media mobile breve divergerà da quella lunga. Viceversa, accade per valori negativi del PPO. È descritto dalla seguente formulazione:

$$PPO = 100 \cdot \frac{MA_{SHORT} - MA_{LONG}}{MA_{LONG}}$$

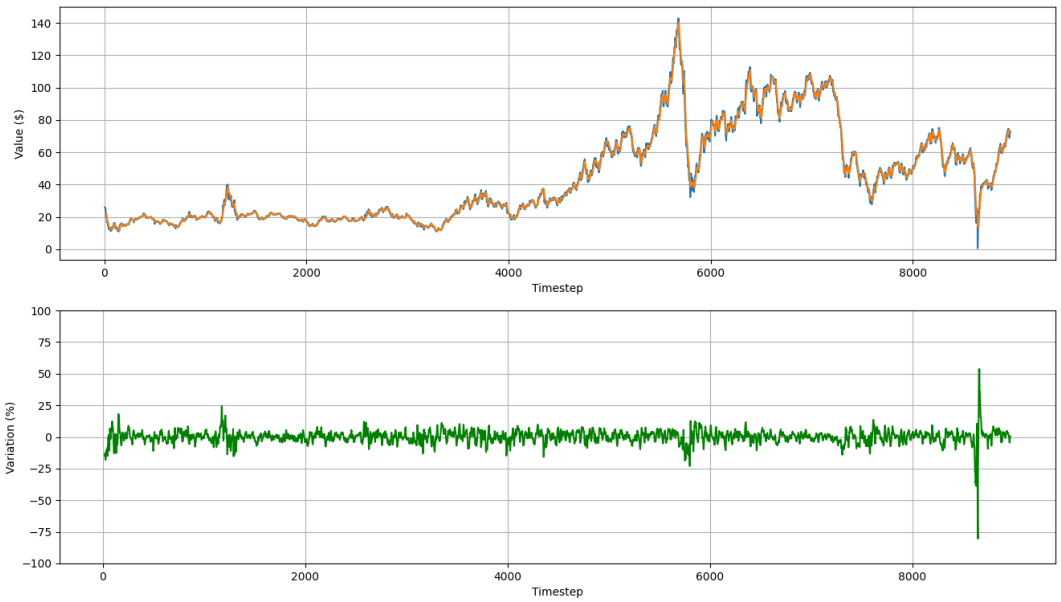



Figura 3.6: Sopra le due medie mobili, short (in blue) e long (arancione). Sotto il PPO

Il PPO è usato tipicamente per comparare le performance e la volatilità del mercato. Inoltre, è un indicatore utile nell'individuare inversioni del mercato ed è in grado di confermare o smentire tendenze di mercato con orizzonti temporali diversi tra loro.

Ogni indicatore è stato testato con differenti intervalli di osservazione tra [5,10,15,20] in modo tale che ogni finestra coincida con una o più settimane di mercato. Nella tabella seguente sono riportati i valori usati nella configurazione finale:

| **Indicatore finanziario** | **Intervallo di osservazione** |
|---|---|
| WMA | 20 |
| RoC | 20 |
| RSI | 5 |
| CMO | 10 |
| PPO | short = 5<br>long = 10 |

Tabella 3.2: Migliori parametri per gli indicatori finanziari

## 3.4 Bayesian Online Changepoint Detection

Come illustrato nelle sezioni precedenti, la fase di changepoint detection risulta particolarmente delicata in quanto da essa dipende la creazione dei domini a partire da una serie storica finanziaria. In un problema di rilevamento di regimi (o andamenti) in una serie in continuo aggiornamento gli algoritmi di detection *offline* non sono applicabili. Essi, infatti, richiedono di osservare l'intera serie prima di effettuare l'analisi e ciò non è possibile siccome la serie storica cresce continuamente. Pertanto, è stato necessario utilizzare algoritmi *online*, in grado di rilevare changepoints in real-time o con un ritardo, misurato in timestep, non significativo rispetto alla lunghezza della serie finanziaria. La *Bayesian Online Changepoint Detection* [29] permette di rilevare changepoints all'interno di una serie storica. Essa si basa su un filtro predittivo causale, un filtro in grado di generare una distribuzione accurata dei dati futuri, ancora ignoti, usando solo i dati già osservati. Assumiamo di avere diverse partizioni $\rho$ di dati $x_1 \dots x_t$. Da ognuna di queste sequenze viene calcolata una distribuzione $P(x_t|n_p)$ con $\rho = 1, 2, ecc$ con $x_t \sim P(x_t|n_p)$ una variabile indipendente e identicamente distribuita e $n_p$ ad indicare la partizione. Indichiamo, inoltre, l'intervallo di osservazioni contigue tra i timestep $a$ e $b$ come $x_{a:b}$. La distribuzione a priori di tutta la sequenza tra due changepoint è indicata con $P_{gap}$.

L'obiettivo è stimare la probabilità a posteriori lungo la *"run length"* corrente, vale a dire il numero di datapoints osservati a partire dall'ultimo changepoint. Una run length lunga *t* timestep è indicata con $r_t$. La figura 3.6 può facilitare la compresione della notazione usata:

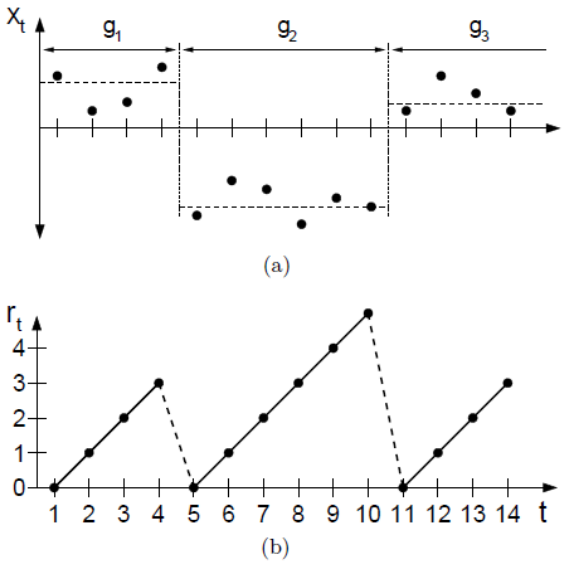



Figura 3.7: Esempio di sequenze e *run length*

Data quindi una sequenza $x_{1:t}$ vogliamo calcolare la probabilità della run length corrispondente $P(r_t|x_{1:t})$, vale a dire la probabilità che all'interno di quella sequenza non ci sia un changepoint.

$$P(r_t \mid \boldsymbol{x}_{1:t}) = \frac{P(r_t, \boldsymbol{x}_{1:t})}{P(\boldsymbol{x}_{1:t})}$$

$$\begin{aligned} P(r_t, \boldsymbol{x}_{1:t}) &= \sum_{r_{t-1}} P(r_t, r_{t-1}, \boldsymbol{x}_{1:t}) \\ &= \sum_{r_{t-1}} P(r_t, x_t \mid r_{t-1}, \boldsymbol{x}_{1:t-1}) P(r_{t-1}, \boldsymbol{x}_{1:t-1}) \\ &= \sum_{r_{t-1}} P(r_t \mid r_{t-1}) P(x_t \mid r_{t-1}, \boldsymbol{x}_t^{(r)}) P(r_{t-1}, \boldsymbol{x}_{1:t-1}) \end{aligned}$$

Queste formule apparentemente complesse permettono di avere un algoritmo ricorsivo basato su due calcoli:

- $P(r_t|r_{t-1})$, ovvero la probabilità a priori di $r_t$ data la run length allo step precedente $r_{t-1}$
- La distribuzione $P(x_t|r_{t-1}, x_t)$ del dato corrente dati tutti gli esempi a partire dal changepoint precedente

La probabilità a priori $P(r_t|r_{t-1})$ è estremamente importante in quanto fornisce all'algoritmo la sua efficienza computazionale. Essa, infatti, assume un valore diverso da zero in due casi, ovvero quando la run length continua a crescere con nuovi esempi oppure si ha una nuova run length subito dopo la presenza di un changepoint.

$$P(r_t \mid r_{t-1}) = \begin{cases} H(r_{t-1}+1) & \text{if } r_t = 0 \\ 1 - H(r_{t-1}+1) & \text{if } r_t = r_{t-1} + 1 \\ 0 & \text{otherwise} \end{cases}$$

dove la funzione $H(\tau)$ è la funzione di *hazard*:

$$H(\tau) = \frac{P_{\text{gap}}(g{=}\tau)}{\sum_{t=\tau}^{\infty} P_{\text{gap}}(g{=}t)}$$

La figura 3.8 illustra il risultato dell'algoritmo ricorsivo. In questa figura i punti vuoti rappresentano l'ipotesi sui run length e le linee indicano il passaggio tra istanti temporali successivi. In particolare, le linee continue indicano la probabilità che la run length continui a crescere, mentre quelle tratteggiate indicano che la run corrente è troncata e si ha un changepoint.

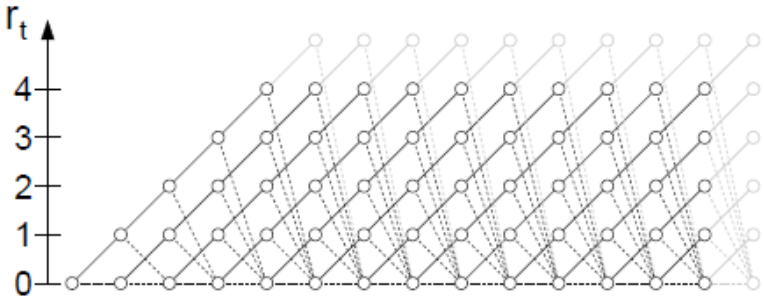



Figura 3.8: Risultato algoritmo ricorsivo

L'algoritmo infine performa la predizione $P(x_{t+1}|x_{1:t})$. Ad ogni datapoint quindi viene assegnata una percentuale che indica la probabilità che il dato successivo ha di appartenere alla sequenza corrente. Al contempo $1 - P(x_{t+1}|x_{1:t})$ è la probabilità che il prossimo dato non appartenga alla sequenza e quindi di avere un changepoint. Ogni dato potrebbe essere dunque un changepoint ed è necessario individuare quelli con percentuale più alta per eleggerli "candidati". Inserendo una soglia decisa dall'utente attraverso un iper-parametro è possibile scartare tutti i punti aventi *score* inferiore ad essa e selezionare solo i punti più papabili. Questo non è però sufficiente a renderli a tutti gli effetti dei changepoints, è infatti necessario che un changepoint sia isolato in un intorno sinistro di punti, vale a dire che sia l'unico candidato in una finestra di punti in cui quello che si sta analizzando corrisponde all'ultimo elemento della finestra. Anche la dimensione della finestra è un iper-parametro deciso dall'utente.

Essendo un algoritmo *online* la BOCD è in grado di rilevare la presenza di un changepoint osservando il minor numero di esempi successivi ad esso per confermarlo. Più breve è la finestra successiva al changepoint più rapido sarà l'algoritmo, ma al contempo una finestra di osservazione più ampia permetterà di avere changepoint più robusti. La finestra di osservazione sui punti passati svolge esattamente questo compito, rendendo certa la detection di un changepoint solo se esso è l'unico punto che soddisfa due condizioni: 1) $P(x_{t+1}|x_{1:t}) \geq th$ e 2) unico punto "candidato" tra gli ultimi $N$ osservati. Scegliere i valori di questi due iper-parametri ($th$ e $N$) non è semplice e possono variare in base ai dati a disposizione e al risultato che si vuole ottenere. Nel caso di questa tesi non ci sono vincoli particolari sul valore della predizione per definire un candidato, ma invece ne esistono per la finestra d'osservazione. Infatti, si vogliono ottenere dei changepoint "forti" che indichino variazioni significative del regime di mercato, scartando brevi picchi momentanei considerabili outlier del mercato. Inoltre, è desiderabile che i changepoint individuati corrispondano il più possibile a cambi di regime del mercato reale, risultato che è stato ottenuto con successo (figura 3.10). La scelta della dimensione della finestra permette di garantire una distanza minima tra i changepoint ma al contempo si incorre nel rischio di un effettivo changepoint scartato perché troppo vicino temporalmente al precedente. La scelta finale è ricaduta su una finestra volutamente ampia per tre principali motivi: 1) una finestra di grandi dimensioni permette di garantire grande solidità ai changepoints; 2)la finestra di grandi dimensioni permette di individuare changepoints lontani tra loro nel tempo e quindi rende possibile rilevare andamenti di medio-lungo periodo e 3) scartando changepoints ravvicinati

permette di evitare di avere task con dataset di dimensioni molto piccole ed eliminare worst cases di sbilanciamento del dataset, fenomeno di per sé non del tutto eliminabile.

Un algoritmo ricorsivo però, non deve occuparsi solo della transizione tra stati ma anche dell'inizializzazione di essi e gli autori dell'algoritmo si focalizzano su due casi. Nel primo, il primo dato temporale osservato corrisponde ad un changepoint, condizione tipica che si presenta all'inizio di una serie temporale. Il secondo, invece, si ha quando si osserva un changepoint all'interno di una sequenza, e pertanto la nuova sequenza possederà i dati successivi al changepoint della sequenza precedente. In questo 'ultimo caso la probabilità della run length viene stimata con una *survival function*. Nell'ambito della teoria della probabilità bayesiana, se due distribuzioni a posteriori $p(\theta|x)$ appartengono alla stessa famiglia della distribuzione a priori $p(\theta)$, queste sono dette coniugate e la distribuzione a priori è detta distribuzione a priori coniugata per la verosimiglianza. Volendo fare inferenza di una distribuzione di dati $x$ con un generico parametro $\theta$ dal teorema di Bayes abbiamo che:

$$p(\theta|x) = \frac{p(x|\theta)\, p(\theta)}{\int p(x|\theta)\, p(\theta)\, d\theta}$$

dove $p(x|\theta)$ è la funzione di verosimiglianza, $p(\theta)$ la distriuzione a priori e $\int p(x|\theta)\, p(\theta)\, d\theta$ è la probabilità dei dati $p(x)$. Tipicamente la funzione di verosimiglianza è fissa poiché viene determinata in base ad ipotesi sulla distribuzione che genera i dati. Di conseguenza la scelta della distribuzione a priori $p(\theta)$ può facilitare o meno il calcolo dell'integrale a denominatore e il prodotto $p(x|\theta)\, p(\theta)$ può assumere determinate strutture algebriche. Se ciò accade e la distribuzione a priori e quella a posteriori hanno la stessa struttura algebrica allora siamo dinanzi ad una distribuzione a priori coniugata [31]. I modelli coniugati, in particolare quelli esponenziali, sono particolarmente convenienti con l'algoritmo di changepoint detection. Il punto di forza di questi modelli è che entrambe le distribuzioni che dipendono da un parametro η possono essere espresse dagli iperparametri υ e χ. L'obiettivo è determinare il parametro η associato ai dati che compongono la run length corrente $r_t$. Per farlo vengono inizializzati i parametri υ e χ ed essi vengono aggiornati ad ogni esempio che compone la run length. L'algoritmo in figura 3.9 illustra tutta la pipeline della Bayesian Online Changepoint Detection usata nella tesi:

1. **Initialize**

$$P(r_0) = \tilde{S}(r) \text{ or } P(r_0{=}0) = 1$$
$$\nu_1^{(0)} = \nu_{\text{prior}}$$
$$\chi_1^{(0)} = \chi_{\text{prior}}$$

2. **Observe New Datum** $x_t$

3. **Evaluate Predictive Probability**

$$\pi_t^{(r)} = P(x_t \mid \nu_t^{(r)}, \chi_t^{(r)})$$

4. **Calculate Growth Probabilities**

$$P(r_t{=}r_{t-1}{+}1, x_{1:t}) = P(r_{t-1}, x_{1:t-1})\pi_t^{(r)}(1{-}H(r_{t-1}))$$

5. **Calculate Changepoint Probabilities**

$$P(r_t{=}0, x_{1:t}) = \sum_{r_{t-1}} P(r_{t-1}, x_{1:t-1})\pi_t^{(r)} H(r_{t-1})$$

6. **Calculate Evidence**

$$P(x_{1:t}) = \sum_{r_t} P(r_t, x_{1:t})$$

7. **Determine Run Length Distribution**

$$P(r_t \mid x_{1:t}) = P(r_t, x_{1:t}) / P(x_{1:t})$$

8. **Update Sufficient Statistics**

$$\nu_{t+1}^{(0)} = \nu_{\text{prior}}$$
$$\chi_{t+1}^{(0)} = \chi_{\text{prior}}$$
$$\nu_{t+1}^{(r+1)} = \nu_t^{(r)} + 1$$
$$\chi_{t+1}^{(r+1)} = \chi_t^{(r)} + u(x_t)$$

9. **Perform Prediction**

$$P(x_{t+1} \mid x_{1:t}) = \sum_{r_t} P(x_{t+1} | x_t^{(r)}, r_t) P(r_t | x_{1:t})$$

10. **Return to Step 2**

Figura 3.9: BOCD [29].


Figura 3.10: Confronto tra changepoint detection e periodi di recessione del mercato su prezzo del petrolio

Calibrando correttamente gli iper-parametri dell'algoritmo di changepoint detection otteniamo un risultato soddisfacente, come nella prima immagine della figura 3.10. Come detto in precedenza, punti di cambio del mercato spesso si manifestano in presenza di crisi o recessioni rilevanti dal punto di vista temporale, cioè crisi che durano un intervallo sufficiente per essere identificate e catalogate come tali. Osservando la figura nella sua completezza possiamo vedere come l'algoritmo rilevi con successo le principali finestre di crisi che danno vita a changepoint. La prima crisi, nel 1990, molto probabilmente non viene rilevata dall'algoritmo in quanto la durata di essa è inferiore alla dimensione della finestra di osservazione dell'algoritmo e pertanto non si ha una variazione nei dati sufficientemente ampia. Un'altra possibile motivazione la si può

trovare in un comportamento non di rado riscontrato dall'algoritmo: esso, infatti, potrebbe trovare changepoints candidati agli estremi della recessione. Per essere confermati però, devono essere unici in un intervallo pari alla finestra di osservazione e ciò non accade in questo caso. Questa casistica si presenta quando non si è effettuato un tuning accurato dei parametri dell'algoritmo, motivazione che porta a sottolineare ancora maggiormente quanto sia importante l'ottimizzazione dell'algoritmo. L'ultima grande crisi dal punto di vista temporale, quella da COVID-19 ancora in atto, avrebbe una durata sufficiente per essere rilevata e confermata come changepoint dall'algoritmo, ma a causa del tempismo della raccolta dei dati non si hanno sufficienti datapoints. Sicuramente se questa analisi venisse ripetuta tra 3-6 mesi ci troveremmo un ulteriore changepoint. Infine, l'algoritmo ha individuato un changepoint non corrispondente ad un periodo di recessione; questo non è risultato di scarsa precisione o un errore nella detection, ma piuttosto un sintomo di forza dell'algoritmo. Infatti, il mercato può variare nell'andamento anche a causa di regolamentazioni o eventi non prettamente finanziari. Questo è il caso del crollo dei prezzi nell'autunno 2014 a causa della decisione dei membri dell'OPEC, l'organizzazione dei paesi esportatori di petrolio nata per contrastare i colossi americani e inglesi e che oggi rappresenta un vero e proprio cartello, di tagliare le produzioni di barili di petrolio in seguito alla riduzione della domanda di esso. Ciò ha portato una variazione nella distribuzione dei datapoints che è stata correttamente individuata dall'algoritmo.

## 3.5 Training, validation e test set

A livello di singoli task che compongono il problema di continual learning è necessario affrontare il problema con il classico metodo utilizzato in contesti di supervised learning. Avendo a disposizione dataset molto grandi ognuno di essi viene diviso in tre parti: training set, validation set e test set. Il dataset viene diviso in modo sequenziale, condizione necessaria quando si lavora con serie temporali, utilizzando la prima parte come dataset di training, la parte centrale come dataset di validation e la parte finale come dataset di test. Durante la fase di apprendimento il modello ha la possibilità di allenarsi sui dati di training, cercando di ricavare un pattern nei dati con le migliori performance su di essi. Al termine di ogni epoca di allenamento viene effettuata una verifica sullo stato dell'apprendimento e della generalizzazione del modello, verificando i pattern sul dataset di validation. Questo check è necessario poiché spesso accade che il modello impari eccessivamente sui dati di training perdendo la capacità di generalizzazione e ottenendo performances peggiori sul validation e test set. Questo fenomeno è detto *overfitting*, e alcune possibili contromisure sono state illustrate nel paragrafo 2.2.2.

L'obiettivo finale, nel caso del training su un singolo task, è apprendere correttamente i pattern visti nel training e verificare la qualità dell'apprendimento sui dati mai visti del test set. Questa caratteristica è fondamentale per quanto riguarda l'affidabilità dei risultati e del modello stesso, cosa necessaria in un ambiente in cui si effettuano investimenti di ingenti somme di denaro che coinvolgono più attori.

**Periodi temporali**

La suddivisone del dataset in tre parti ci pone davanti ad una riflessione sulla grandezza di ognuna di esse. Anche questo parametro è importante poiché è necessario un trade-off tra quanti dati mettere a disposizione al modello per l'apprendimento e quanti possono essere usati per la verifica.

Per scegliere la grandezza degli split bisogna valutare il trade-off tra:

- Training set troppo grande: tipicamente è normale avere dataset di grandi dimensioni in algoritmi che includono reti neurali, poiché le reti tendono a imparare sempre di più all'aumentare dei dati a disposizione, senza un upper-bound dettato dalle feature. Una controindicazione risiede nel punto di vista del tempo di apprendimento: un dataset più grande implica maggior tempo di training di una singola epoca e, siccome in questo problema si effettuano training di centinaia di epoche e soprattutto su più task sequenzialmente, una scelta del genere comporterebbe un aumento del tempo per allenare completamente il modello.
- Training set troppo piccolo: la rete in questo caso potrebbe non avere sufficienti dati per estrapolare pattern efficaci. Per arrivare a convergenza e produrre un modello accettabile di generalizzazione sono necessari grandi quantitativi di dati.

Tenendo conto di questi problemi e considerando le configurazioni più usate in letteratura e non, è stata adottata una configurazione 60-40: 60% dati di training e 40% di valutazione divisi in 20% dati di validation e 20% dati di test effettivi.


Figura 3.11: Split dei dati

Come detto nelle sezioni precedenti, la coppia esempio-label $(x_i , y_i)$ è composta da una sequenza pari a 20 giorni di osservazione e il valore della serie temporale 20 giorni (pari a un mese solare) al termine della sequenza stessa che determinerà il valore della label effettiva:

$$x_i = [s_t : s_{t+20}]$$

$$y_i = \begin{cases} 1 & if \quad s_{t+40} \geq s_{t+20} \\ 0 & else \end{cases}$$

Ogni dataset è costituito da coppie come quella appena descritta, dove la coppia successiva è ottenuta incrementando il timestep dell'inizio della sequenza di 1. Essendo di primaria importanza l'ordine temporale inoltre non è possibile mescolare i dati

all'interno dei vari dataset, ed essi dovranno necessariamente essere presentati al modello nel corretto ordine temporale. Una coppia oggetto-label richiede dunque 40 timestep contigui. Per evitare che alcuni dati vicini ai margini dei dataset siano presenti, per esempio nel training come label e nella validation parte di una sequenza, è consigliato introdurre delle brevi fasce di dati non presenti in alcun set per evitare questo fenomeno, detto *leakage* dei dati. Se presente, questo fenomeno falserebbe le performance in quanto il modello verrebbe testato su dati correlati a quelli di training.

Per una migliore stabilità e validità dei risultati spesso si ricorre ad una tecnica chiamata *cross-validation*. Essa consiste nella divisione del dataset in *n* gruppi di dati, su cui ciclicamente effettuare il training del modello su *n-1* gruppi e la valutazione sul gruppo rimasto escluso, offrendo così una stima più precisa. Non è possibile usare una configurazione di questo tipo per due motivi principali:

- Frammentazione delle serie temporali: i dati appartenenti al mondo della finanza sono intrinsecamente sequenziali. È decisamente deleterio effettuare il training su dati non nel corretto ordine e questo inficerebbe la capacità predittiva dello stesso.
- Tempo computazionale: la cross-validation dal punto di vista computazionale è molto onerosa. Se si effettua un cross-validation a *n* gruppi si avranno *n* differenti training, aumentando il tempo computazionale di ogni singolo task, all'aumentare degli *n* gruppi.

**Fasi dell'apprendimento**

Il training del modello avviene secondo le seguenti modalità. Considerando il dataset come un insieme di sequenze disposte in modo sequenziale, la fase di training si concentra sulla prima parte di esso. Vengono presentate le sequenze al modello in batch di dimensione fissata, e i dati che compongono le batch sono tutti contigui dal punto di vista temporale, senza alcuno *shuffle*. Si forniscono batch al modello fino a concludere il training set, concludendo così un'epoca. Al termine di essa si effettua la stessa operazione sul validation set per verificare la correttezza dell'apprendimento. Ogni 100 epoche di training viene salvato lo stato della rete, inteso come parametri di essa. Questo avviene per poter permettere, in caso di necessità, di valutare lo stato dell'apprendimento sul test set per avere una metrica di performance. Inoltre, al termine del training di ogni task, viene creato un ulteriore checkpoint in modo tale da avere a disposizione tutti i checkpoint al termine di ogni task per eventualmente calcolare metriche di continual learning.

## 3.6 Architetture

Una volta definito il dataset è necessario scegliere la rete neurale da usare come modello. Sono disponibili molte configurazioni possibili considerando il dataset a disposizione dove un dato è composto da un vettore multidimensionale di dimensione [7,30], cioè da 7 features ognuna composta da un vettore di 30 dati. La rete più semplice

da implementare è un multi layer perceptron, come illustrato nella sezione 2.2.2. Possiamo vedere i MLP come la rete base per la risoluzione di problemi di Deep Learning e, poiché a problemi diversi corrispondono reti diverse, è prima di tutto necessario procedere con la personalizzazione della rete, operazione non banale che decreterà la configurazione finale. La dimensione della rete determina inoltre la capacità computazionale e di apprendimento del modello. Una rete piccola rischia di non avere neuroni a sufficienza per la risoluzione del problema, mentre una rete troppo profonda rischia di avere potenza computazionale non necessaria e non sfruttata, o peggio usata per imparare pattern errati e cadere nell'overfitting.

Il primo punto da affrontare è decidere il numero di hidden layer all'interno della rete. Una configurazione senza hidden layer vale a dire con solo input layer e output layer, non potrebbe generare nuove features e dovrebbe valutare i dati così come presentanti in input. Una soluzione di questo tipo è stata scartata a priori perché in grado di rappresentare solamente funzioni linearmente separabili. Se il problema fosse così semplice da richiederebbe un separatore lineare l'uso di una rete deep sarebbe solamente uno spreco di risorse. Il problema trattato in questa tesi è, infatti, un problema complesso che richiede un separatore non lineare. Pertanto, è necessaria la presenza di almeno 1 hidden layer. Nel deep learning difficilmente sono presenti euristiche in grado di fornire linee guida precise sulle soluzioni da adottare. Per la ricerca del numero di layer e di neuroni da usare nella configurazione sono stati usati diversi approcci:

**Letteratura**. Il primo passo è stato ricercare nella letteratura di settore problemi simili trattati con successo e studio delle soluzioni proposte. Questo può essere utile per ottenere delle architetture di partenza da personalizzare secondo esigenza.

**Sperimentazione**. Una volta decisa una rete base si può procedere attraverso test multipli cambiando determinati parametri, o addirittura istanziare una architettura del tutto nuova, alla ricerca di performances migliori.

**Go for depth**. In [4], Bengio e Goodfellow hanno evidenziato come reti profonde generalmente performano meglio. Possiamo cogliere questo come suggerimento incrementando la profondità della rete e testandone l'efficacia.

Tenuto conto di ciò, sono state definite e testate sei configurazioni differenti nel numero di layer e di neuroni al loro interno. Per una migliore comprensione viene usata una notazione molto comune nel descrivere questo tipo di architetture: ad esempio una rete con 2 variabili in input, un hidden layer con 8 neuroni e un output layer con un neurone è espressa con 2/8/1.

**Configurazione Multi Layer Perceptron**

| Architettura | Numero di hidden layer | Configurazione | Numero di parametri allenabili |
|---|---|---|---|
| A | 1 | 210/100/2 | 21 302 |
| B | 2 | 210/80/40/2 | 23 442 |
| C | 2 | 210/210/210/2 | 89 042 |
| D | 3 | 210/100/50/25/2 | 27 477 |
| E | 3 | 210/210/210/210/2 | 133 352 |
| F | 4 | 210/200/150/100/50/2 | 92 602 |

Tabella 3.3: Configurazioni MLP

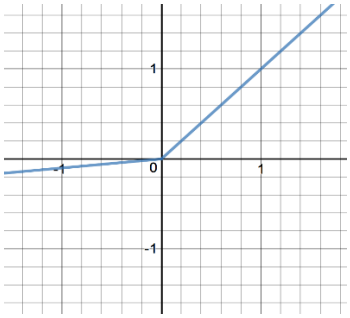
Nelle reti di classificazione binaria tradizionalmente l'output layer è composto da un singolo neurone seguito da una funzione di attivazione di tipo sigmoide. La funzione, infatti, ritorna un valore bounded nel range [0,1]. In questo modo se il risultato è inferiore a 0.5 l'input verrà associato alla classe A, altrimenti B. Questo approccio però non consente di avere dei valori non scalati (*logits*) prima della funzione di attivazione per ogni classe, valori richiesti in diverse tecniche di continual. Per questo ho optato per un output layer a due classi, dove tipicamente viene applicata la funzione softmax per ottenere la probabilità che il dato in input appartenga alle varie classi. Il softmax riceve in input un vettore di lunghezza *N*, dove *N* è il numero di classi di output (2 in questo caso specifico) e ritorna un vettore della stessa dimensione con la percentuale per ognuna delle *N* classi secondo la relazione

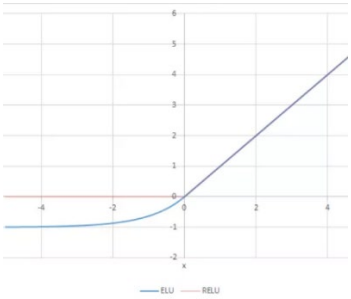
$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

Una funzione di tipo softmax con un vettore di lunghezza 2 equivale a mappare i valori in input nel range [0,1] e potrebbe portare ad una perdita di informazioni; pertanto, nella configurazione finale non verrà usata alcuna funzione di attivazione finale per ottenere le percentuali di appartenenza ad ogni classe ma verranno usati direttamente gli output dei neuroni dell'output layer.

Come illustrato nelle sezioni precedenti le reti neurali richiedono delle tecniche di regolarizzazione per evitare il fenomeno dell'overfitting e ognuna delle configurazioni proposte non è da meno. Ad ogni hidden layer viene applicato il dropout riducendo così la capacità di ogni layer in maniera randomica in fase di training. Nonostante ciò, alcune configurazioni (C, E ed F) hanno manifestato un alto overfitting e di conseguenza sono state scartate. Tra le configurazioni rimaste la D è stata scelta come configurazione finale in quanto offre prestazioni soddisfacenti, mentre A e B hanno troppi pochi parametri per effettuare un corretto learning con tecniche di regolarizzazione e rimuovendo queste manifesterebbero comunque overfitting.

L'ultimo aspetto da definire per concludere la personalizzazione della rete è la scelta della funzione di attivazione per gli hidden layer, funzione che deve necessariamente essere uguali per tutti i layer. Le funzioni possibili sono quelle già illustrate per il singolo neurone, a cui se ne aggiungono alcune derivate dalla ReLu. Sigmoide e tangente iperbolica sono state subito scartate in quanto ritornano dei valori bounded e di conseguenza portano ad una perdita di informazione. Anche la ReLu classica presenta il problema di perdita di informazioni, in particolare se i valori in input sono negativi la funzione ritorna zero. Le sue due varianti più famose, ELU e Leaky ReLu, risolvono questo problema e sono le due candidate per la funzione di attivazione finale. La differenza tra le due funzioni consiste nel trattamento di valori negativi: ELU usa un esponenziale mentre LeakyReLu una retta con pendenza α, come illustrato in figura 3.12. Dal punto di vista delle performance le due funzioni sono molto simili e di conseguenza ho optato per la Leaky ReLu siccome non ha un comportamento asintotico per i valori negativi.




$$LeakyReLu\ (x) = \begin{cases} x & if\ x > 0 \\ \alpha * x & if\ x \leq 0 \end{cases} \qquad ELU\ (x) = \begin{cases} x & if\ x > 0 \\ \alpha * (e^x - 1) & if\ x \leq 0 \end{cases}$$

Figura 3.12: LeakyReLu e ELU

È possibile utilizzare anche altre tipologie di reti per effettuare classificazione con le serie temporali e tra queste troviamo le CNN. Anche se sono reti nate per processare matrici N-dimensionali come le immagini possono essere utilizzate per elaborare alcun tipo di dato grazie all'operazione che le contraddistingue, la convoluzione. Dal punto di vista matematico la convoluzione consiste nell' applicare una maschera (o *kernel*) ad una funzione in modo tale da ottenere come risultato l'unione tra questi due fattori. Nel machine learning il kernel assume il ruolo di un filtro in grado di catturare le caratteristiche desiderate dal tipo di kernel. Grazie all'utilizzo di diversi filtri all'interno di un layer e di molteplici layer è possibile ottenere diversi livelli di features a diversi livelli di astrazione, partendo da features molto specifiche nei primi layer fino a individuare features generiche o pattern completi negli ultimi. L'obiettivo di una CNN è di ricavare un vettore di features $[1,1,n]$ partendo da un'immagine $[h,w,c]$, riducendo quindi la dimensione spaziale ed aumentando la profondità di canale fino ad ottenere un unico vettore di features in grado di descrivere l'intera immagine. Questo vettore sarà poi passato ad un classificatore, simile a quelli usati nelle altre reti neurali, per ottenere un output. I layer che realizzano la convoluzione, detti convolutivi, assieme a layer di pooling sono i principali responsabili del cambio di dimensionalità dei dati.

Un processo simile può essere applicato a serie temporali. Se nelle immagini la convoluzione avviene su due dimensioni nelle serie temporali questo può accadere solo sulla dimensione temporale. In questo modo il kernel può scorrere lungo il vettore riducendone la dimensione ed estrapolando le features principali. Nel definire l'operatore di convoluzione è necessario specificare alcuni parametri:

- *Channels_out* ($C_{out}$): Il numero di canali dell'output, essi corrispondono al numero di kernel di cui è composto il neurone e al numero features prodotte dall'operazione.
- *Kernel_size (k)*: La dimensione del filtro che dovrà scorrere sui dati, filtri di grandi dimensioni saranno in grado di osservare più dati e rilevare caratteristiche su un campione maggiore di dati.
- *Stride (s)*: Quanto il filtro deve spostarsi tra una convoluzione e la successiva, assieme alla dimensione del kernel è il principale responsabile della riduzione della dimensione temporale.
- *Padding (p)*: Permette di inserire una "cornice" di dati per far combaciare la dimensionalità risultante dalla convoluzione.
- *Dilation (d)*: Permette di incrementare il *receptive field* del kernel senza aumentarne però la dimensione, consentendo di escludere alcuni dati dall'operazione.

Sebbene sia possibile applicare *dilation* e *padding* alla convoluzione-1D dal punto di vista teorico, farlo sarebbe un errore in quanto il padding introduce nuovi dati, tipicamente con valore zero o copie del dato più vicino, senza alcun collegamento con la sequenza reale. La dilation invece permette di escludere dei dati dall'operazione della convoluzione ed anche questo sarebbe deleterio siccome verrebbero meno relazioni tra dati contigui. Data una sequenza con shape $[L_{in}, C_{in}]$ il risultato della convoluzione sarà un dato avente shape $[L_{out}, C_{out}]$ con $C_{out}$ specificato dall'utente e

$$L_{out} = \frac{L_{in} + 2 * p - d * (k - 1) - 1}{s} + 1$$

Non utilizzando padding e dilation è stato dunque necessario utilizzare *kernel_size* e *stride* per ridurre la dimensionalità temporale.

In base a queste considerazioni sono stati proposti tre modelli dove differiscono i due parametri sopra citati, mentre la funzione di attivazione tra i layer è la medesima utilizzata per il MLP, ovvero la LeakyReLu. Anche il blocco fully connected posto alla fine dei layer convolutivi cambia in funzione del numero di parametri risultanti dalla convoluzione.

| Configurazione Convolutional Neural Network | | | |
|---|---|---|---|
| | A | B | C |
| **Numero di layer** | 3+2 | 3+2 | 3+2 |
| Conv-1 | $kernel\_size$ = 1<br>$stride$ = 1<br>$C_{out}$ = 16 | $kernel\_size$ = 1<br>$stride$ = 1<br>$C_{out}$ = 16 | $kernel\_size$ = 1<br>$stride$ = 1<br>$C_{out}$ = 16 |
| Conv-2 | $kernel\_size$ = 3<br>$stride$ = 1<br>$C_{out}$ = 32 | $kernel\_size$ = 3<br>$stride$ = 1<br>$C_{out}$ = 32 | $kernel\_size$ = 3<br>$stride$ = 1<br>$C_{out}$ = 32 |
| Conv-3 | $kernel\_size$ = 3<br>$stride$ = 3<br>$C_{out}$ = 32 | $kernel\_size$ = 3<br>$stride$ = 3<br>$C_{out}$ = 64 | $kernel\_size$ = 3<br>$stride$ = 3<br>$C_{out}$ = 64 |
| Flatten | | | |
| FC | $in$ = 329<br>$out$ = 32 | $in$ = 649<br>$out$ =32 | $in$ = 649<br>$out$ = 64 |
| FC | $in$ = 32<br>$out$ = 2 | $in$ = 32<br>$out$ = 2 | $in$ = 64<br>$out$ =2 |
| **Numero di parametri allenabili** | 14 114 | 26 434 | 44 962 |

Tabella 3.4: Configurazioni CNN

Tutte le configurazioni hanno solamente 3 layer convolutivi, apparentemente pochi rispetto alle CNN più famose con decine di layer. Essendo l'input una serie temporale esso è decisamente meno complesso rispetto ad un'immagine e pertanto è possibile estrarre un minore contenuto informativo ed usare molti neuroni porterebbe sicuramente all'overfitting. I primi due layer convolutivi sono uguali in tutte le configurazioni, facendo variare solamente l'ultimo nel numero di features in output ma non variando le dimensioni di larghezza e lunghezza della matrice risultante. Tra i layer convolutivi e la classificazione è necessario "srotolare" la matrice di output dalla convoluzione in quanto i layer lineari possono lavorare solo con vettori monodimensionali. In questo scenario l'ultimo layer convolutivo comporta una variazione della lunghezza del vettore, come osservabile nella tabella 3.2.

Come per i MLP anche le CNN richiedono di essere regolarizzate in maniera opportuna, seppur generalmente presentino meno parametri siccome un layer convolutivo è più leggero rispetto ad uno lineare. Infatti, nelle tre configurazioni presentate la maggior parte dei parametri allenabili risiede nel blocco responsabile alla classificazione. Su questo tipo di reti non è possibile applicare il dropout come tecnica di

regolarizzazione siccome non abbiamo più neuroni ma in cui il kernel rappresenta il neurone stesso ed escluderlo con una probabilità *p* significherebbe escludere tutto il layer e compromettere il corretto funzionamento della rete. Ho scelto quindi di utilizzare una funzione di regolarizzazione per evitare l'overfitting. Come descritto nella sezione 2.2.2 le due funzioni più famose sono la L1 e L2, ma nulla vieta di crearne altre per ottenere una loss aggiuntiva personalizzata in base alla rete presa in esame. La prima soluzione presa in esame è proprio quest'ultima: creare una funzione di regolarizzazione che tenga conto dei parametri dei soli layer convolutivi pensando che essi siano gli effettivi responsabili dell'apprendimento. Questo approccio dal punto di vista implementativo non risulta troppo complesso in quanto è possibile calcolare i pesi di ogni layer durante il training e tenere traccia dell'aggiornamento di essi. Purtroppo, tutti i layer sono responsabili dell'apprendimento, anche i layer fully connected, considerando anche che nelle configurazioni prese in esame la maggior parte dei parametri allenabili provengono proprio dal blocco responsabile della classificazione e pertanto questo approccio si è rivelato non applicabile. Successivamente ho testato la funzione L1, semplice in quanto come funzione di regolarizzazione effettua la semplice somma di tutti i parametri della rete, somma che sarà poi pesata con un parametro λ. Come loss non è invasiva, infatti durante la backpropagation ogni parametro ha lo stesso peso nel computo della loss globale. Dal punto di vista della somma dei pesi è stato possibile osservare che essa calava gradualmente all'aumentare delle epoche di training e ciò è corretto e indica che la rete nell'aggiornamento dei pesi cerca effettivamente di limitarne i valori e di conseguenza la capacità computazionale per rispettare i vincoli imposti dalla minimizzazione della loss globale. Per completezza ho realizzato anche la regolarizzazione L2, nota anche come *weight decay* e già implementata in molti framework di Deep Learning. Essa consiste nella somma del quadrato di ogni parametro e a differenza della L1, durante la backpropagation ogni parametro β impatta con un fattore $2 \cdot \beta$ al computo della loss globale. Recentemente si è iniziato a combinare queste due funzioni ed applicarle a diversi layer della rete: nei layer convolutivi L1 è la funzione predominante e talvolta è accompagnata anche dalla L2, mentre quest'ultima trova molto successo nei layer finali di classificazione. Tra le tre soluzioni proposte che usano queste due funzioni ho scelto di utilizzare solamente la regolarizzazione L1 ed applicarla a tutta la rete. Potrebbe non essere la scelta migliore dal punto di vista delle performances ma la differenza tra di esse non sono così significative da impattare sui risultati finali dell'ambito oggetto di questa tesi.

Un ultimo aspetto da considerare ma di vitale importanza, comune ad entrambe le reti, MLP e CNN, è la scelta dell'algoritmo di ottimizzazione da usare per l'aggiornamento dei parametri della rete. Il più comune è la discesa del gradiente che aggiorna i parametri basandosi su tutti i dati del dataset. Come strategia è la migliore perché considera tutti i dati a disposizione ma è anche costosa se la dimensione del dataset è molto importante per il calcolo del gradiente. Di conseguenza è stata sviluppata una variante che oggi è lo standard *de facto* per gli algoritmi di ottimizzazione: lo *Stochastic Gradient Descent*. Esso consiste in aggiornamenti frequenti del gradiente dopo ogni elemento visto e ciò consente una minore memoria richiesta e la possibilità di trovare nuovi minimi globali vista la diversa direzione del gradiente tra i singoli esempi e il dataset completo. Rispetto al GD classico introduce una varianza maggiore e per arrivare alla stessa convergenza richiede un learning rate inferiore. Per ridurre la varianza introdotta dal SGD è stato proposto il Momentum [32]. Esso introduce un termine dipendente dalle iterazioni

precedenti, moltiplicato per un iperparametro α in grado di rappresentare l'importanza dell'iterazione passata. L'aggiornamento dei parametri usa una combinazione lineare del gradiente nell'iterazione corrente e dell'aggiornamento nell'iterazione precedente:

$$m_{t+1} = \alpha \cdot m_t + \nabla f(x_t)$$
$$w_{t+1} = w_t - \eta \cdot m_{t+1}$$

dove $m_{t+1}$ rappresenta la media dei gradienti calcolata come la somma pesata del gradiente precedente e quello corrente, in modo tale che scegliendo $\alpha \in [0,1)$ i gradienti nei primi step assumano importanza sempre più inferiore. Quest'operazione permette di ridurre la varianza nell'aggiornamento dei pesi a discapito di un ulteriore iperparametro. Infine, tra gli algoritmi presi in considerazione troviamo Adam [33], un metodo in grado di calcolare un learning rate adattivo per ogni iterazione. Concettualmente è molto simile al Momentum ma a cui viene aggiunto un ulteriore termine per regolarizzare il gradiente attraverso il momento secondo (varianza) e l'iperparametro $\beta \in [0,1)$:

$$m_{t+1} = \alpha \cdot m_t + (1-\alpha) \cdot \nabla f(x_t)$$
$$v_{t+1} = \beta \cdot v_t + (1-\beta) \cdot \nabla f^2(x_t)$$
$$\eta_{t+1} = \eta \cdot \frac{\sqrt{1-\beta^{t+1}}}{1-\alpha^{t+1}}$$
$$w_{t+1} = w_t - \eta_{t+1} \cdot \frac{m_{t+1}}{\sqrt{v_{t+1}} + \varepsilon}$$

L'aggiornamento del learning rate $\eta$ è molto utile soprattutto nei primi step di aggiornamento, mentre il parametro $\varepsilon > 0$ è usato solamente per garantire un denominatore diverso da zero nel caso in cui la regolarizzazione del momento secondo sia uguale a zero. Adam ha riscontrato un notevole successo grazie alle ottime prestazione offerte nelle prime iterazioni, garantendo una discesa più rapida grazie proprio al learning rate adattivo, oltre che la possibilità di lavorare con learning rate meno bassi rispetto a SGD e Momentum. D'altro canto, però, Adam è l'algoritmo più costoso in termini di memoria e computazione e inoltre permette di ottenere generalmente minimi qualitativamente peggiori rispetto agli altri algoritmi. Pertanto, la scelta finale è ricaduta sullo SGD con Momentum, ottenendo minimi globali migliori a fronte di una potenza computazionale e memoria richiesta non eccessiva.

# Capitolo 4

# Metodi di Continual Learning

Nella sezione 2.3 sono stati illustrati gli aspetti generali del Continual Learning e introdotte le principali strategie. In questo capitolo verranno approfonditi nello specifico tutti gli algoritmi, usati e scartati, e nel caso di quest'ultimi le motivazioni che hanno portato a questa scelta ed eventuali stratagemmi per ottenere metodi simili. Le metodologie proposte rappresentano, al momento della scrittura di questo elaborato, lo stato dell'arte per quanto riguarda gli algoritmi di Continual Learning. Ciò nonostante, non tutti si applicano al problema della classificazione di serie storiche finanziarie oppure non offrono le performance desiderate. Questo non è un fattore negativo ma al contrario permetterà di identificare singoli algoritmi o strategie che mal si adattano a questo tipo di problema. Infatti, i metodi non saranno valutati solamente attraverso le performance e le metriche di valutazione del CL ma si terrà conto anche di come essi si comportano dinanzi a tre fattori: 1) memoria, 2) scalabilità (intesa come aumento del numero di task) e 3) potenza computazionale.

I metodi verranno, infine, presentati in ordine cronologico per poter evidenziare come alcuni metodi siano ricavati dai precedenti e sfruttino loro punti di forza oppure siano realizzati per risolvere eventuali limitazioni.

## 4.1 Gradient Episodic Memory

I primi problemi di learning di task multipli sequenziali si basavano sull' *Empirical Risk Minimization* (ERM), principio con il quale veniva posto il vincolo della limitata conoscenza che un modello di Deep Learning poteva possedere, soprattutto se la distribuzione dei dati in input cambiava nel corso del tempo, ovvero si osservava un cambio di task. Questo vincolo rappresenta l'aspetto fondamentale del *catastrophic forgetting*.

Nel 2107, Lopez e Ranzato proposero un modello di learning slegato dalla distribuzione dei dati e focalizzato sull'osservazione *example by example*. In particolare, veniva abbandonata la classica coppia esempio-label $(x_i, y_i)$ a favore di una tripletta

$(x_i, t_i, y_i)$ dove $t_i$ corrisponde ad un *task descriptor*. Il task descriptor nel caso più semplice può essere un intero per identificare il task a cui è associata la tripletta, ma nulla vieta sia un oggetto strutturato per descrivere al meglio il comportamento del task. L'uso di task descriptor può facilitare lo *zero shot learning* di nuovi task come illustrare similarità tra task apparentemente opposti o divergenze tra due task molto simili tra loro. Applicato alle serie temporali, il task descriptor può essere un identificativo del task, come in questa tesi, oppure una struttura che descriva la distribuzione (media e varianza ad esempio) a cui appartengono i dati.

La feature principale di questo metodo è l'*episodic memory* $M_t$ in grado di salvare un piccolo subset del training set per ogni task *t*. Considerati un totale di *T* task ed una memoria complessiva disponibile *M* ogni task avrà una memoria esclusiva pari a $M/T$. Nel caso in cui il numero dei task non sia noto a priori (come in questo caso) è possibile ridurre gradualmente il numero di esempi per ogni task all'aumentare del numero dei task. La loss di ogni memoria e ogni task viene calcolata nel seguente modo:

$$l(f_\theta, M_k) = \frac{1}{|M_k|} \sum_{(x_i, t_i, y_i) \in M_k} l(f_\theta(x_i, k), y_i)$$

La loss complessiva equivale alla media delle loss di tutti gli esempi che compongono la memoria del task. La notazione $f_\theta$ indica lo stato della rete al termine del task $k$ e ciò richiede di salvare lo stato della rete al termine di ogni task. In questo modo si potrà calcolare la loss per ogni memoria e ogni task.

L'obiettivo del GEM è allenare un modello su *T* task sequenzialmente e per evitare che task futuri sovrascrivano i precedenti viene posto un vincolo sulla loss di ogni task: essa non dovrà mai aumentare nei task successivi, siccome un aumento della loss significherebbe una variazione dei parametri della rete che portano ad un peggioramento delle prestazioni sul task. Data una tripletta $(x, t, y)$ il problema di ottimizzazione da risovlere è dunque il seguente:

$$\begin{aligned} &min_\theta \quad l(f_\theta(x, t), y) \\ s.t. \quad &l(f_\theta, M_k) \leq l\left(f_\theta^{t-1}, M_k\right) for\ all\ k < t \end{aligned}$$

Così formulato il problema è complesso da risolvere ma è possibile fare due osservazioni che permetteranno di riscrivere i vincoli. La prima riguarda il salvataggio dei parametri di ogni task: se viene mantenuto il vincolo sul non aumento della loss non è più necessario salvare lo stato della rete ad ogni task. La seconda e più importante, permette di rappresentare la variazione della loss tra un aggiornamento e l'altro tramite l'angolo tra i due vettori gradiente se la funzione è localmente lineare, assunzione valida tra due passi del gradiente ravvicinati. La seconda osservazione ci permette di riscrivere i vincoli di ottimizzazione nel seguente modo:

$$\langle g, g_k \rangle := \langle \frac{\delta l(f_\theta(x, t), y)}{\delta \theta}, \frac{\delta l(f_\theta, M_k)}{\delta \theta} \rangle \geq 0 \quad for\ all\ k < t$$

Questo controllo avviene per ogni esempio (o batch) fornito al modello. Per ogni training step si ha quindi un sistema di $k$ task da risolvere. Come è facile intuire all'aumentare del numero di esempi e di task quest'operazione diventa sempre più onerosa. Se tutte le disequazioni sono rispettate allora non si ha aumento di loss su tutti i task passati e può essere effettuato l'aggiornamento. Se invece, si ha almeno un task dove il prodotto tra gradiente e aggiornamento proposto è negativo allora è necessario effettuare un passo del gradiente in una nuova direzione. Questo rende il problema di ottimizzazione per quello specifico aggiornamento un problema NP-completo, ovvero un problema non risolvibile in tempo polinomiale in alcun modo noto. Pertanto, sono possibili solo approssimazioni del risultato di esso e gli autori dell'algoritmo ne hanno proposta una valida usando il problema duale riducendone la complessità in [16]. L'aspetto rilevante è che risolvere questo problema porta ad un aumento significativo del tempo di esecuzione e della capacità computazionale richiesta. Inoltre, è facile intuire che all'aumentare del numero di task sia sempre più probabile trovare un vincolo non rispettato e incorrere nel problema NP-completo.

A differenza dei metodi di learning tradizionali, aumentando il numero di epoche di training per ogni task aumenta il rischio di incorrere nel problema NP-C e quindi di ottenere performance peggiori. È consigliato dunque allenare i task per poche epoche, alcune volte anche solo 1 epoca è sufficiente. Anche la dimensione della memoria riservata ad ogni task può giocare un ruolo significativo: infatti aumentando le dimensioni della memoria è possibile ottenere performances migliori. Questo aspetto però è delicato se il problema riguarda serie temporali. Infatti, una memoria troppo grande consentirebbe di salvare troppi dati ed incorrere in uno pseudo learning parallelo di più task e ciò è da evitare in un contesto dove l'ordine temporale è fondamentale. Infine, è possibile agire sulla metodologia di riempimento della memoria: tipicamente si inseriscono i primi o ultimi $n$ esempi del dataset ma nulla vieta di riempirlo in maniera più efficiente, sempre rispettando la contiguità temporale nel caso delle serie storiche finanziarie.

GEM è un valido algoritmo di CL ma possiede alcune aree di miglioramento: 1) esso si concentra principalmente sull'ottenere un *backward transfer* positivo (o se negativo il più alto possibile) tra i task non esplorando la possibilità di *transfer learning* tra i task; 2) gli autori hanno proposto metodi basici sul riempimento della memoria e della sua gestione e infine 3) il calcolo dei gradienti per ogni esempio porta necessariamente ad un tempo di esecuzione lungo e superiore ad altre metodologie.

Dal punto di vista della memoria richiesta questo metodo è particolarmente oneroso in quanto devono essere salvati non solo gli esempi dei task passati ma anche i gradienti dopo ogni aggiornamento. La possibilità di incorrere nel problema NP-C impatta notevolmente la scalabilità ma soprattutto la potenza computazionale richiesta, rendendo questo metodo interessante ma di non così semplice utilizzo nel caso in cui l'infrastruttura hardware non sia adeguatamente implementata. In ambito produttivo questo potrebbe rappresentare un problema di non poco conto effettuando learning su molteplici serie finanziare e quindi ripetere l'analisi un numero di volte pari alle serie usate.

## 4.2 Averaged Gradient Episodic Memory

Algoritmo nato nel 2018 ad opera dei Facebook Research Labs con la collaborazione di Ranzato, già autore del GEM. GEM aveva posto le basi del CL seppur in modo non pienamente efficiente e l'*Averaged Gradient Episodic Memory* (A-GEM) è stato presentato come ottimizzazione del predecessore. A-GEM si concentra in particolare sulle lacune del GEM, ovvero tempo e complessità computazionale, tenendo in considerazione anche vincoli di memoria. La prima area d'azione è stata il numero di epoche di training; se GEM ammetteva più di una epoca con risultati soddisfacenti, A-GEM è stato pensato e realizzato per ottenere le migliori performance osservando ogni esempio solo una volta e questo ovviamente permette di ridurre il tempo computazionale ma non solo. La modifica sostanziale risiede nella loss function; essa è stata parzialmente ripensata per scongiurare il problema NP-C e ridurre tempo e complessità, pur mantenendo performance simili.

Il problema di minimizzazione da affrontare rimane invariato, ovvero ridurre la loss su tutti i task, ma sono stati riscritti i vincoli da rispettare, vero tallone d'Achille di GEM:

$$min_\theta \quad l(f_\theta(x,t),y)$$
$$s.t. \quad l(f_\theta, M_k) \leq l\left(f_\theta^{t-1}, M_k\right) for\ all\ k < t$$

Questi vincoli con il crescere del numero di task possono facilmente diventare proibitivi. Per alleviare il peso della computazione si è optato per un rilassamento dei vincoli, passando dal non aumento della loss su ogni task passato dopo ogni training step al non aumento della loss media su tutta la memoria episodica dei task passati. Formalmente:

$$min_\theta \quad l(f_\theta, D_t)$$
$$s.t. \quad l(f_\theta, M) \leq l\left(f_\theta^{t-1}, M\right) \quad where\ M = \cup_{k<t} M_k$$

Il problema posto in questi vincoli rimane di non facile risoluzione, ma rimane valida l'osservazione fatta in precedenza sui vettori dei gradienti. Il problema diventa quindi del tipo:

$$min_{\tilde{g}} \ \frac{1}{2}\|g - \tilde{g}\|_2^2$$
$$s.t. \ \widetilde{g^T} \cdot g_{ref} \geq 0$$

dove $g_{ref}$ è il gradiente calcolato da una batch randomica campionata dalla memoria episodica di tutti i task passati $\left(x_{ref}, y_{ref}\right) \sim M$. In altre parole, A-GEM sostituisce *t-1*

vincoli di disuguaglianza con uno solo. Rimane però possibile che il vincolo non venga rispettato, ma in questo caso non si ha nessun problema complesso e sarà possibile risolvere il vincolo mediante il metodo del gradiente proiettato ottenendo:

$$\tilde{g} = g - \frac{g^T \cdot g_{ref}}{g_{ref}^T \cdot g_{ref}} \cdot g_{ref}$$

Questa operazione permette di ottenere un'efficienza maggiore dal punto di vista della memoria, siccome non dovrà essere salvata una matrice G di gradienti. Inoltre, il tempo di computazione di A-GEM è di ordini di magnitudine inferiore rispetto al predecessore grazie a tre considerazioni: 1) non viene richiesto il calcolo della matrice dei gradienti ma solamente di un gradiente randomico ottenuto da un subset della memoria, 2) nel caso di violazione dei vincoli si passa da un problema NP-C ad un prodotto interno e 3) anche con un grande numero di task si riducono significativamente le violazioni dei vincoli. Tutte queste considerazioni rendono A-GEM più veloce senza inficiare sulle performance di una singola epoca di training.

Come per il GEM, anche questo metodo ammette diversi algoritmi per il riempimento della memoria episodica. I più famosi riguardano i primi o ultimi *n* esempi di ogni task, in modo tale che ogni task sia equamente presente nella memoria. Siccome però non viene calcolata la loss su ogni task ma una loss media campionando una batch casuale, la composizione della batch potrebbe non rispettare la distribuzione originale tra i task. Questo rappresenta un problema significativo in quanto si pongono dei vincoli che non rispecchiano a pieno la situazione reale. Per questo motivo in questa tesi è stato testato A-GEM sia nella configurazione classica proposta dagli autori, sia di una versione alternativa che sfrutta l'algoritmo di *reservoir sampling* per garantire che la distribuzione della memoria sia il più fedele possibile all'originale dei task. L'algoritmo è estremamente semplice e non aggiunge alcuna complessità o tempo extra di computazione, ma che cambia in maniera sostanziale il problema, come sarà possibile osservare nel capitolo 5.

```
1: Input the reservoir size k
2: i = 0
3: for arriving stream item S_i do
4:     i = i + 1
5:     if i ≤ k then
6:         Add item S_i into the reservoir
7:     else
8:         Generate a random number j from 0 to i
9:         if j < k then
10:            Replace the j-th item with S_i
11:        end if
12:    end if
13: end for
```

Figura 4.1: Algoritmo di reservoir sampling

Siccome A-GEM tratta e risolve le maggiori problematiche del predecessore memoria e scalabilità non costituiscono più una limitazione alle applicazioni dell'algoritmo. Anche il vincolo della potenza computazionale si è notevolmente rilassato dovendo risolvere solo un problema di proiezione del gradiente. Questo pone A-GEM come uno degli algoritmi trattati più flessibile ed applicabile anche a molte serie finanziare.

## 4.3 Synaptic Intelligence

Un grande limite nello sviluppo di reti neurali in grado di imparare task multipli sequenziali risiede nella struttura monodimensionale del neurone, portando un modello al catastrophic forgetting. Per permettere ai neuroni di consolidare la loro conoscenza su un task, passato o futuro, è necessario definire uno stato di essi relativo all'importanza del neurone stesso sul task appreso. Questo stato può variare ad ogni training step e necessita di essere continuamente aggiornato. Il miglior modo per valutare quanto un neurone sia significativo per un task è calcolare il suo contributo alla loss globale del task corrente. Così facendo, al termine di ogni task sarà possibile determinare quali neuroni contribuiscano maggiormente all'apprendimento del task e quindi prevenire il loro aggiornamento nei task futuri, scongiurando così il forgetting e mantenendo conoscenza del passato.

Questo è un metodo di regolarizzazione, dove non si ricorre all'uso di memorie esterne o alla variazione dell'architettura, ma si agisce solamente sui neuroni definendo un'ulteriore loss legata allo stato dei neuroni stessi. Questo porterà il modello a non modificare i parametri di essi per scongiurare un aumento della loss globale, focalizzandosi, esclusivamente o quasi, sui neuroni rimanenti del modello.

Per realizzarlo, Zenke et al, hanno sviluppato una serie di algoritmi in grado di calcolare l'importanza $\omega_k^{\mu}$ di ogni neurone relativo al task $\mu$ e al neurone con parametri $\theta_k$. Il training della rete neurale è caratterizzato da una traiettoria $\theta(t)$ nello spazio dei parametri. Per ogni task questa traiettoria di learning arriverà il più vicino possibile al punto di minimo della loss funztion sul task corrente. Consideriamo ora un aggiornamento dei parametri $\delta(t)$ al tempo $t$ che porta ad una variazione della loss sul task corrente. La variazione della loss può essere approssimata dal gradiente $g$ e in tal caso la relazione

$$L\big(\theta(t) + \delta(t)\big) - L\big(\theta(t)\big) \approx \sum_k g_k(t) \cdot \delta_k(t)$$

è da considerarsi valida. La variazione $\delta_k(t) = \theta_k'(t)$ contribuisce quindi alla variazione della loss globale. Se volessimo calcolare la variazione su tutta la traiettoria dovremmo sommare tutti questi piccoli aggiornamenti, equivalenti all'integrale del vettore gradiente da $t$=0 fino al termine $T$ del training del task:

$$\int_0^T g\big(\theta(t)\big)\, d\theta = \int_{t_0}^{t_1} g\big(\theta(t)\big) \cdot \theta'(t)\, d\theta = \sum_k \int_{t^{\mu-1}}^{t^{\mu}} g_k(\theta(t)) \cdot \theta_k'(t) dt \equiv -\sum_k \omega_k^{\mu}$$

Siccome il gradiente è un campo conservativo il valore dell'integrale equivale alla differenza della loss tra gli istanti $t_1$ e $t_0$. Inoltre, l'integrale può essere scomposto come somma dei parametri individuali. In questo modo è possibile avere un'interpretazione dell'impatto dell'importanza $\omega_k^{\mu}$ sulla variazione della loss.

Rimane da definire come l'importanza dei parametri può aiutare a risolvere un problema di CL. Essendo task sequenziali, il modello avrà a disposizione solamente la loss $L_{\mu}$ con μ task corrente. Si ha catastrophic forgetting qunado, nel tentativo di minimizzare $L_{\mu}$ inavvertitamente si incrementa significativamente la loss $L_{\upsilon}$ dei task passati υ<μ. In questo contesto l'importanza dei parametri $\theta_k$ è determinata da due fattori: 1) quanto il parametro contribuisce ad un calo della loss e 2) la differenza $\theta_k(t^{\mu}) - \theta_k(t^{\mu-1})$, ovvero la variazione del parametro tra due task consecutivi. Per evitare una variazione importante nei parametri significativi, gli autori hanno proposto una loss modificata

$$\widetilde{L_{\mu}} = L_{\mu} + c \sum_k \Omega_k^{\mu} (\widetilde{\theta_k} - \theta_k)^2 \quad con \quad \widetilde{\theta_k} = \theta_k(t^{\mu-1})$$

con un parametro *c* per definire l'impatto della regolarizzazione e il coefficiente $\Omega_k^{\mu}$ che determina la forza di regolarizzazione di ogni specifico parametro

$$\Omega_k^{\mu} = \sum_{\upsilon<\mu} \frac{\omega_k^{\upsilon}}{(\Delta_k^{\upsilon})^2 + \varepsilon} \quad con \quad \Delta_k^{\upsilon} = \theta_k(t^{\upsilon}) - \theta_k(t^{\upsilon-1})$$

In particolare, se *c*=1 la loss riceverà un egual contributo dalla memoria vecchia e nuova. La soluzione proposta porta non solo a scoraggiare l'aggiornamento di pesi ritenuti importanti, ma se è necessario effettuare questa operazione impone di minimizzare anche lo scostamento dal vecchio valore.

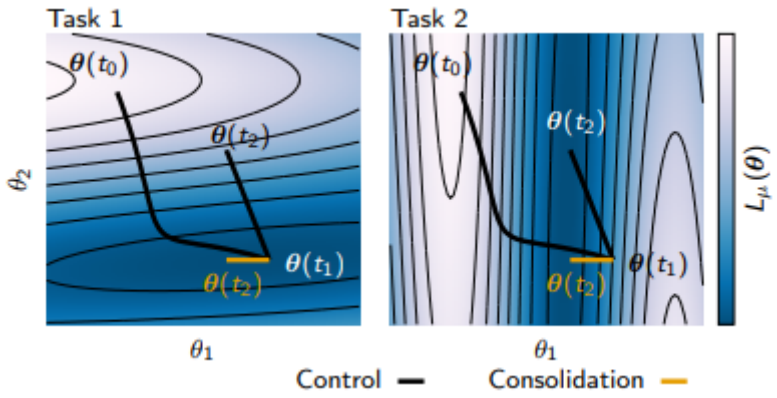



Figura 4.2: Schema aggiornamento parametri

Nella figura 4.2 possiamo vedere come l'aggiornamento tra $\theta(t_1)$ e $\theta(t_2)$ porti ad un aumento della loss sul task 1. In questo caso la regolarizzazione agisce portando ad un consolidamento dei parametri e, dovendo modificare necessariamente il parametro coinvolto, impone che la direzione del gradiente non avvenga in una direzione nello spazio dove la loss sul task 1 sia maggiore.

La SI è un metodo innovativo per quanto riguarda la tecniche di regolarizzazione nel CL, introducendo una regolarizzazione online e a granularità fine, andando ad agire sui singoli neuroni e i loro parametri. Una possibile contestazione a questo algoritmo ma che accomuna tutte le metodologie di regolarizzazione è la scarsa scalabilità. Infatti, con pochi task non si rischia di compromettere la capacità di apprendimento degli ultimi task da apprendere, ma con un numero importante di task non viene posto alcun vincolo sui neuroni minimi disponibili, trovandosi dinanzi a due scenari: 1) un numero insufficiente di neuroni non significativi per i task precedenti e 2) la necessità di modificare neuroni rilevanti per task passati andando a compromettere le loro performance. Questi scenari potrebbero compromettere la capacità di analisi in serie temporali composte da molti timestep e dunque la possibilità che manifestino più andamenti (ovvero i task del problema di CL) al loro interno. Dal punto di vista della memoria non sono presenti vincoli di nessuna natura siccome essa non viene coinvolta, mentre la complessità computazionale non rappresenta un problema siccome tutte le equazioni da risolvere sono piuttosto semplici.

## 4.4 Elastic Weight Consolidation

Un altro popolare algoritmo di regolarizzazione, sviluppato in parallelo alla SI, è l'*Elasitc Weight Consolidation*. Come la SI, esso si basa sulla possibilità di determinare un coefficiente di importanza per ogni neurone da utilizzare poi nel calcolo della loss globale. Questo algoritmo si basa fortemente sul fatto che dato un generico task A esistono multiple configurazioni $\theta_A$ di parametri valide che permettono di ottenere le stesse performance. In questo modo, dopo aver appreso il task A e passando al task successivo B, EWC proteggerà le performance di A vincolando i parametri del modello in

uno spazio avente basso errore per A e al contempo cercando di massimizzare le performance di B.

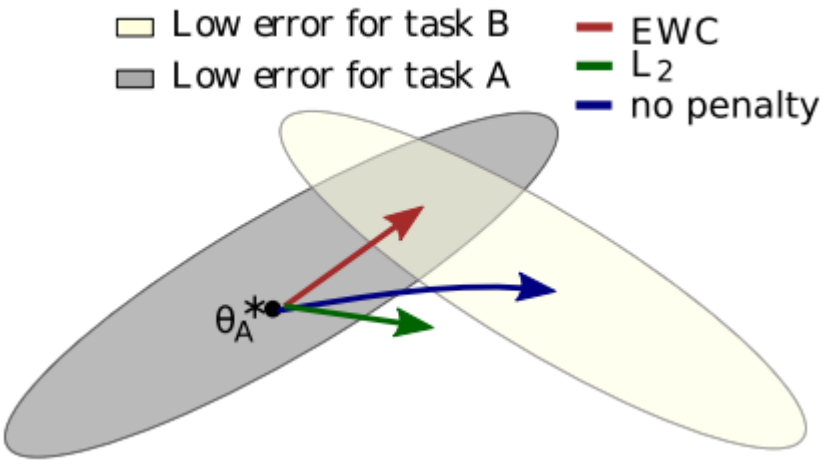



Figura 4.3: Schema funzionamento EWC

Come osservabile dalla figura, EWC non punta ad ottenere l'errore minimo possibile per ogni task ma si focalizza sulla ricerca di un'intersezione tra gli spazi a basso errore dei task coinvolti. Se non venisse applicata alcuna penalty sui parametri sarebbe possibile minimizzare sempre la loss dei task nuovi, sovrascrivendo però le conoscenze acquisite nel passato e sfociando nel catastrophic forgetting.

Per determinare quali parametri sono i più significanti per un task gli autori dell'algoritmo hanno considerato il problema dal punto di vista probabilistico. Sotto questo punto di vista, ottimizzare i parametri di una rete dato il training set equivale alla probabilità $p(\theta|D)$. Questa probabilità può essere fafcilemten ottenuta grazie alla regola di Bayes:

$$\log p(\theta|D) = \log p(D|\theta) + \log p(\theta) - \log p(D)$$

Assumendo ora che il dataset sia diviso in due task indipendenti A ($D_a$) e B ($D_b$) la stessa equazione risulterebbe

$$\log p(\theta|D) = \log p(D_b|\theta) + \log p(\theta|D_a) - \log p(D_b)$$

dove a sinistra dell'equazione abbiamo ancora la probabilità a posteriori, mentre a destra possiamo osservare che le probabilità dipendono unicamente dalla loss del task B ($\log p(D_b|\theta)$). Al contempo tutte le informazioni riguardanti il task A ($\log p(\theta|D_a)$ sono state assorbite dalla probabilità a posteriori. Questo è l'aspetto chiave nell'implementazione di EWC. Se da un lato la vera probabilità a posteriori non è calcolabile, possiamo ottenere una buona approssimazione da una distribuzione gaussiana con media data dai parametri $\theta_A$ e precisione angolare proveniente dalla diagonale della matrice di Fisher *F*. Questa matrice possiede tre caratteristiche chiave: 1) nell'intorno di un minimo equivale alla derivata seconda, 2) calcolabile a partire dalla

derivata prima e comunque facile da ottenere anche per sistemi di grandi dimensioni e 3) è semi definita positiva. La funzione da minimizzare diventa quindi:

$$L(\theta) = L_B(\theta) + \sum_i \frac{\lambda}{2} F_i \, (\theta_i - \theta_{A,i})^2$$

dove l'iperparametro $\lambda$ definisce l'importanza del vecchio task. Incrementando il numero di task, EWC cercherà una soluzione nell'intersezione degli spazi delle soluzioni accettabili per ogni singolo task. Come è facile intuire però aumentando il numero di task aumenta anche la probabilità che lo spazio intersezione sia sempre più piccolo o addirittura nullo. Va da sé che la scalabilità rappresenta un problema in questo tipo di algoritmo, e di conseguenza potrebbe non essere applicabile in serie temporali molto lunghe che esibiscono diversi scenari al loro interno.

Uno dei punti di forza di questo algoritmo è il tempo d'esecuzione; infatti, EWC ha un tempo lineare al numero di parametri del modello e degli esempi che compongono il training set. Infatti, generalmente, i metodi regolarizzazione calcolano le metriche relative all'importanza dei parametri alla fine del training di un task in modo tale da non appesantire eccessivamente la computazione, al contrario di GEM e A-GEM per esempio. Come SI, EWC non possiede vincoli di memoria o di potenza computazionale significativi, rendendolo un valido algoritmo se rispettati i vincoli di scalabilità sopra citati.

## 4.5 Experience Replay

I metodi di CL presentati in precedenza si focalizzano sul consolidamento delle conoscenze pregresse tramite il congelamento di parametri rilevanti oppure il controllo che l'apprendimento di task futuri non infici sui precedenti. Utilizzare queste metodologie è possibile ma esse presentano diversi limiti tra scalabilità, potenza e tempo di computazione. Il consolidamento di conoscenze nel cervello umano può avvenire, però, in diversi metodi. Uno di questi consiste nella periodica osservazione per consolidare conoscenze precedentemente acquisite ma potenzialmente sovrascritte nel tempo. Questo è l'approccio proposto nel 2019 dal gruppo di DeepMind di Google [12]. L'utilizzo di esempi di task precedenti è un metodo relativamente semplice ma molto potente, in grado di risolvere diverse problematiche dei predecessori. In particolare, l'*Experience Replay* si dimostra efficace quando la memoria destinata ad esperienze passate è limitata da vincoli, e questo permette di ottenere diversi vantaggi:

- ER offre la soluzione migliore al problema della *stability-plasticity dilemma*.
- Semplicità. Il metodo proposto è di gran lunga il meno complesso e ciò permette di combinarlo con altri per ottenere migliori performance.

- ER non necessita di *boundaries* che delimitino il cambio di task, a differenza dei metodi precedentemente proposti (GEM su tutti con il task descriptor)

ER nella sua formulazione originale aveva come oggetto l'allenamento di agenti in un contesto di Reinforcement Learning, ma i metodi proposti possono essere facilmente trasporti in un framework di Deep Learning. Come detto in precedenza, ER usa una memoria (o *buffer*) per salvare coppie esempio-label *(x,y)* appartenenti a task passati, senza alcun task descriptor o riferimento al task di appartenenza, in linea con l'assenza di boundaries precedentemente illustrata. Durante ogni training step, oltre che la batch di esempi del task corrente viene campionata una batch composta da esempi dei task precedenti salvati in memoria e verrà calcolata la loss su entrambe le batch:

$$L_{task} = \mathrm{E}_{(x,y)\sim D_t}[l(y, f_\theta(x)] \quad e \quad L_{memory} = \mathrm{E}_{(x,y)\sim M}[l(y, f_\theta(x)]$$

La loss complessiva risulterà dunque:

$$L_{global} = L_{task} + L_{memory}$$

Le due componenti della loss globale in ambito RL sono note anche come *on-policy* e *off-policy loss*. Un aspetto interessante è dato dal fatto che entrambe le loss usano i parametri della configurazione del task corrente. Così facendo il modello è libero di modificare i propri parametri, basandosi non solo sul task corrente ma anche su una piccola porzione dei task precedenti.

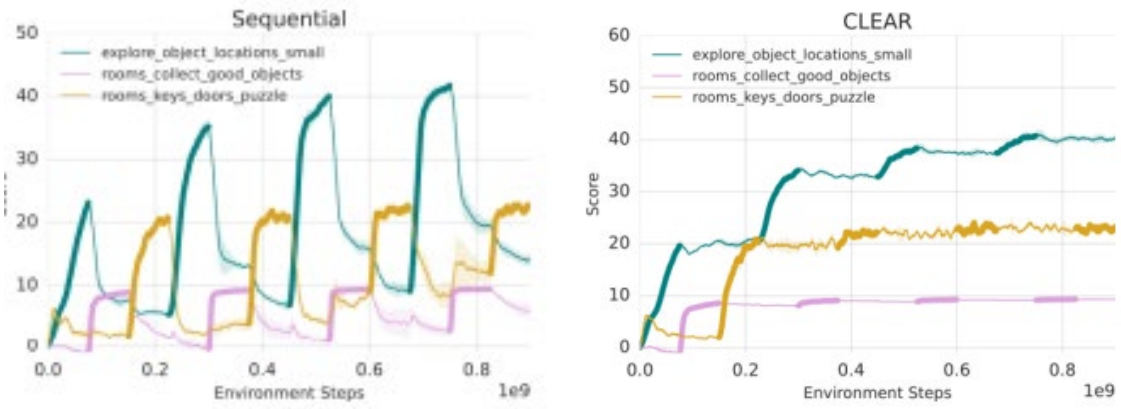


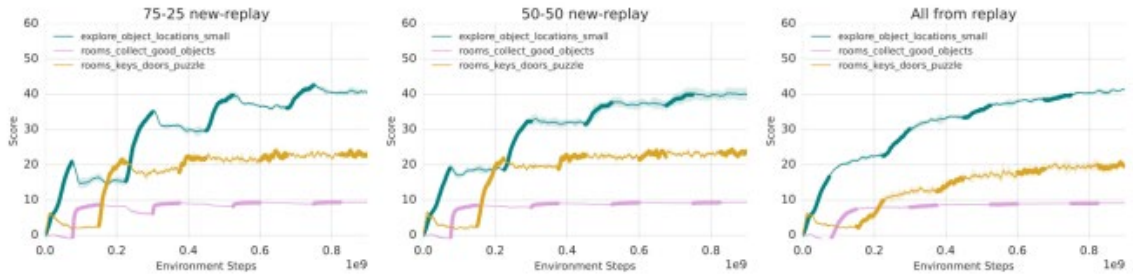

Figura 4.4: Funzionamento ER

Nella figura 4.4 possiamo osservare come si comporta ER in confronto con il training sequenziale e ciclico di tre task. Nel training sequenziale i task, quando non è il loro turno di training, manifestano non solo forgetting ma è possibile osservare il diverso comportamento di un generico task rispetto agli altri. Il training ciclico di tutti i task rappresenta una strategia efficace per l'ER ma purtroppo non è applicabile a problemi

che trattano serie temporali in quanto il modello osserverebbe dati non nel corretto ordine temporale.

Utilizzando un buffer di memoria è necessario definire la strategia di riempimento di esso. Se in GEM e nelle sue varianti i dati del buffer venivano usati solo per effettuare un controllo sui vincoli, nell'ER essi ricoprono un ruolo attivo. Di conseguenza, la composizione del buffer deve necessariamente essere fedele alla distribuzione di dati tra i task. Il reservoir sampling, già illustrato nei metodi precedenti, garantisce esattamente ciò ed è pertanto l'algoritmo scelto per il riempimento del buffer. L'aggiornamento del buffer avviene al termine di ogni step della prima epoca di training, vale a dire che se un dato viene osservato per la prima volta può essere inserito nel buffer ed essere campionato per comporre la batch di memoria anche durante il training del task stesso nelle epoche successive. Il fatto che il potenziale inserimento avvenga solo durante la prima epoca impedisce di avere delle copie all'interno del buffer di memoria.

Importante è anche il bilanciamento in ogni training step tra dati appratenti al task corrente e dati campionati dal buffer. In precedenza, è stato utilizzato il termine batch per indicare i dati ma la batch può assumere dimensioni diverse. Non esiste una corretta proporzione tra i dati ma valori diversi portano a comportamenti diversi, che verranno illustrati con l'ausilio della figura 4.5:




(a) (b) (c)

Figura 4.5: Configurazioni bilanciamento dati

a) Una configurazione 75-25 tra dati nuovi e replay consente di ottenere il miglior aumento di performance durante il training del task corrente ma al contempo si osserva un drop importante negli altri task.
b) La configurazione 50-50 è la più bilanciata tra quelle proposte con un buon *trade-off* tra il prevenire il forgetting e imparare il task corrente.
c) Configurazione più estrema in cui la batch è composta solamente da dati campionati dal buffer. Sembrerebbe che in questo modo il task corrente non venga allenato ma non è così: in precedenza è stato descritto il momento di riempimento e aggiornamento del buffer in cui anche dati del task corrente possono comporre la batch di memoria e di conseguenza si realizza il training anche sul task corrente.

La configurazione 50-50 essendo la più bilanciata permette di evitare forgetting e al contempo garantisce un adeguato apprendimento dei task ed è stata scelta come configurazione finale per questo metodo.

Rimane infine da trattare la dimensione del buffer di memoria. Non ci sono vincoli particolari dettati dall'algoritmo ma dalle performance e dai dati. La dimensione minima

del buffer deve essere tale da garantire almeno un esempio per ogni task vale a dire minimo $T$ slot. Sulla dimensione massima invece è necessario considerare che il buffer di memoria deve essere rappresentativo della distribuzione dei task ma non essere uguale al dataset stesso; pertanto, un buffer di dimensione tale da permettere di salvare al suo interno tutti i dataset di tutti i task è ideologicamente errato, oltre che non applicabile al problema trattato nella tesi, dove l'ordine temporale è significativo e deve essere mantenuto. Inoltre, potendo campionare dai dataset completi non si tratterebbe di Continual Learning ma bensì Multi-task learning. Infine, con buffer di dimensioni considerevoli si ottiene un consolidamento leggermente migliore delle conoscenze dei task precedenti ma a fronte di un considerevole uso di memoria extra.

La semplicità e le buone performance garantite dal ER lo hanno portato rapidamente ad essere uno dei metodi più utilizzati e più studiati, portando allo sviluppo di moltissime varianti o *tricks* per ottenere un aumento di performance o miglior resistenza al forgetting, come in [34].

Infine, è possibile effettuare l'analisi dei vincoli citati all'inizio del capitolo. La memoria è sicuramente il vincolo più delicato, siccome con dataset di grandi dimensioni, anche con buffer pari al 20-25% della dimensione del dataset originale si ottengono occupazioni di memoria importanti. La scalabilità invece non presenta alcun limite, se non legata alla memoria a disposizione, mentre l'algoritmo ha una scalabilità pressoché infinita. Per concludere la potenza e il tempo di computazione non rappresentano problemi di alcun genere siccome sono coinvolte solo operazioni di base per un modello di deep learning.

## 4.6 Dark Experience Replay

Sviluppato in [13] come perfezionamento del ER mantenendo una formulazione molto semplice. Il nome *Dark Experience Replay* deriva dal termine *dark knowledge*, presentato in [36] per distillare le esperienze passate.

Nel CL una funzione $f$ con parametri $\theta$ viene ottimizzata su un task alla volta in maniera sequenziale. Possiamo indicare con $h_\theta(x)$ i *logits*, ovvero i valori di output non scalati, e con $f_\theta(x) \triangleq softmax(h_\theta(x))$ i valori post funzione di attivazione rappresentanti la distribuxzione di probabilità rispetto le possibili classi. La funzione obiettivo risulta:

$$argmin_\theta \sum_{t=1}^{T} L_t \;\; con \;\; L_t \triangleq E_{(x,y)\sim D_t} \left[ l\big(y, f_\theta(x)\big) \right]$$

Questa appare subito complessa siccome i dati dei task passati risultano non più disponibili. Idealmente, questo algoritmo nell'allenare i parametri sul task corrente, ricerca quelli che consentono le performance migliori senza dimenticare i task precedenti:

$$L_t + \alpha \sum_{k=1}^{t-1} E_{x\sim D_t} \left[ D_{KL}(f_{\theta_k}(x) \,||\, f_\theta(x)) \right]$$

dove $\theta_k$ sono i parametri migliori al termine del task $k$ e α è un iperparametro per regolare l'impatto del termine. Ma anche in questo caso è richiesto ogni dataset di ogni task. Per ovviare al problema viene adottata la soluzione che caratterizza l'ER, ovvero un buffer di memoria $M_t$ capace di salvare le esperienze passate del task $t$. A differenza di ER però, il modello viene riallenato non più con la coppia esempio-label *(x,y)* ma con esempio-logit *(x,z)* con $z \triangleq h_{\theta_t}(x)$ ovvero il logit ottenuto durante il training del task a cui il dato appartiene. La strategia scelta per riempire il buffer è ancora una volta il reservoir sampling in modo tale da mantenere la distribuzione tra i task sia nel buffer che nella batch ottenuta dal campionamento di esso. L'equazione precedente riscritta risulta:

$$L_t + \alpha \cdot E_{(x,z)\sim M}\left[D_{KL}(softmax(z) \,||\, f_\theta(x))\right]$$

Sotto determinate assunzioni l'ottimizzazione della divergenza di Kullback-Liebler equivale alla minimizzazione della distanza euclidea tra gli output pre softmax (ovvero i logits). Comparare i logits al posto delle label permette di evitare la perdita di informazioni nello spazio delle probabilità dovuta alla compressione della funzione di attivazione finale, in questo caso il softmax. Possiamo dunque scrivere la funzione obiettivo finale da ottimizzare con DER:

$$L_t + \alpha \cdot E_{(x,z)\sim M}\left[\|z - h_\theta(x)\|_2^2\right]$$

L'uso dei logits permette di ottenere modelli più calibrati, intesi come misura della confidenza della precisione. Infatti, tipicamente le reti neurali che usano le *true label* si trovano in una pozione di *over confidence* sulle predizioni effettuate. Se con output discreti questo problema è rilevante solo con valori a metà tra due output, nell'ambito dei problemi con output appartenenti allo spettro continuo questo può portare a predizioni errate. Inoltre, i logits non scalati permettono un aggiornamento dei parametri con una granularità più fine.

DER possiede le medesime caratteristiche del ER rispetto a memoria e scalabilità. Anche la complessità computazionale non varia rispetto alla metodologia da cui è tratto, mentre varia il tempo di training. Esso è leggermente maggiore per ogni singolo task, vuol dire che con un gran numero di task la differenza di tempo di training tra i due metodi può risultare importante.

## 4.7 Altre metodologie

In questa sezione verranno presentate tutte le tecniche di CL studiate in una prima fase e che dal punto di vista teorico ben si adattano anche con serie temporali. Siccome però questa tesi tratta il CL dal punto di vista del *Domain-IL*, ovvero un problema dove

non è disponibile un separatore o identificatore tra i task è stato necessario scartare le principali tecniche definite architetturali o di *parameter isolation*.

### 4.7.1 Progressive Neural Networks

I primissimi algoritmi di CL per garantire il learning di una serie di task consecutivi facevano pesantemente utilizzo di tecniche di transfer learning, non focalizzandosi sull'impedire ai nuovi task di sovrascrivere i vecchi ma piuttosto di come questi potessero trarre profitto da conoscenze già acquisite. Le *Progressive Neural Networks* integrano questo desiderio direttamente dentro l'architettura della rete neurale: il forgetting viene scongiurando istanziando una nuova rete neurale (o *column*) per ogni task da risolvere e abilitare il transfer learning tra task tramite delle connessioni laterali.

Una PNN viene inizializzata con una singola colonna, ovvero una rete neurale profonda $L$ layer, funzioni di attivazione $h_i \in \mathcal{R}$ con $i \leq L$ e parametri $\theta$ allenati fino a convergenza. Quando si presenta un nuovo task, i parametri del primo task vengono "congelati" e una nuova colonna con parametri $\theta_2$ viene istanziata, del tutto uguale alla precedente. L'inizializzazione dei parametri della colonna può essere randomica oppure essere uguale per tutte le colonne. Il transfer learning viene reso possibile grazie a connessioni laterali dove il layer $h_i^{(2)}$ riceve input sia da $h_{i-1}^{(2)}$ che da $h_{i-1}^{(1)}$. Generalizzando per $K$ task si ottiene:

$$h_i^{(k)} = f(W_i^{(k)} \cdot h_{i-1}^{(k)} + \sum_{j<k} U_i^{(k:j)} \cdot h_{i-1}^{(j)})$$

dove $W_i^{(k)}$ rppresenta la matrice dei pesi del layer *i* per la colonna *k* e $U_i^{(k:j)}$ sono le connessioni laterali dal layer *i-1* della colonna *j* verso il layer *i* della colonna *k*. La funzione *f* introduce una non linearità: tipicamente viene usata $f(x) = max(0,x)$ per tutti i layer intermedi.

Un'architettura di questo tipo permette di risolvere $K$ task in maniera indipendente, accelerare il learning dove possibile grazie al transfer learning ed infine evitare il catastrophic forgetting. Il transfer learning permette di superare il classico paradigma del fine tuning; infatti, esso sarà utile soltanto se i task esprimono interferenza costruttiva tra loro, e questa è una considerazione impossibile da fare a priori.

In figura 4.6 è descritta una PNN con K=3. Dall'immagine è possibile notare la presenza di un blocco al termine di ogni connessione laterale. Esso è denominato *adapter*, e la sua presenza è necessaria per adattare gli output dei layer dei task passati per i layer del task corrente. L'adapter di norma performa una *dimensionality reduction* se i dati in input tra i task differiscono di dimensionalità e se necessario applicano funzioni di scaling e attivazione. In altre parole, performano tutta una serie di operazioni necessarie per garantire il corretto transfer.

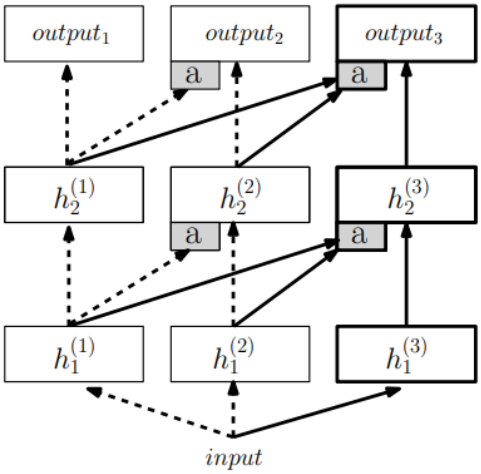



Figura 4.6: Descrizione di una PNN

Questo metodo presenta ovviamente delle limitazioni: la prima è dovuta al numero di reti da istanziare, pari al numero di task da risolvere, che necessariamente porta ad avere una grandissima quantità di parametri da memorizzare e ottimizzare. Inoltre, anche la potenza computazionale è notevole, infatti un dato in input viene presentato a tutte le colonne già istanziate della rete. Al contempo soltanto una porzione dell'effettiva capacità viene utilizzata. Se un task precedente non produce alcun effetto benefico sul task passato esso non verrà utilizzato fino al termine del training e questa tendenza aumenta all'aumentare dei task. È possibile concludere che le PNN costituiscono uno strumento potenzialmente molto potente per la risoluzione di problemi di CL, anche applicati a serie temporali se esse sono dotate di identificatore del task, ma presenta limiti su memoria, scalabilità e potenza/tempo di computazione che lo rendono decisamente meno allettante e conveniente rispetto ad altre tecniche di learning, anche nei confronti di tecniche di regolarizzazione.

### 4.7.2 Learning without Forgetting

Molte applicazioni di deep learning richiedono un apprendimento continuo di nuovi task e al contempo il mantenimento delle conoscenze pregresse. Per garantire quest'ultimo aspetto i metodi architetturali espandono la rete neurale in modo da non modificare i parametri di task passati, come accade nelle PNN. Il primo requisito, invece, può essere soddisfatto in diversi modi, partendo da una rete neurale che condivide tutta la *backbone* con parametri $\theta_s$ e presenta un *output head* (con parametri $\theta_k$) riservata ad ogni task $k$ (Figura 4.7 a):

- *Fine-tuning*: Si ottimizzano tutti i parametri $\theta_s$ della backbone, così come la nuova head $\theta_k$, mentre le heads dei task passati rimangono invariate (Fig. 4.7 b).
- *Feature Extraction*: Gli unii parametri ad essere ottimizzati sono quelli del task corrente $k$ (Fig. 4.7 c).
- *Joint Training*: Tutti i parametri della rete vengono ottimizzati in un contesto di training simultaneo dei task, per esempio intervallando il

training. Può essere considerato un *upper bound* delle performance ottenibili (Fig. 4.7 d).

È possibile combinare tutti questi metodi, in modo tale da allenare la rete solamente sul task corrente ma ottimizzando tutti i parametri della backbone $\theta_s$, del task corrente $\theta_k$ e dei task passati $\theta_j$ con $j<k$. Questo metodo è chiamato *Learning without Forgetting* [21]. Esso è il metodo più simile al joint training ma con il training sequenziale dei task. Se nell'head riservata al task corrente si effettua un training standard, in tutte le head dei task passati viene effettuato uno pseudo fine-tuning con gli esempi del task corrente. Questa tecnica si è rivelata inoltre un'ottima strategia di regolarizzazione per i task passati, soprattutto in un contesto dove gli output di tutti i task condividono lo stesso dominio.

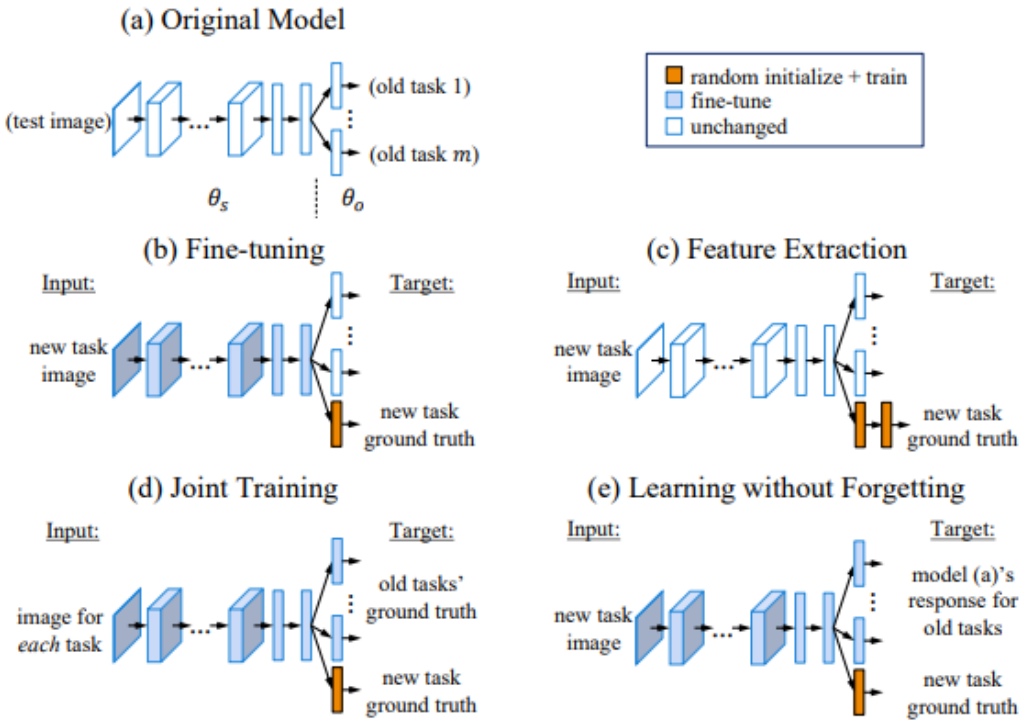



Figura 4.7: Comparativa tra LwF e altri metodi di learning

Quando si presenta un nuovo task *k* avviene una fase detta di *setup*: viene istanziata una nuova head con inizializzazione casuale dei suoi parametri $\theta_k$ e al contempo vengono raccolti gli output $y_j$ con $j=1,k-1$ dalle head dei vecchi task. Essendo un problema di classificazione viene salvato per ogni head il vettore di probabilità per ogni esempio di training, per un totale di $(k-1)\cdot N$ output, dove $N$ è il numero di elementi che compongono il training set. Successivamente avviene il *warm-up* della nuova head, con un training di un basso numero di epoche della sola head del task corrente, "congelando" i parametri $\theta_s$ e $\theta_j$. Quest'operazione permette di aver un miglior punto di partenza per la fase successiva, ovvero il vero e proprio training della rete neurale nella sua completezza. Durante il training saranno necessarie di diverse loss function, una per l'head corrente e un'altra per le head dei task passati:

$$L_{new}(y_k, \widehat{y_k}) = -y_k \cdot \log(\widehat{y_k}) \quad e \quad L_{old}(y_o, \widehat{y_o}) = -\sum_{j=1}^{k-1} y_j \cdot \log(\widehat{y_j})$$

$$argmin_{\theta_s, \theta_k, \theta_j} \ \alpha \cdot L_{old}(y_o, \widehat{y_o}) + \beta \cdot L_{new}(y_k, \widehat{y_k})$$

con $\widehat{y_k}$ e $\widehat{y_j}$ l'output proveniente rispettivamente dall'head corrente e dalle head dei task passati, $y_k$ le *true label* dei dati del training set e infine $y_j$ gli output delle head dei task precedenti raccolti durante la fase di setup. Gli iperparametri α e β regolano l'importanza delle due loss (di default α=β=0.5). Se quindi per il task corrente avviene il training classico, nelle head dei task precedenti l'obiettivo è minimizzare la differenza tra le predizioni originali e quelle al termine del training del task corrente, in modo tale da imporre un cambiamento di parametri che portino allo stesso output pre-tuning. Questo si può rivelare anche un ottimo strumento di regolarizzazione e consolidazione per i task passati.

LwF presenta innumerevoli vantaggi rispetto alle metodologie standard dal punto di vista delle performance, superando la feature extraction e performando a livello del fine-tuning. Anche la computazione non è troppo onerosa in quanto le fasi di setup e warm-up coinvolgono solo una piccola parte della rete e durante il training dei task il fine-tuning delle head appartenenti a task passati riguarda solamente pochi layer di classificazione. L'utilizzo della memoria è efficiente siccome non richiede di salvare esempi di task passati come i metodi di rehearsal e singolarmente le head dei task non sono complesse e hanno un numero di parametri contenuto. Se però il numero dei task è molto ampio il numero totale dei parametri potrebbe diventare un vincolo impattante non solo sulla memoria ma anche sulla scalabilità. Nel complesso il LwF costituisce una migliore alternativa alle PNN come metodo architetturale e potrebbe competere con altre tecniche di CL, anche se possiede evidenti limitazioni dovute all'esigenza di boundaries tra task e dunque un identificativo per essi.

# Capitolo 5

# Risultati

L'applicazione di tecniche di Continual Learning a serie temporali e in particolare al mercato finanziario hanno portato a risultati interessanti per quanto riguarda l'apprendimento dei diversi regimi che una serie può assumere nel corso del tempo. In base all'architettura e al metodo adottati sono osservabili una moltitudine di comportamenti differenti. Nell'ambito di questa tesi si è cercato di coprire nel miglior modo possibile lo spettro delle soluzioni per quanto riguarda la scelta degli iperparametri più importanti degli algoritmi adottati.

Si presentano in questo capitolo le modalità di valutazione dei risultati, introducendo alcune problematiche specifiche riguardo la compatibilità tra gli algoritmi di Continual Learning all'interno del mondo finanziario. Si offre in seguito una ampia vista nell'ambito dell'esplorazione degli iperparametri e del loro impatto sui risultati finali.

## 5.1 Considerazioni preliminari

Prima di precedere all'analisi dei risultati reali, si ritiene necessario dedicare attenzione ad alcune considerazioni che contraddistinguono il mondo della classificazione di immagini, dominio di studio nel quale il Continual Learning ha ottenuto migliori risultati. Innanzitutto, si osserva la differenza di comportamento a fronte della variazione degli iperparametri all'interno del training, validation e test set. Si prosegue poi con alcune considerazioni inerenti al mercato finanziario e all'impossibilità si prevedere alcuni eventi al suo interno.

### 5.1.1 Fase di training

Nel caso del training set è spesso possibile osservare con facilità le ripercussioni dirette durante l'esplorazione dei vari iperparametri. Esso, infatti, rispecchia quello che può essere la classificazione d'immagini, dove il modello osserva e impara pattern in un

dominio con potenzialmente un numero moto grande ma finito di stati. Come esso impara a classificare rispettando vincoli prestabiliti, la stessa cosa accade anche nell'ambito finanziario, poiché l'osservazione continua dei dati li permette di apprendere nel migliore dei modi i pattern presenti al loro interno. Per questo motivo il cambiamento degli iperparametri di apprendimento incide direttamente sui risultati riguardanti il set di training offrendo la possibilità di confrontare al meglio le varie configurazioni e distringere i risultati tra loro.

### 5.1.2 Fase di validation

È estremamente complesso effettuare osservazioni in questa fase. Il modello non ha alcun modo di allenarsi né sul set di validazione né sul set di test e semplicemente il validation set serve per effettuare la validazione dei parametri appresi durante il training su dati non visti, per capire quanto i pattern imparati possano essere generalizzabili.

Non sempre è possibile riscontrare una corrispondenza precisa per quanto riguarda il training e validation set e ogni configurazione degli iperparametri porta a differenti comportamenti soprattutto sul validation set.

Una considerazione simile si può fare anche tra validation e test set. L'osservazione più evidente è data del fatto che un'ottima performance sul validation set, e quindi il salvataggio della strategia, spesso si traduce in una prestazione pessima sul test set, risultando quindi un evidente caso di overfitting sul validation set. Questa situazione non è rara, accade infatti spesso che nelle prime epoche di training, momento in cui il modello ha solo una visione parziale del problema da affrontare, esso abbia dei picchi di performance durante la validazione. Gli stessi picchi non vengono riscontrati sul test set poiché la strategia appresa non è sufficientemente "generale", ma è solamente una configurazione dei parametri della rete neurale che porta a ottenere risultati di rilievo in maniera fortuita senza rispecchiare gli stessi risultati su altre configurazioni di input.

### 5.1.3 Fase di test

Teoricamente i risultasti del test set sono quelli più rilevanti poiché provengono da dati mai osservati prima dal modello e rispecchiano le reali potenzialità del sistema su dati futuri e mai usati come input come in un ambiente di lavoro. Di conseguenza i risultati mostrati in seguito dal punto di vista delle metriche vengono calcolati solamente su questo set di dati. La differenza di prestazioni rispetto ai dati che il modello ha avuto modo di osservare per molte epoche di training è abissale ma gli sforzi maggiori sono stati senza dubbio quelli riguardanti la ricerca di una configurazione che permettesse di ottenere i migliori risultati sul test set.

### 5.1.4 Considerazioni sul mercato finanziario

Il mercato finanziario, sia dal punto di vista azionario che delle *commodities*, è altamente imprevedibile e influenzabile da diversi fattori. È possibile rilevare nelle varie economie una serie di eventi molto particolari che possono portare gli andamenti di

prezzo delle materie prime o metalli rari a subire variazioni casuali e assolutamente imprevedibili dal punto di vista algoritmico. Questo accade principalmente per due motivi:

- Imprevedibilità dell'evento: si possono verificare eventi di varia natura non ipotizzabili a priori che vanno a influenzare a cascata tutti i mercati, inclusi quelli apparentemente non correlati agli eventi verificatisi.
- Imprevedibilità della variazione del mercato: accade spesso che non vi sia alcuna correlazione tra l'entità dell'evento e la reazione dei mercati. Di conseguenza due eventi molto simili potrebbero portare a movimenti differenti come due eventi ben distinti potrebbero portare alle stesse reazioni.

Si sono dunque identificati una serie di tipologie di eventi che vanno a influenzare in maniera più o meno marcata le imprevedibilità dei mercati.

- In primo piano vi sono le occasioni durante le quali un'alta personalità a livello economico o politico mondiale parla. Spesso qualsiasi informazione divulgata in ambito finanziario da parte di autorità di grande influenza causa una fluttuazione dei mercati.
- Altro fattore importante sono le decisioni impreviste in ambito finanziario o bancario per quanto riguarda direttamente l'economia mondiale o regionale. In questo caso si riporta un evento concreto che ha scaturito un cambio di regime e di conseguenza un changepoint, ovvero la decisione dell'OPEC di tagliare i prezzi del petrolio al barile per fare concorrenza alle grandi compagnie USA e UK [36].
- Un'altra tipologia molto incisiva sono gli eventi in ambito politico, poiché il panorama politico di una nazione svolge un ruolo di primaria importanza per quanto riguarda l'economia dei diversi paesi. Considerando il legame strettissimo tra l'economia di una paese e la forza del paese stesso e delle sue compagnie sul mercato, anche gli eventi politici hanno un riscontro diretto sugli andamenti dei mercati, finanziari come quelli delle commodities. Si possono trovare esempi concreti dal punto di vista finanziario, dove a seguito della decisione politica dell'allora presidente USA Donald Trump di imporre dazi sui prodotti importati dalla Cina si è osservato un periodo di incertezza per tutte le compagnie americane coinvolte nell'importazione di prodotti o componenti. O ancora, la promessa elettorale di riaprire le miniere di carbone in Ohio durante la campagna elettorale di Trump ha portato ad una variazione nel prezzo di tale materia prima.
- Ultimi ma non meno importanti risultano vari fattori legati a cause esterne o fatti di cronaca mondiale. Anch'essi vanno a influenzare le economie dei paesi sotto vari punti di vista. Un esempio può essere l'attentato al World Trade Center dell'11 settembre 2001 a New York. Un evento di tale entità risulta totalmente imprevedibile e ha impatti devastanti sull'economia mondiale. Infatti, a seguito di quest'ultimo, i mercati azionari statunitensi e non solo hanno subito un crollo che ha sancito un cambio di andamento e che ha richiesto molto tempo prima che il mercato potesse tronare ai

livelli pre-attentato. Il crollo si va a ripercuotere su vari livelli, per esempio il valore delle azioni delle compagnie aeree, crollato in seguito allo stop totale dei voli imposto a seguito dell'attentato e la paura dei viaggiatori nel periodo immediatamente successivo di incorrere in nuovi dirottamenti.

### 5.1.5 Considerazioni sui metodi utilizzati

L'obiettivo di questa tesi è la valutazione di algoritmi di Continual Learning applicati a serie temporali e le potenzialità che esso può avere nell'ambito finanziario. Di conseguenza si è deciso di utilizzare la maggior parte degli algoritmi, ove possibile, che compongono lo stato dell'arte in materia. Per ogni "famiglia" di metodi si sono voluti provare più algoritmi in modo tale da osservare diversi metodi. Non si è badato però in alcun modo all'integrazione tra più algoritmi o con altre tecniche proposte che potrebbero portare grandi vantaggi nei risultati finali. Inoltre, non si è approfondita la previsione di possibili avvenimenti descritti in precedenza ed essi potrebbero sicuramente portare a periodi di grande variazione difficilmente valutabile correttamente per il modello. Si discuterà in seguito dunque dei possibili sviluppi futuri del progetto anche in quest'ottica, considerando diversi accorgimenti per poter implementare altre metodologhe ora non applicabili.

### 5.1.6 Metodi e metriche di valutazione

La valutazione degli algoritmi viene fatta in diverse forme. Prima di tutto si effettua una valutazione delle diverse configurazioni di MLP e CNN proposte applicando il learning sequenziale dei task senza alcun metodo di Continual Learning. Questo passaggio è necessario per definire una configurazione per ogni tipologia da eleggere come configurazione finale. Successivamente verranno testati tutti i metodi per entrambe le reti proposte, confrontando i risultati grazie all'uso di grafici e tabelle tratti dalla valutazione sul test set. Le metriche di valutazione usate sono quelle che caratterizzano il Continual Learning, definite nel paragrafo 2.4.

## 5.2 Valutazione configurazioni

In questa sezione si testano e si discutono i risultati delle configurazioni proposte in precedenza, sia per il MLP che per le CNN. Si vedranno in seguito anche alcune variazioni degli iperparametri che definiscono il learning con il MLP, come epoche di training, learning rate scelto e percentuale di dropout negli hidden layer. Si è deciso di non effettuare quest'operazione sulle CNN in quanto il loro sviluppo non è stato sufficientemente approfondito, lasciando questo aspetto come uno degli sviluppi futuri.

Sebbene siano state testate diverse serie temporali per i test sono stati presi in considerazione soltanto i risultati provenienti da una di esse: la serie che descrive

l'andamento dei prezzi del petrolio grezzo giornalmente. Questo perché è una serie che racchiude al suo interno un numero sufficiente di changepoint, e quindi possibili task, per valutare nella loro completezza gli algoritmi di Continual Learning. Infatti, alcune serie prese in considerazioni non hanno portato a risultati attendibili oppure non sono stati rilevati dall'algoritmo di changepoint detection cambi di regime ritenuti attendibili.

Come detto in precedenza, il dataset finale è composto dalla serie originale più tutte le features derivate da indicatori finanziari, come illustrato nel capitolo 3.3. Il numero finale di indicatori (e quindi features) è volutamente non molto alto per non rischiare di incappare nell'overfitting. Al contempo potrebbe esser possibile usare indicatori specifici sostituendoli a quelli presenti in base alle serie storiche finanziare da voler analizzare, favorendo così caratteristiche proprie di quest'ultime.

### 5.2.1 Configurazione migliore MLP

Le configurazioni proposte per il Multi Layer Perceptron sono quelle illustrate nel capitolo 3.6. Si è deciso di testare i modelli non applicando nessun algoritmo di Continual Learning ma bensì in un ambiente di Online Learning, allenando in sequenza i task esponendo così il modello a catastrophic forgetting. Scegliere un algoritmo di Continual per testare le diverse configurazioni porterebbe portare alla scelta di una architettura che ben si sposa con un determinato algoritmo, mentre l'obiettivo è scegliere un'architettura il più generica possibile. Al contempo è impossibile testare tutte le configurazioni per tutti i metodi di Continual Learning per un discorso di tempistiche e utilità. Ogni task viene allenato per un totale di 300 epoche.

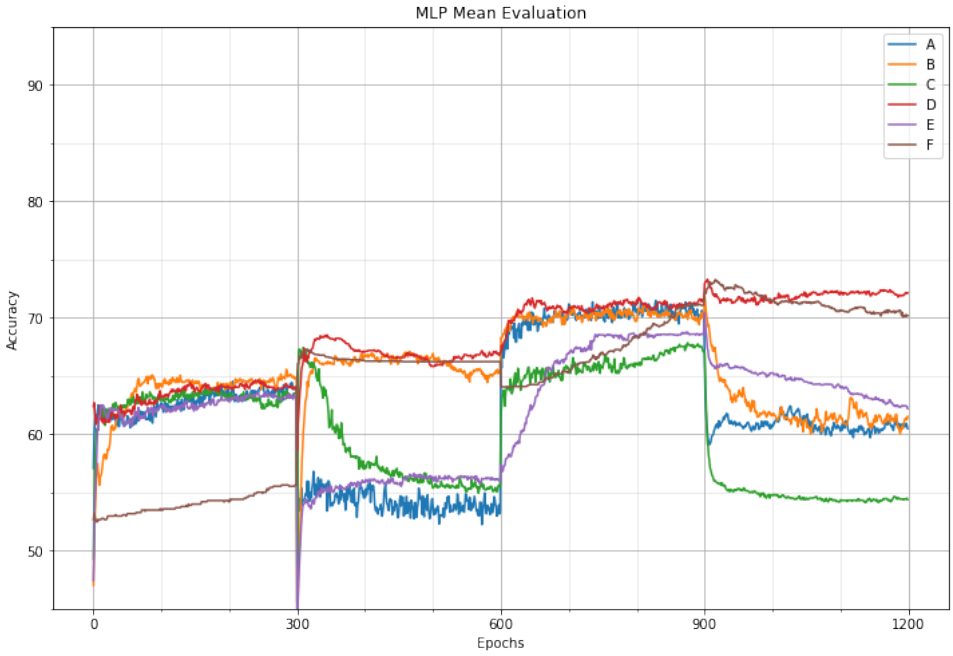



Figura 5.1: Valutazione MLP

Le varie configurazioni sono state valutate tramite l'accuracy media tra i task. Di conseguenza soltanto i risultati provenienti dalle epoche [900:1200] sono significativi

siccome la valutazione avviene su tutti i task. Ciò nonostante, è interessante osservare come varia la media nel cambio di task in base alla configurazione adottata: in alcune di queste la media crolla repentinamente e ciò può essere dovuto ai pochi parametri a disposizione del modello, dove una variazione piccola dei loro valori porta a sovrascrivere in maniera importante il task precedente.

Interessante anche il comportamento dei modelli durante il secondo task affrontato (epoche [300:600]). La metà di essi ha dimostrato performance appena superiori al 50% e ciò indica sia la complessità del task, riscontrabile anche nella valutazione dei singoli algoritmi di Continual Learning successivamente, ma anche la scarsa resistenza al forgetting dei modelli. Ad eccezione della configurazione E, le altre due, ovvero A e C si sono dimostrate le più deboli in entrambi i frangenti.

Come anticipato nella discussione delle architetture, la configurazione D è quella che ha mostrato migliori performance e maggiore resistenza al forgetting, venendo scelta come configurazione standard per quanto riguarda il MLP. Sebbene la configurazione F abbia performance paragonabili, durante la valutazione del modello si è osservato un principio di overfitting sugli ultimi task, non troppo rilevante in questo contesto ma con un impatto potenzialmente devastante in serie finanziare con un maggior numero di task al loro interno. Le configurazioni C ed E, rispettivamente la seconda e terza per numero di parametri, hanno anch'esse mostrato un'overfitting molto marcato che ha portato ad un crollo importante nell'ultimo task. Infine, le configurazioni più leggere dal punto di vista dei parametri, A e B, hanno mostrato una scarsissima capacità di apprendere task nuovi senza sovrascrivere totalmente i precedenti. È importante ricordare che in questa valutazione si è deciso di usare l'Online Learning, accettando il rischio del forgetting, dove l'obiettivo è stato la ricerca di un modello che potesse gestire questo rischio anche dal punto di vista architetturale, con un numero sufficiente di parametri tale da minimizzare il suo impatto senza compromettere la qualità del learning e al contempo non manifestare le problematiche tipiche del deep learning, overfitting e insufficiente capacità di learning.

### 5.2.2 Configurazione migliore CNN

Le premesse sulle configurazioni sono le medesime effettuate per i Multi Layer Perceptron. In questo caso però si ha maggiore possibilità di personalizzazione grazie ai molteplici parametri della convoluzione. L'obiettivo è usare questa operazione per osservare ed estrarre features da diversi intervalli temporali della serie. Per fare ciò è necessario agire sulla dimensione del kernel e sullo stride. Come illustrato in precedenza le configurazioni proposte agiscono unicamente su questi due parametri. Esse inoltre presentano lo stesso numero di layer, ovvero 3 convolutivi e 2 di classificazione lineari, ed un numero superiore è dimostrato essere deleterio con una serie temporale in input.

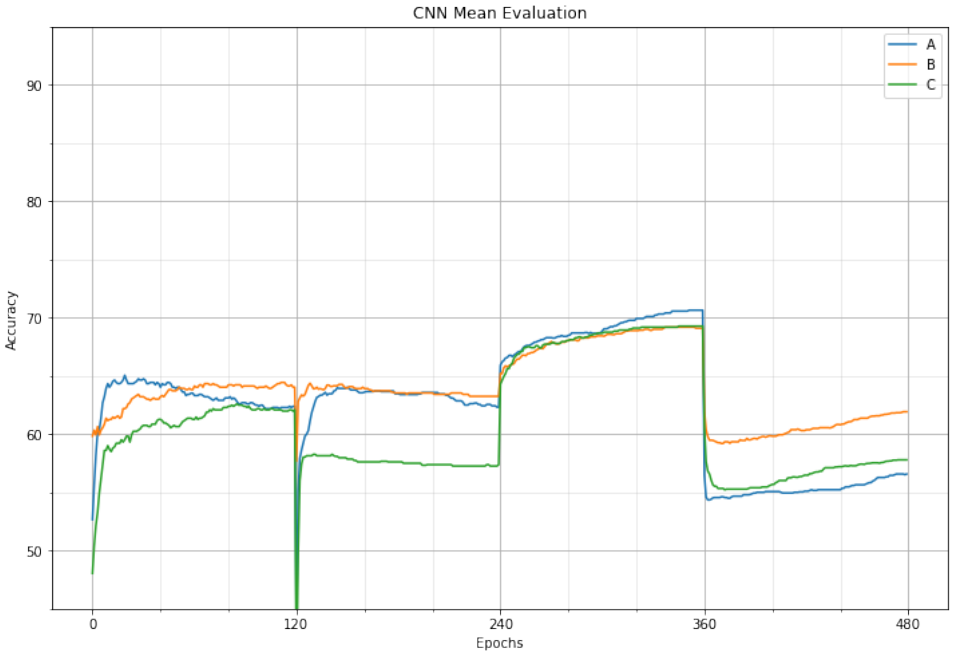



Figura 5.2: Valutazione CNN

Innanzitutto, il numero di epoche di training per le CNN è sensibilmente inferiore rispetto al MLP (120 rispetto a 300). Sebbene la struttura dei layer delle reti convolutive sia più complessa sono usati kernel di dimensioni piccole che necessitano di meno cicli sul training set per essere ottimizzati, dove un numero di epoche uguale al MLP avrebbe sicuramente portato all'overfitting della rete.

Le variazioni tra le configurazioni non sono molto marcate e ciò può trovare conferma nella figura 5.2. In tutte le configurazioni i primi due layer convolutivi sono uguali e, tra A e le configurazioni B e C, varia il numero di feature in output. In seguito, è necessario trasformare la matrice risultante dal blocco convolutivo in un vettore monodimensionale per permettere la classificazione. All'interno del blocco lineare le configurazioni B e C variano in base al numero di features (e quindi la dimensione del vettore).

Le performance ottenute con l'Online Learning sono tutt'altro che entusiasmanti, sintomo del fatto che le CNN, avendo molti meno parametri rispetto al MLP e la maggior parte di essi situati nei layer di classificazione, siano reti particolarmente soggette al catastrophic forgetting. Anche le differenze tra le configurazioni non risultano così marcate, anche se tra queste spicca la configurazione B. Rispetto alla configurazione A possiamo dire che quest'ultima probabilmente non presenta un numero sufficiente di parametri mentre la configurazione C ha mostrato un aumento della loss nella fase di testing, vale a dire ha manifestato forgetting.

## 5.3 Valutazione algoritmi con MLP

In questa sezione vengono analizzate le performance di ogni algoritmo di Continual Learning utilizzato in questa tesi. Lo studio delle performance avverrà in maniera

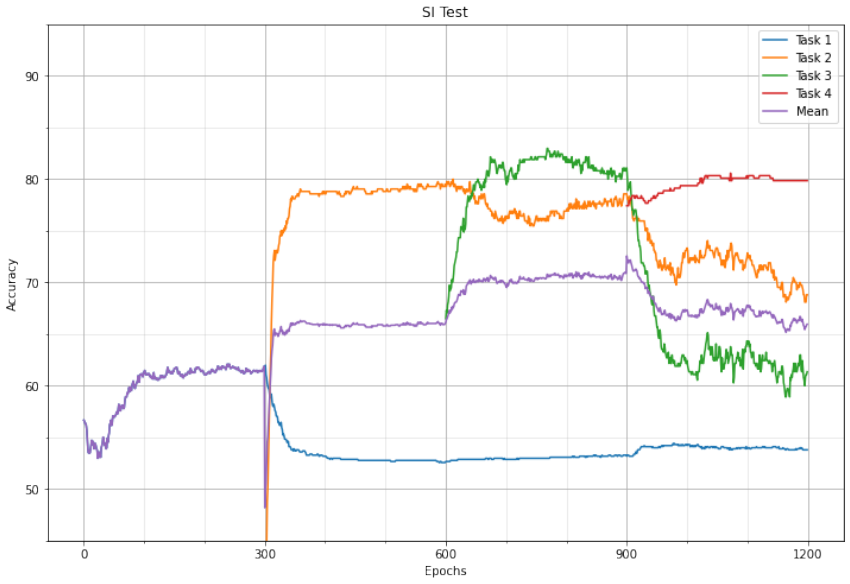
distinta per ogni famiglia di algoritmi in modo da determinare quello che si adatta meglio al problema affrontato. Successivamente verranno paragonati i migliori metodi per decretarne l'ipotetico da utilizzare in un sistema di produzione. Infine, si avrà una breve analisi dei tempi di training per ogni algoritmo, strumento utile a capirne la complessità e che potrebbe facilitare la scelta tra due o più algoritmi se questi manifestassero performance simili.

Nell'analisi dei vari metodi verranno usate le performance medie dell'Online Learning come riferimento e *lower bound*. Alcuni metodi avranno performance medie simili se non inferiori ad esso ma ciò non significa necessariamente che il metodo non è applicabile o errato, ma che la configurazione usata in questa tesi non si sposa con il problema in essere. Nel capitolo successivo saranno discusse possibili variazioni per evitare il problema.

I diversi algoritmi non verranno valutati solamente secondo l'accuracy ma anche attraverso tutte le metriche proprie del Continual Learning, ovvero *forward transfer* (FWT), *backward transfer* (BWT) e *forgetting* (FRG).

### 5.3.1 Metodi di regolarizzazione

Nella figura sottostante è riportata l'accuracy di ogni task e l'accuracy media per gli algoritmi di regolarizzazione, Elastic Weight Consolidation e Synaptic Intelligence.



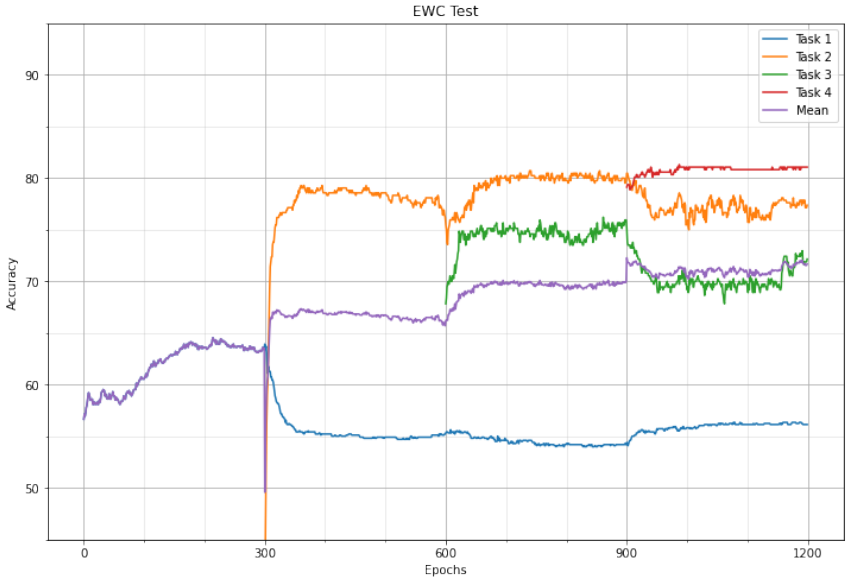
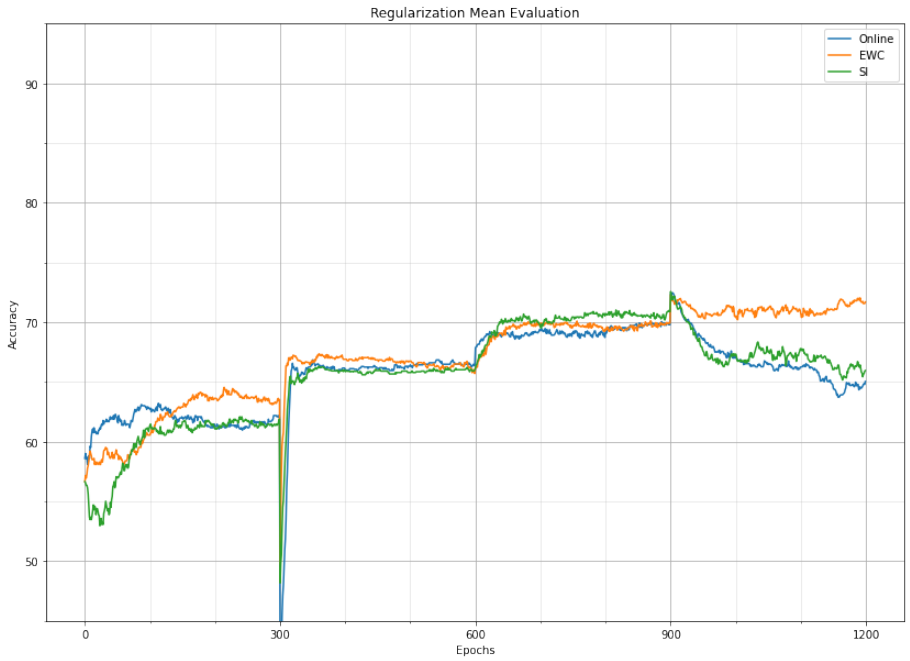



Figura 5.3: Metodi di regolarizzazione con MLP; sopra accuracy SI, sotto EWC

Il primo metodo, SI, aggiorna i parametri del modello ad ogni iterazione e ciò è osservabile anche all'interno di un singolo task. È possibile osservare come il presentarsi di un nuovo task porta alla riduzione dell'accuracy sui task precedenti. Ciò può essere giustificato dal fatto che ogni neurone ha un impatto importante sulla loss del task, e quindi l'algoritmo "congela" i neuroni coinvolti oppure che è necessario modificare in ogni caso i parametri, aspetto interessante perché la penalità introdotta dalla modifica porta in ogni caso ad una diminuzione della loss globale del task. Ciò nonostante, l'accuracy media aumenta ad ogni task, ad eccezione dell'ultimo e, presa singolarmente l'accuracy di ogni task si osservano dei cali in prossimità del cambio di task ma le performance vengono mantenute al di sopra del 50% (ovvero la casualità) e non si osserva il catastrophic forgetting.

Per quanto riguarda EWC invece, l'aggiornamento dei parametri rilevanti avviene al termine di ogni task. Come per SI, il primo task osservato in ordine temporale risulta anche quello più penalizzato, mentre le performance sugli altri task si mantengono stabili grazie ad una miglior analisi data dalla matrice di Fisher. Grazie a questa al termine di ogni task si aggiornano i parametri rilevanti per il task corrente, andando a modificare solo parzialmente i parametri rilevanti per gli altri task.




| Algoritmo | Accuracy (%) | BWT (%) | FWT (%) | FRG (%) |
|---|---|---|---|---|
| Online | 65,04 | - | - | - |
| SI | 65,96 | -12,06 | 25,28 | 12,06 |
| EWC | 71,64 | -4,15 | 26,02 | 4,15 |

Figura 5.4 e Tabella 5.1: Confronto metodi di regolarizzazione con MLP

Osservando anche le metriche di Continual possiamo notare come SI performi male in termini di accuracy, avvicinandosi pericolosamente all'Online Learning, considerato un *lower bound* in questi esperimenti. Al contempo EWC è superiore in ogni metrica a SI, soprattutto dal punto di vista del *backward transfer*, metrica molto più significativa del *forward transfer*, quest'ultimo calcolato con una configurazione randomica della rete e dunque non così affidabile. Il metodo di regolarizzazione scelto come candidato finale è dunque EWC.

### 5.3.2 Metodi di rehearsal

Sono riportate le performance dei due metodi replay-based utilizzati in questa tesi, Experience Replay e Dark Experience Replay. La differenza tra i due dal punto di vista implementativo risiede nella tipologia di dato salvato nel buffer di memoria, la coppia esempio-label $(x,y)$ per ER mentre in DER vengono salvati al posto delle label gli output non scalati dell'ultimo layer di classificazione pre-softmax.

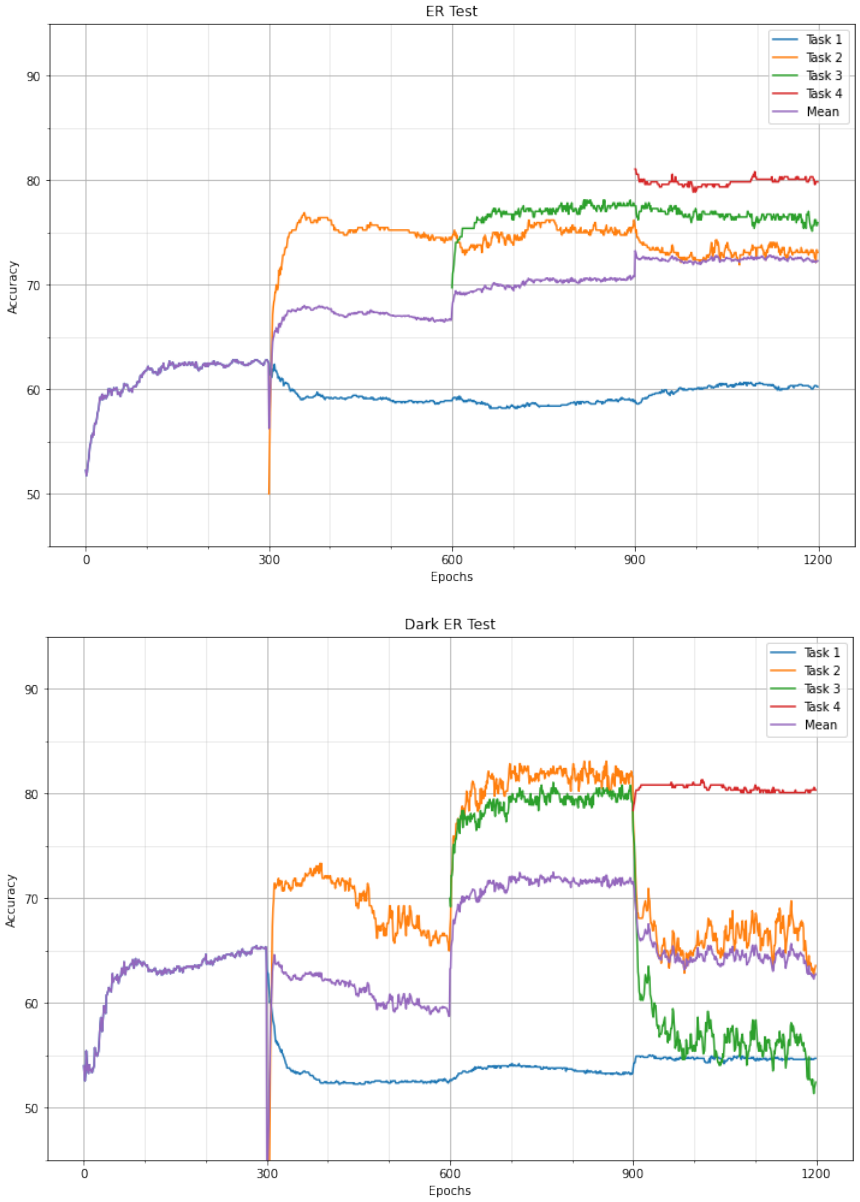
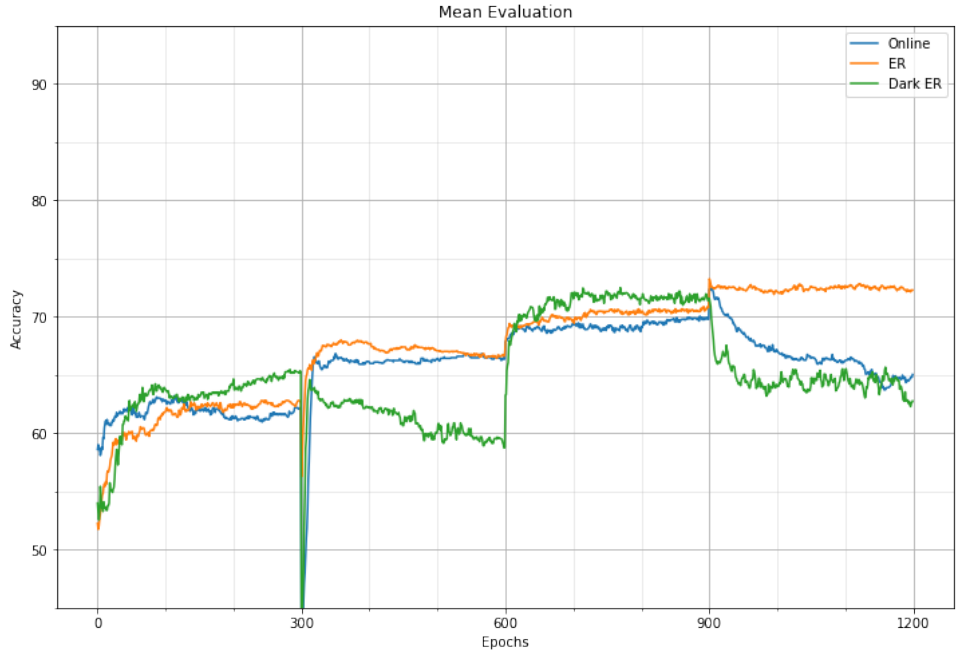



Figura 5.5: Metodi di rehearsal con MLP; sopra ER e sotto DER

La prima differenza tra i due algoritmi la si può notare dal comportamento dei task; in DER i task passati manifestano, seppur in maniera molto contenuta, il forgetting, mentre in ER ciò non accade mai e anzi l'accuracy sui task passati rimane stabile, portando un aumento considerevole all'accuracy media. Di norma ci si aspetterebbe lo stesso comportamento anche da DER, ma il salvataggio dei logits e il vincolo della loss imposto dall'algoritmo potrebbero aver portato il modello ad una sorta di overfitting, dove esso non impara dagli esempi campionati dal buffer di memoria ma si limita a modificare i parametri per mantenere i logits in output simili a quelli del buffer. Al contrario, ER riesce a mantenere conoscenza dei task passati grazie all'osservazione degli esempi. Questo compito probabilmente è facilitato dal fatto che il dominio degli output tra i task sia sempre uguale, ovvero che le possibili label siano sempre 0 o 1. Anche il fatto che si stia facendo Continual Learning a livello di *domain-IL* aiuta non poco l'algoritmo, non dovendo fare inferenza sul task di appartenenza degli esempi del buffer.

DER ha performance addirittura peggiori del lower bound definito dall'Online Learning e ritengo che questo sia dovuto esclusivamente all'overfitting trattato in

precedenza. Applicato alle immagini DER performa alla stessa maniera, se non meglio, dell'Experience Replay classico ma non è detto che ciò sia valido anche per le serie storiche finanziarie.




| Algoritmo | Accuracy (%) | BWT (%) | FWT (%) | FRG (%) |
|---|---|---|---|---|
| Online | 65,04 | - | - | - |
| ER | 72,29 | -3,72 | 24,05 | 3,72 |
| Dark ER | 64,41 | -12,91 | 27,24 | 12,91 |

Figura 5.6 e Tabella 5.2: Confronto metodi di rehearsal con MLP

### 5.3.3 Metodi ibridi

Nelle figure sottostanti è possibile osservare l'accuracy dei metodi ibridi proposti, ovvero la Gradient Episodic Memory e le sue varianti A-GEM con la metodologia classica e con il reservoir sampling.

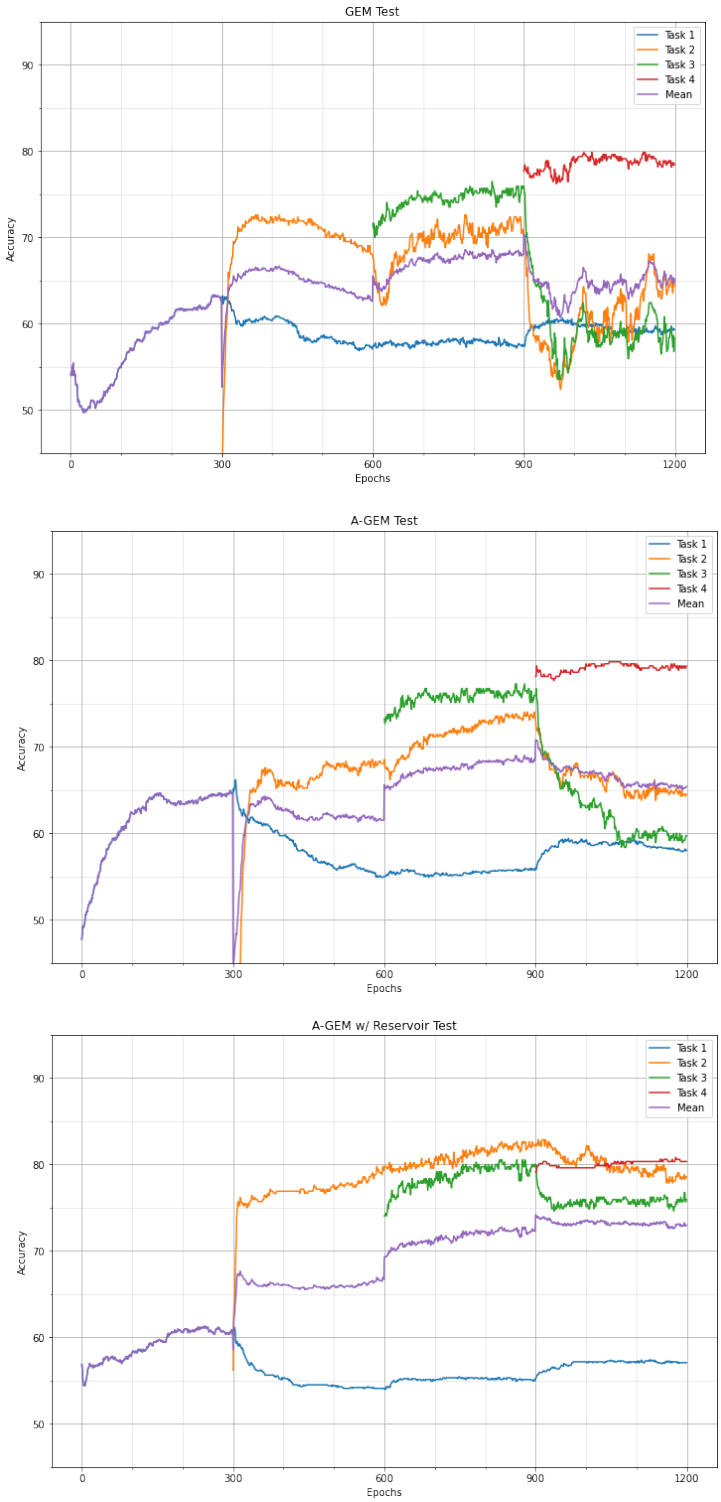


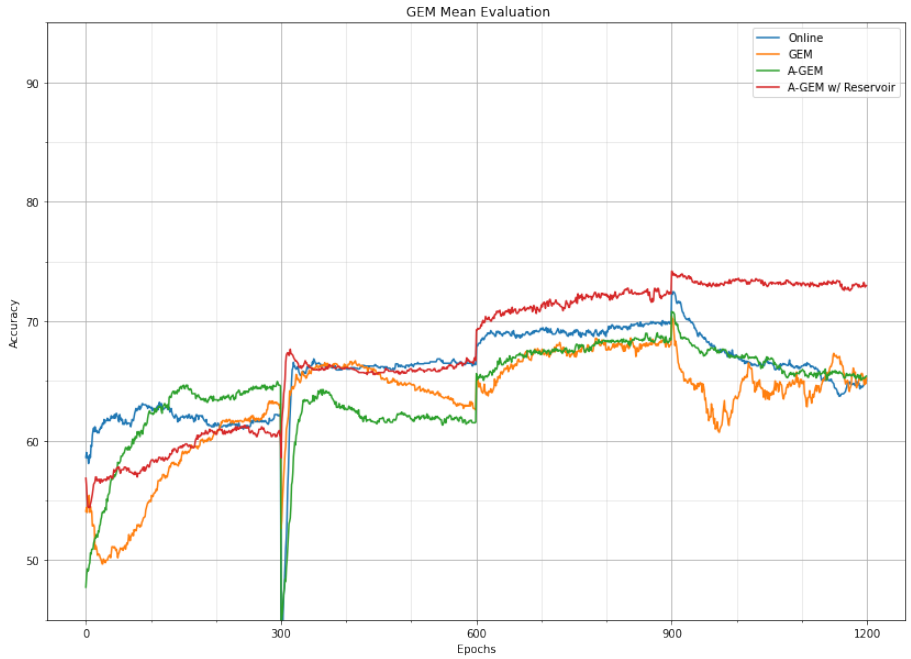

Figura 5.7: Metodi ibridi con MLP; in ordine GEM, A-GEM e A-GEM w/ Reservoir

In GEM è immediato l'andamento dei task all'aumentare delle epoche totali di training; infatti, in corrispondenza degli ultimi due task si osserva un comportamento altamente altalenante. Questo era auspicabile dato il tipo di algoritmo; GEM, infatti, pone come vincolo durante l'ottimizzazione il non aumento della loss sulla batch di esempi salvati nella piccola memoria a disposizione e, in caso questo non venga

rispettato, si ricorre alla ricerca di un passo del gradiente del task corrente che rispetti tutti i vincoli trovandosi dinanzi ad un problema NP-C e alla sua soluzione approssimata.

Questo comportamento viene fortemente limitato nell'A-GEM dove l'unico vincolo è dato dalla media delle loss. Ciò nonostante, si ha un peggioramento delle performance all'aumentare dei task, anche in questo caso è dovuto al non rispetto dei vincoli e alla risoluzione del problema con il gradiente proiettato che offre una soluzione accettabile ma non ottima, portando al rapido decadimento delle prestazioni.

Nell' A-GEM realizzato con il reservoir sampling invece, non è presente affatto questo comportamento. Questo significa che la batch campionata casualmente dalla memoria, quest'ultima che possiede una distribuzione fedele all'originale tra i task, possiede elementi di un task con spazio di soluzioni simili al task corrente e quindi l'aggiornamento del gradiente è valido per entrambe le batch. La scelta dell'algoritmo di riempimento della batch è quindi significativo e potrebbero essere testati altri algoritmi per valutarne l'efficacia.




| Algoritmo | Accuracy (%) | BWT (%) | FWT (%) | FRG (%) |
|---|---|---|---|---|
| Online | 65,04 | - | - | - |
| GEM | 65,25 | -11,29 | 12,21 | 11,29 |
| A-GEM | 65,47 | -5,74 | 25,97 | 5,74 |
| A-GEM w/ Reservoir | 72,98 | -7,60 | 25,75 | 7,60 |

Figura 5.8 e Tabella 5.3: Confronto metodi ibridi con MLP

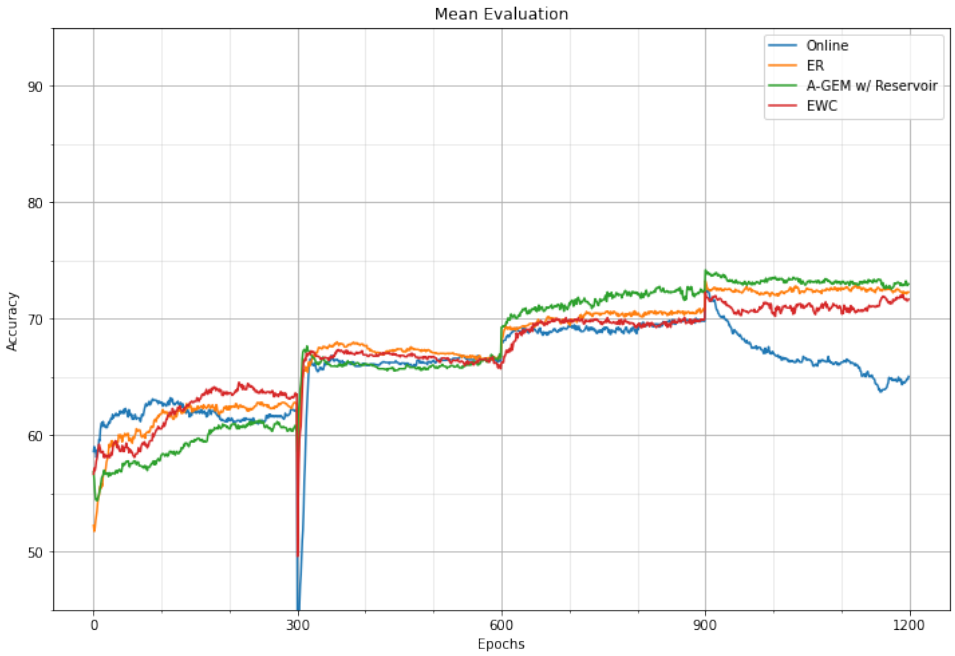
GEM e A-GEM classico mostrano un accuracy media simile al riferimento dell'Online Learning, mentre A-GEM con il reservoir sampling consente di ottenere risultati soddisfacenti. Uno degli obiettivi degli algoritmi basati sul GEM è la massimizzazione del backward transfer oltre all'accuracy. Questo non viene rispettato

dall'algoritmo originale se applicato a serie temporali, ma le sue evoluzioni e in particolare la variante con il reservoir sampling consentono sia di ottenere un backward transfer migliore dell'originale, e in linea con i migliori algoritmi delle altre "famiglie" trattati in precedenza, sia di registrare un ottimo forward transfer.

In conclusione, GEM e A-GEM non risultano algoritmi validi a causa dei vincoli sempre più stringenti da rispettare all'aumentare del numero dei task analizzabili, mentre A-GEM implementato con il reservoir sampling si è dimostrato un algoritmo robusto al forgetting e al contempo capace di apprendere nuovi task senza manifestare un calo in quelli precedenti; pertanto, è stato scelto come metodo di riferimento tra quelli ibridi.

### 5.3.4 Considerazioni finali

Dopo un'analisi dei principali metodi applicati ad un MLP è possibile affermare che almeno un metodo per ogni famiglia permette di risolvere in maniera efficace, anche se non sempre efficiente, il problema del Continual Learning applicato a serie storiche finanziarie. Di seguito sono analizzati i metodi migliori paragonati al learning sequenziale dei task:




| Algoritmo | Accuracy (%) | BWT (%) | FWT (%) | FRG (%) |
|---|---|---|---|---|
| Online | 65,04 | - | - | - |
| EWC | 71,64 | -4,15 | 26,02 | 4,15 |
| ER | 72,29 | -3,72 | 27,05 | 3,72 |
| A-GEM w/ Reservoir | 72,98 | -7,60 | 25,75 | 7,60 |

Figura 5.9 e Tabella 5.4: Performance medie metodi migliori

Le performance tra questi metodi sono molto simili tra loro e pertanto non è possibile determinare un metodo in grado di superare i concorrenti, aspetto tra l'altro non ricercato all'interno di questa tesi. È dunque impossibile decretare il metodo assoluto da applicare alla classificazione di serie storiche finanziarie ma è possibile effettuare alcune considerazioni che potrebbero portare all'adozione di un metodo piuttosto che un altro:

- L'Elastic Weight Consolidation, metodo di regolarizzazione, ha fatto registrare il tempo di training maggiore e questo è collegato alla complessità computazionale dell'algoritmo; un altro aspetto in grado di penalizzare la scelta di questo algoritmo è la scalabilità: come tutti i metodi di regolarizzazione offre una scarsissima capacità di adattamento ad un numero crescente di task. Se, però, il problema è composto da un numero ridotto di task anche la complessità diminuisce e ciò rende, assieme al non utilizzo di memoria, l'algoritmo competitivo rispetto agli altri.
- L'A-GEM realizzato con il reservoir sampling possiede l'accuracy migliore sul test set, soffrendo però particolarmente sotto l'aspetto del forgetting con il valore più alto tra i 3 candidati finali. Rimane però un algoritmo affidabile e molto conveniente come complessità e tempo di training. Alcune limitazioni possono essere trovare dalla memoria occupata e dalla scalabilità, direttamente dipendente da quest'ultima, anche se non si trattano di vincoli così limitanti nell'applicazione.
- L'Experience Replay, ha riportato le migliori performance nelle metriche specifiche del Continual Learning, senza sfigurare nemmeno nell'accuracy. L'algoritmo è il più semplice tra i tre dal punto di vista della complessità ed anche il più rapido come training. Possiede gli stessi vincoli di memoria e scalabilità dell'A-GEM ma questi non limitano in alcun modo la sua efficacia. La scelta della dimensione di memoria può essere volutamente bassa per evitare di cadere in uno pseudo multi-task learning e quindi forzare l'algoritmo nel mantenere conoscenza dei task passati, conferendo un naturale upper bound a questo aspetto e facilitandone l'applicazione.

Tenuto consto di questi aspetti l'Experience Replay è l'algoritmo che offre il miglior trade-off tra prestazioni, vincoli e complessità ed è stato scelto per verificare come la variazione di iperparametri della rete neurale possa modificare e influenzare la qualità dell'apprendimento. Questi esperimenti saranno trattai nella sezione 5.6

Possiamo concludere l'analisi riportando in figura 5.10 il tempo di training di ogni algoritmo preso in esame in questa tesi, aspetto non di primaria importanza ma utile in determinati contesti, con una rete neurale di tipo MLP. Tutti i task sono stati testati 300 epoche ognuno, per un totale di 1200 epoche per ogni algoritmo.

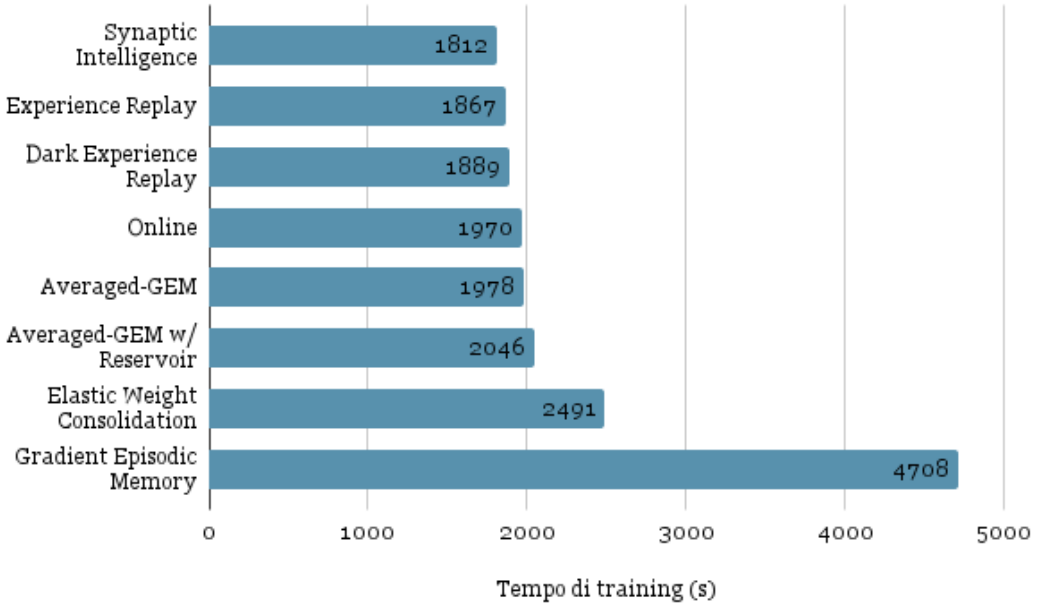



Figura 5.10: Tempo di training con MLP

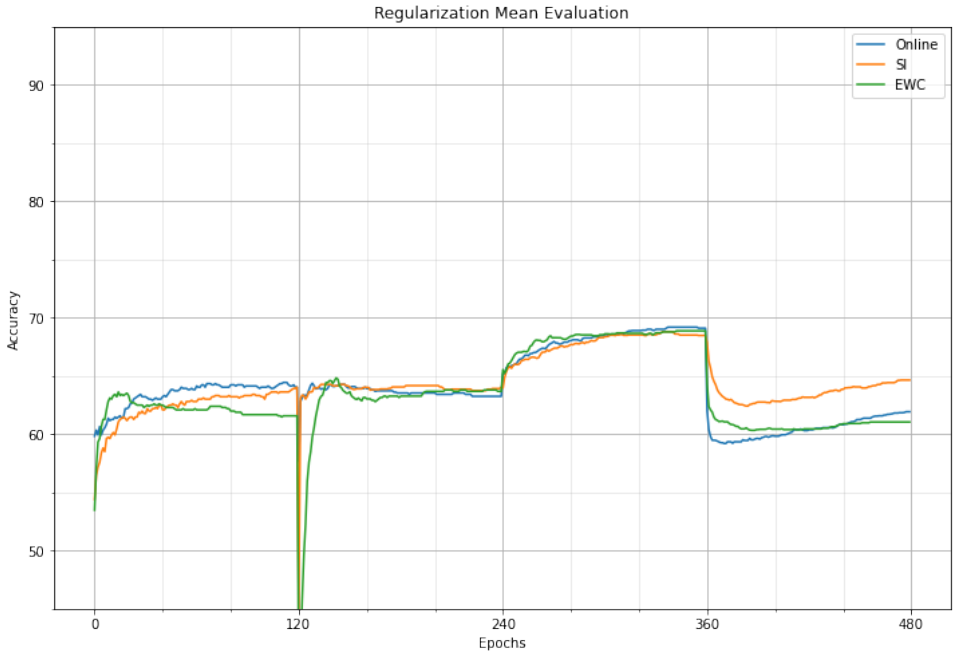
## 5.4 Valutazione algoritmi con CNN

In questa sezione verrà affrontata la stessa analisi effettuata nel paragrafo 5.3, con la differenza che in questo caso la rete neurale utilizzata è una Convolutional Neural Network. L'obiettivo di questa analisi è verificare l'applicabilità e le performance degli algoritmi di Continual Learning usando una rete neurale differente.

Come affermato nel paragrafo 5.2.2, ci si aspettano performance generalmente inferiori rispetto al MLP, dovute principalmente ad una meno approfondita esplorazione delle configurazioni possibili della rete neurale.

### 5.4.1 Metodi di regolarizzazione

Nella figura 5.10 sono riportate le performance dei due metodi di regolarizzazione implementati, ovvero Synaptic Intelligence ed Elastic Weight Consolidation.




| Algoritmo | Accuracy (%) | BWT (%) | FWT (%) | FRG (%) |
|---|---|---|---|---|
| Online | 61,93 | - | - | - |
| SI | 64,64 | -11,36 | 21,88 | 11,36 |
| EWC | 61,10 | -5,51 | 11,59 | 5,51 |

Figura 5.11 e Tabella 5.5: Performance metodi di regolarizzazione con CNN

La prima, immediata, osservazione è il fatto che a differenza del MLP cambia il comportamento dei metodi di regolarizzazione. Con le CNN, EWC performa sensibilmente peggio rispetto a SI e addirittura è inferiore al lower bound imposto dall'Online learning. Anche le metriche di Continual sono peggiori rispetto al MLP per quanto riguarda EWC, mentre SI si mantiene in linea con i risultati ottenuti in precedenza.

Questo diverso comportamento da parte di EWC può essere giustificato dal minor numero di parametri: se nel caso del MLP l' algoritmo congelava un numero di parametri in percentuale poco rilevante, con la riduzione del numero di parametri ottimizzabili e il mantenimento dei parametri rilevanti per un generico task $k$ è naturale che la percentuale di parametri su cui il modello possa apprendere i task successivi sia minore e quindi compromettere la qualità di questo.

Invece, abbastanza sorprendentemente, SI mantiene le performance e ciò è un risultato interessante in quanto tra le due reti neurali testate l'algoritmo non mantiene lo stesso numero di neuroni ma la proporzione rispetto al totale di quelli disponibili, sintomo di grande capacità di adattamento a diverse configurazioni.

### 5.4.2 Metodi di rehearsal

Sono illustrate le performance ottenute dai metodi replay based.

| Algoritmo | Accuracy (%) | BWT (%) | FWT (%) | FRG (%) |
|---|---|---|---|---|
| Online | 61,93 | - | - | - |
| ER | 71,26 | -2,36 | 21,37 | 2,36 |
| Dark ER | 58,07 | -18,16 | 25,79 | 18,16 |

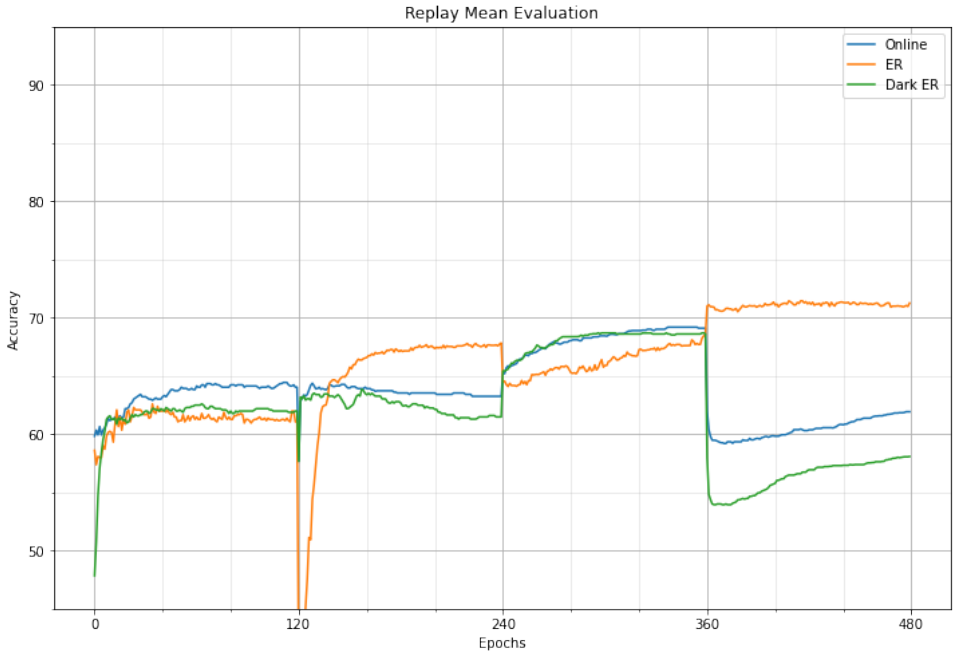



Tabella 5.6 e Figura 5.12: Performance metodi di rehearsal con CNN

A differenza dei metodi di regolarizzazione, il comportamento dei metodi di rehearsal è il medesimo del MLP. Le performance della DER rimangono inferiori rispetto alla soglia dell'Online Learning, mostrando un overfitting più marcato rispetto ai test precedenti e in questo caso la composizione della rete neurale non gioca a favore dell'algoritmo. Durante la valutazione dell'algoritmo il vincolo della loss che caratterizza la DER portava ad aggiornamenti causali i layer di classificazione fully connected della rete, vanificando di fatto gli aggiornamenti dei layer convolutivi.

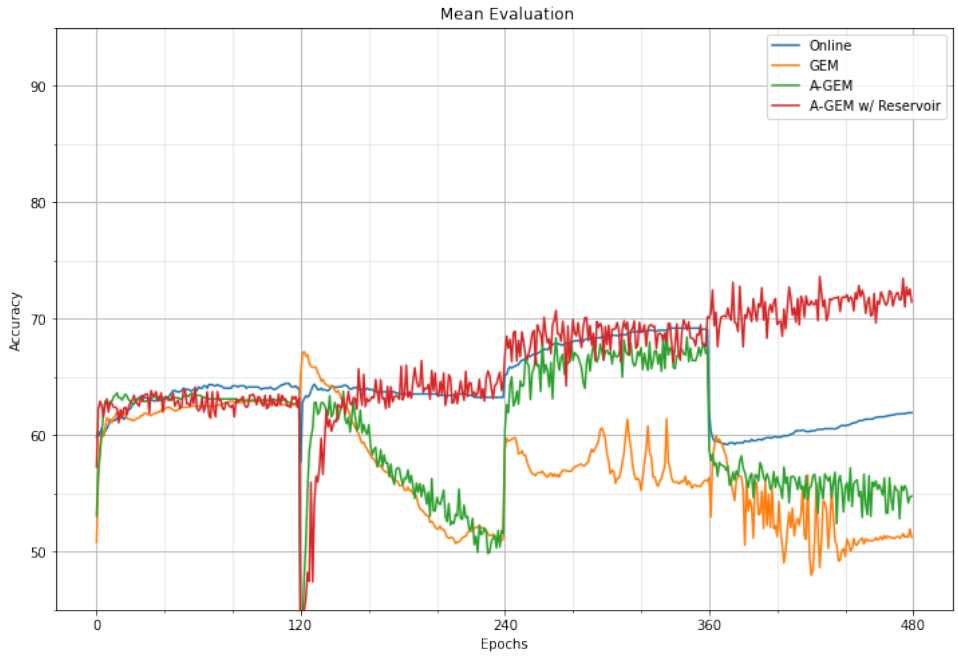
Al contempo Experience Replay si conferma un metodo molto solido e che non soffre il cambio di rete neurale.

### 5.4.3 Metodi ibridi

Prima di effettuare l'analisi è necessaria una doverosa premessa su questi metodi. In condizioni di learning di tipo non continual il comportamento e le modalità di

apprendimento di un MLP e di una CNN sono decisamente diversi, con traiettorie per la discesa del gradiente diverse per uno stesso problema. Siccome questi metodi verificano che la direzione di discesa sia valida sia per il task corrente che per i task precedenti attraverso gli esempi salvati nel buffer e nel caso non siano rispettati i vincoli è necessario risolvere problemi diversi, NP-C con approssimazione del risultato per GEM e il metodo del gradiente proiettato per A-GEM, la nuova direzione di discesa potrebbe stravolgere totalmente il learning e portare a traiettorie non in grado di convergere ad un minimo globale o locale.

Si è verificato proprio questo nel caso delle CNN, con una variazione ed un comportamento non attendibile dell'accuracy media in corrispondenza delle epoche in cui non viene rispettato il vincolo nell'osservazione di diverse batch di esempi. Pertanto, i risultati presentati non sono da ritenersi completamente attendibili per la valutazione degli algoritmi dal punto di vista delle performance, ma possono fornire una chiara visione di quanto questi siano complessi anche con un numero limitato di task.




| Algoritmo | Accuracy (%) | BWT (%) | FWT (%) | FRG (%) |
|---|---|---|---|---|
| Online | 61,93 | - | - | - |
| GEM | 51,47 | -14,65 | -3,71 | 14,65 |
| A-GEM | 54,76 | -17,55 | 21,15 | 17,55 |
| A-GEM w/ Reservoir | 71,44 | -0,99 | 17,49 | 0,99 |

Figura 5.13 e Tabella 5.7: Performance metodi ibridi con CNN

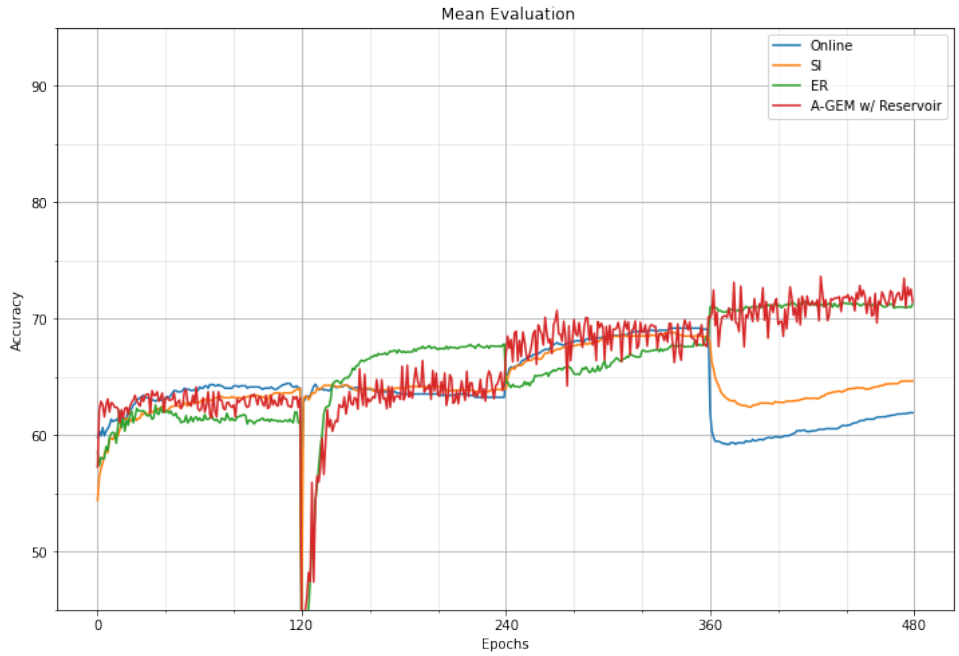
GEM e A-GEM classico mostrano performance ben al di sotto dell'Online Learning, mentre A-GEM con il reservoir sampling è in grado di performare sorprendentemente al livello della MLP. Ciò nonostante, l'andamento assunto dalla media a causa del frequente

non rispetto dei vincoli e variazione della direzione di discesa del gradiente richiede un miglior studio dell'algoritmo alla ricerca di soluzioni in grado di risolvere o quanto meno limitare l'impatto negativo di questo fenomeno. Risulterebbe interessante inoltre osservare nello spazio delle soluzioni la traiettoria descritta dagli algoritmi per valutare se essi convergono verso un minimo, locale o globale, oppure descrivono traiettorie casuali a causa dei frequenti cambi di direzione.

Le metriche di Continual non mostrano, tranne che per A-GEM con il reservoir, prestazioni degne di essere prese in considerazione soprattutto GEM, unico algoritmo tra tutti ad avere un forward transfer negativo.

### 5.4.4 Considerazioni finali

Dopo un'attenta analisi dei principali algoritmi anche con una CNN, è possibile affermare che il problema del Continual Learning applicato a serie storiche finanziarie è risolvibile con successo anche se non tutti i metodi sono applicabili e in grado di offrire le medesime prestazioni. Di seguito è disponibile una panoramica dei migliori metodi:




| Algoritmo | Accuracy (%) | BWT (%) | FWT (%) | FRG (%) |
|---|---|---|---|---|
| Online | 61,93 | - | - | - |
| SI | 64,64 | -11,36 | 21,88 | 11,36 |
| ER | 71,26 | -2,36 | 21,37 | 2,36 |
| A-GEM w/ Reservoir | 71,44 | -0,99 | 17,49 | 0,99 |

Figura 5.14 e Tabella 5.8: Risultati migliori metodi applicati a CNN

Per ogni famiglia di metodi ce n'è uno in grado di primeggiare sugli altri. Utilizzando una CNN come rete di riferimento i metodi di regolarizzazione e ibridi si sono rivelati più efficaci e solidi, nonostante sull' A-GEM con reservoir sampling la valutazione finale non è completa a causa delle osservazioni fatte nel paragrafo precedente.

In questo caso decretare un metodo migliore rispetto agli altri è ben più semplice; se SI mostra performance buone ma non entusiasmanti e A-GEM è da valutare approfonditamente, Experience Replay ha ottenuto risultati di rilievo garantendo stabilità al modello e limitando fortemente il forgetting.

Per concludere sono ripostati i tempi di training di ogni algoritmo con una CNN, da notare che essi sono notevolmente inferiori rispetto al MLP a causa dell'inferiore numero di epoche di training per ogni task e dunque totali ( 480 vs 1200).

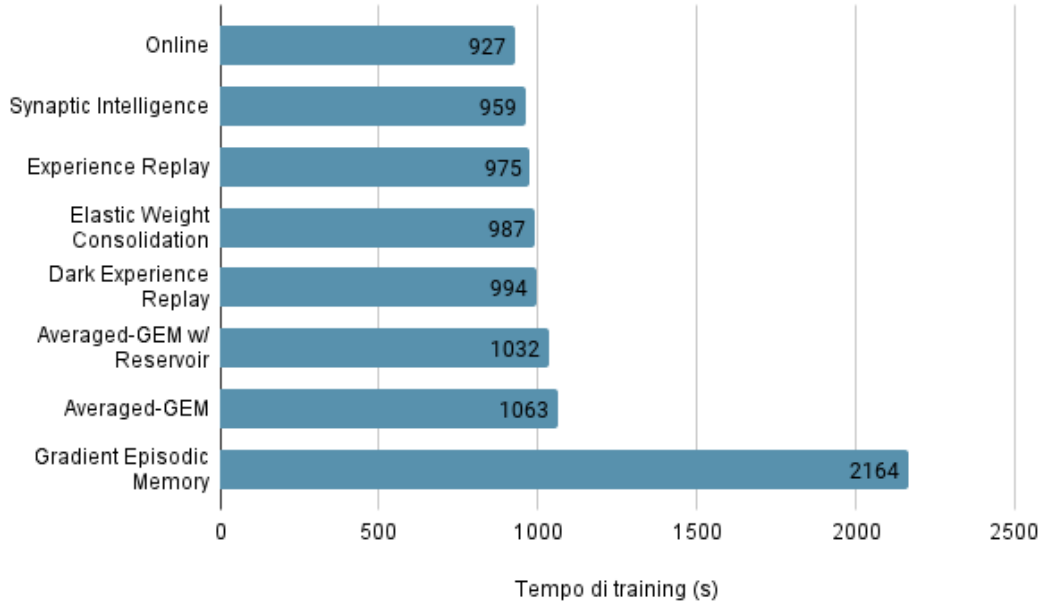



Figura 5.15: Tempo di training con CNN

## 5.5 Confronto tra le configurazioni

Di seguito sono riportate le performance di ogni algoritmo valutato tramite MLP e CNN. Ci si aspettano prestazioni superiori per tutti gli algoritmi nella configurazione con il MLP, come illustrato nei paragrafi precedenti.

| | **MLP** | | | **CNN** | | |
|---|---|---|---|---|---|---|
| | ACC | BWT | FWT | ACC | BWT | FWT |
| **A-GEM** | **65,47** | -5,74 | 25,97 | 54,76 | -17,55 | 21,15 |
| **A-GEM w/ Reservoir** | **72,98** | -9,59 | 25,75 | 71,44 | -0,99 | 17,49 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **DER** | **64,41** | -12,91 | 24,24 | 58,07 | -18,16 | 25,79 |
| **EWC** | **71,64** | -4,15 | 26,02 | 61,10 | -5,51 | 11,69 |
| **ER** | **72,29** | -3,71 | 27,05 | 71,26 | -2,36 | 21,37 |
| **GEM** | **65,25** | -11,29 | 12,21 | 51,47 | -14,65 | -3,71 |
| **SI** | **65,96** | -12,06 | 25,28 | 64,64 | -11,36 | 21,88 |
| **Online** | **65,04** | - | - | 61,93 | - | - |

Tabella 5.9: Riassunto performance

## 5.6 Confronto con altri risultati

La configurazione migliore osservata è il MLP con Experience Replay come algoritmo di Continual. Questa è la configurazione base utilizzata per valutare l'impatto di alcune variazioni effettuate sia sull'algoritmo che su alcuni iperparametri della rete neurale. Le valutazioni sono effettuate sul dataset di test essendo la parte su cui è possibile riscontrare in modo più approfondito i cambiamenti effettuati.

### 5.6.1 Variazione del numero di epoche

Come prima modifica degli iperparametri si tratta l'aumento e la diminuzione del numero di epoche di training per ogni task. Questo è un valore estremamente importante e legato direttamente all'algoritmo di Deep Learning che indica quante volte osservare il training set per migliorare la capacità di predizione. Si è deciso di variare il numero di epoche in 5 valori: 100,200, 300 (come la configurazione finale), 400 e 500 e i risultati sul test set sono interessanti:

| Numero epoche | ACC | BWT | FWT |
|---|---|---|---|
| 100 | 58,55 | -1,01 | 11,56 |
| 200 | 70,25 | -2,96 | 20,06 |
| 300 | 72,29 | -3,72 | 27,05 |
| 400 | 68,04 | -4,77 | 14,90 |
| 500 | 60,21 | -5,91 | 12,53 |

Tabella 5.10: Analisi delle performance al variare del numero di epoche di training

La variazione dell'accuracy in base al numero di epoche indica come la scelta finale di 300 epoche si sia rivelata vincente. Un numero superiore porterebbe sicuramente all'overfitting e conseguente crollo delle performance, mentre un numero inferiore non garantirebbe un learning completo.

Il backward transfer, al contrario, peggiora all'aumentare del numero di epoche. Questo è un comportamento atteso siccome il training su task successivi, anche con tecniche di Continual Learning, porta sempre un backward transfer negativo. Per quanto riguarda il forward transfer, infine, un numero di epoche tra 200 e 300 è ottimale in quanto consente di avere un ottimo valore, ricordando però che essendo ottenuto come differenza tra la valutazione al termine del task con la valutazione pre-training dei vari task e quindi con una configurazione casuale dei parametri del modello, il suo valore non è significante e non deve essere decisivo nella scelta della configurazione migliore.

Per concludere, possiamo dire che un valore tra 200 e 300 epoche potrebbe essere il miglior trade-off tra un'alta accuracy e un miglior backward transfer, tenendo conto anche della diminuzione del tempo di training rispetto alla configurazione finale di questa tesi.

### 5.6.2 Variazione del learning rate

Nella presente sezione si analizza la variazione del learning rate η, parametro che determina l'aggiornamento dei parametri del modello regolando la lunghezza del passo del gradiente. Tipicamente in base al metodo di discesa del gradiente utilizzato (SGD,Momentum ecc.) si hanno alcuni valori tipici. Questo valore viene fatto variare da $10^{-1}$ fino a $10^{-4}$ per un totale di quattro valori. Il learning rate viene valutato in base alla loss tipicamente ,ma in questo problema dove task diversi sperimentato loss diverse, non è possibile e pertanto verrà valutato con l'accuracy dei modelli sul test set.

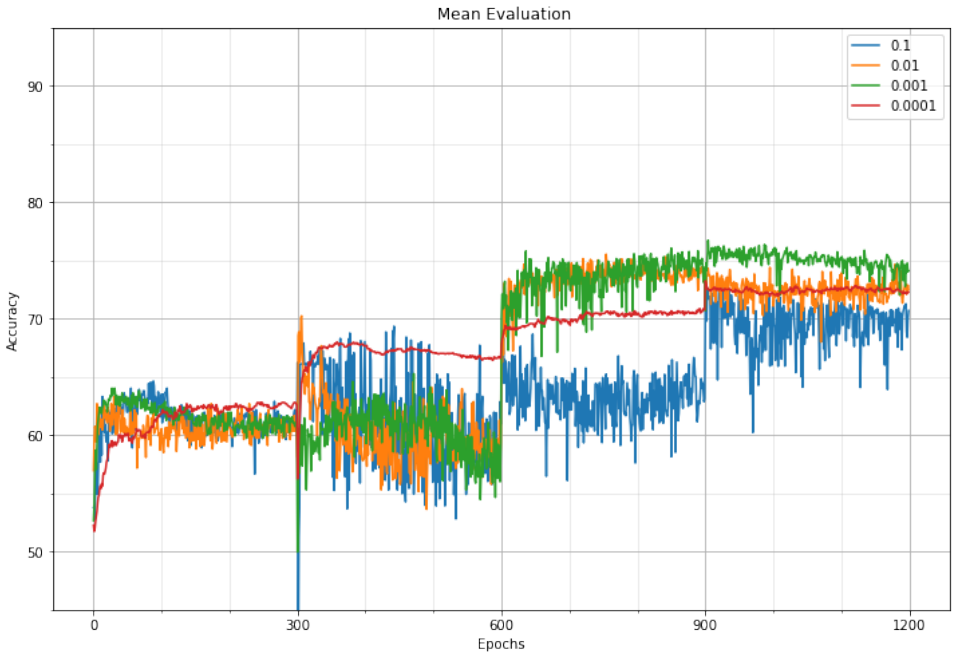



Figura 5.16: Analisi accuracy ER al variare del learning rate

La scelta di questo parametro è delicata e in grado di determinare l'esito dell'apprendimento di tutto il modello. Un learning rate troppo alto porterebbe a superare un minimo o rimbalzarci attorno senza mai convergere. Allo stesso modo un learning rate troppo piccolo non consentirebbe di giungere in un numero accettabile di epoche a convergenza.

Con valori alti del learning rate (0.1 e 0.01) osserviamo una variazione a frequenza altissima dell'accuracy in epoche ravvicinate e questo è il tipico comportamento del rimbalzo appena illustrato.

La scelta ottimale ricade quindi su un valore tra 0.001 e 0.0001. Entrambi sono validi e la decisione finale può dipendere anche dal metodo del gradiente: se il metodo scelto fosse Adam allora il valore maggiore sarebbe preferibile ma siccome per tutte le configurazioni è stato scelto lo SGD con Momentum, il learning rate finale è 0.0001.

### 5.6.3 Variazione del valore di dropout

La regolarizzazione è un fattore necessario e decisivo nelle reti neurali e il MLP adottato in questa tesi non è da meno. Come mostrato nel capitolo 2.2, viene usato il dropout, in grado di spegnere durante il training neuroni diversi in epoche diverse, il tutto in maniera casuale con una percentuale *p*. L'effetto della regolarizzazione si misura, come il learning rate, attraverso la variazione della loss ma come per il parametro precedente ciò non è possibile.

Sono testate tutti i possibili valori percentuali tra 0 e 80%. Un dropout pari allo 0% significa non escludere mai alcun neurone e quindi non applicare regolarizzazione e questo test è stato fatto come metro di paragone. Un valore superiore all' 80% invece porterebbe la rete a non avere un numero sufficiente di neutroni a disposizione per apprendere i task.

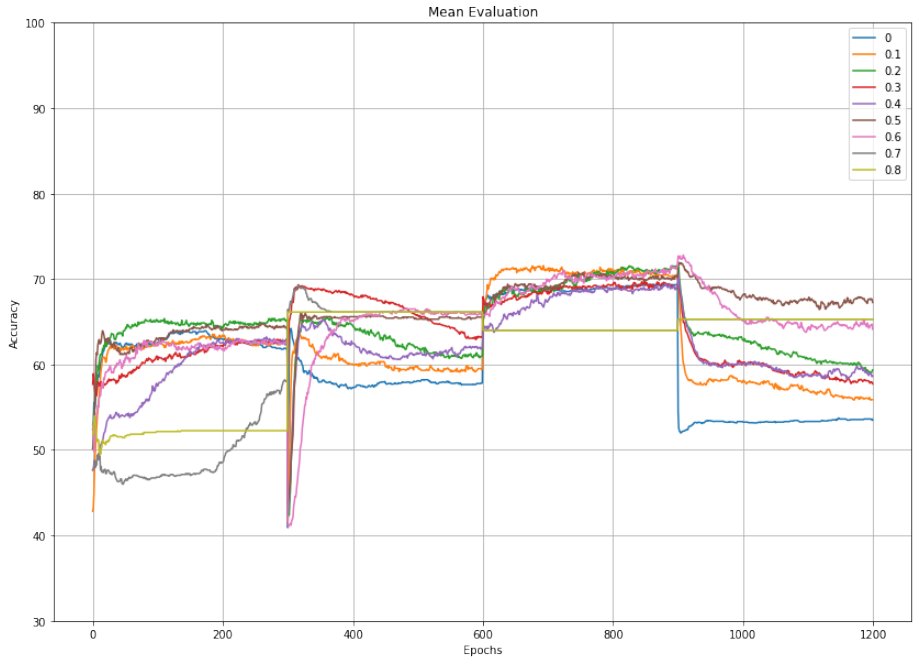



Figura 5.17: Variazione della percentuale *p* di dropout

Il modello senza regolarizzazione sperimenta le performance peggiori, essendo in balia dell'overfitting non ha appreso alcun pattern generale. Aumentando la percentuale di dropout le performance aumentano fino al massimo raggiunto con $p$=0.5, valori superiori escludono troppi neuroni per garantire l'apprendimento. Interessante il caso con $p$=0.8: la curva dell'accuracy è orizzontale per tutti i task e questo può essere giustificato dall'assoluta incapacità del modello di apprendere alcun pattern e quindi di esibire un miglioramento dell'accuracy all'aumentare delle epoche di training.

### 5.6.4 Variazione del buffer di memoria in ER

La dimensione del buffer di memoria nell'Experience Replay, e in generale in tutti i metodi replay based, determina il numero di elementi salvabili dai task passati. Il numero minimo possibile di elementi è pari al numero di task, in modo tale da avere un elemento per task. Non esiste invece una dimensione massima del buffer dal punto di vista teorico, ma avere un buffer grande quanto il dataset completo o poco più piccolo di esso equivale ad allenare tutti i task in parallelo. Tipicamente si sceglie una dimensione del buffer sufficientemente grande da possedere le stesse caratteristiche del dataset originale. Per questo motivo la scelta della dimensione finale per questa tesi nei metodi replay based è stata di 800, a fronte di un dataset totale composto da circa 9000 esempi. Dimensioni maggiori avrebbero sicuramente garantito performance migliori, andando però oltre la soglia auto fissata della dimensione massima del buffer, pari al 10% del dataset originale.

Nonostante nella tabella sottostante non ve ne sia traccia, la letteratura sull'ER dimostra che le performance sulla dimensione del buffer sono limitate, ovvero che oltre una certa soglia la dimensione del buffer diventa ininfluente all'accuracy del modello, ponendo di fatto un *upper bound*. In questo problema non è stato possibile osservarlo a causa dell'impossibilità di svolgere test con un buffer di dimensione maggiore a causa delle dimensioni limitate delle serie temporali usate, comunque tra le più grandi in ambito finanziario.

| Dimensione del buffer | ACC | BWT | FWT |
|---|---|---|---|
| 0 | 65,04 | - | - |
| 100 | 65,89 | -1,46 | 24,13 |
| 200 | 67,49 | -2,48 | 22,73 |
| 500 | 71,48 | -3,45 | 26,55 |
| 800 | 72,29 | - 3,72 | 27,05 |
| 1000 | 74,02 | -1,85 | 27,62 |
| 5120 | 76,63 | -2,01 | 26,88 |

Tabella 5.11: Analisi variazione dimensione buffer

# Capitolo 6

# Conclusioni e sviluppi futuri

In questo capitolo si discutono i risultati ottenuti in base agli obiettivi posti nell'introduzione in termine di vantaggi e svantaggi apportati dalle tecniche di Continual Learning all' interno del mondo finanziario. Si effettua di seguito un'analisi delle possibili tecniche di risoluzione dei limiti e problemi affrontati in questa tesi e i possibili sviluppi futuri in ambito aziendale.

## 6.1 Conclusioni

Durante lo sviluppo di questo progetto di tesi si è analizzata l'efficacia di tecniche di Continual Learning nell'ambito della finanza, in particolare sulla predizione del mercato, in un ambiente noto per la sua incredibile imprevedibilità. La volontà di valutare queste tecniche in un contesto aziendale ha reso necessario riformulare e adattare il problema alle serie temporali in modo tale da poter stabilire se il Continual Learning fosse applicabile in questo contesto, compito non facile in quanto un'errata applicazione in ambito aziendale potrebbe portare alla perdita di molti soldi, non solo all'azienda ma anche a tutti gli investitori coinvolti. Si vedano di seguito gli obiettivi raggiunti durante questo studio:

- Si è studiato ed approfondito il problema del Continual Learning, e le possibili tecniche per realizzarlo in un contesto di classificazione di serie temporali.
- Si è sviluppato un progetto per la risoluzione del problema del mercato finanziario con tecniche di Continual Learning tramite l'utilizzo di librerie e il linguaggio di programmazione Python. Si sono utilizzate le conoscenze aziendali per quanto riguarda i modelli di Deep Learning e la loro costruzione. Questi sono stati estesi in modo tale da poter implementare gli algoritmi di Continual Learning senza stravolgere i modelli a livello di codice. Sono stati infine realizzati ambienti di simulazione del mercato

finanziario in modo da poter valutare l'apprendimento in modo sicuro e controllato nella fasi di training,validation e test.

- È stata effettuata un'approfondita trattazione dei risultati conseguiti, soffermandosi sulle diverse metodologie di Continual Learning e sottolineando la grande instabilità del mercato finanziario. Sono state utilizzate sia le metriche classiche di valutazione dei modelli di Deep Learning sia metriche proprie del Continual Learning.

Le aspettative su questo progetto erano alte e le speranze di ottenere risultati utili alla prosecuzione dello studio non mancavano. Questo progetto però non è altro che un primo studio preliminare per queste nuove metodologie per la predizione del mercato finanziario. Le tecniche di Continual Learning sono svariate e svariate sono le modalità di applicazione, con ampi margini di miglioramento, soprattutto negli ambiti trattati nella sezione successiva.

## 6.2 Sviluppi futuri

Ci sono innumerevoli azioni da svolgere che potrebbero portare un miglioramento delle metodologie trattate e sviluppate fino ad ora. Si offre uno sguardo a quali sono stati i principali problemi riscontrati e quali tecniche potrebbero essere utilizzate al fine di espandere le applicazioni del Continual Learning al mercato finanziario.

### Scelta del dataset

Il dataset preso in esame in questa tesi era composto da una unica serie temporale. Questo è un limite importante in quanto ogni serie temporale ha un andamento ed evoluzione diversa, in particolare per serie temporali che descrivono l'andamento di commodities, come il petrolio grezzo preso in esame in questa tesi. Lo stesso ragionamento è valido per il mercato azionario, anche se spesso serie temporali che descrivono azioni appartenenti allo stesso macro ambito hanno comportamenti simili e quindi sarebbe possibile allenare un unico modello per un gruppo di azioni. Questo è sicuramente un aspetto interessante che richiede uno studio approfondito.

Un altro problema non di poco conto è come costruire il dataset, scegliendo le features più utili per il problema da risolvere. In questa tesi sono state usate features tratte dai più famosi e comuni indicatori finanziari e statistici, però nulla vieta di usarne altre in grado di descrivere determinati fenomeni o specifiche per una tipologia di serie.

Infine, è possibile modificare la granularità dell'osservazione dei dati. In finanza sono disponibili una grandissima quantità di dati e rilevazioni dell'andamento dei prezzi delle azioni o commodities, ed effettuare una analisi su un orizzonte temporale diverso (ore, settimane o mesi) potrebbe portare a risultati diversi.

### Perfezionamento algoritmo changepoint detection

Il primo step del progetto di analisi è determinare i changepoint che andranno a costituire le boundaries tra i task. L'algoritmo di changepoint detection ricopre quindi un grande ruolo nella buona riuscita del progetto. Assodato che gli unici algoritmi applicabili in questo problema sono quelli online, vista l'impossibilità di osservare i dati in un timestep successivo a quello corrente, esistono moltissimi algoritmi in grado di performare un analisi di questo tipo.

Anche il tuning dei parametri dell'algoritmo utilizzato, la BOCD, possono essere perfezionati siccome in questa tesi il compito è stato svolto a mano. Esso può essere automatizzato per trovare la soglia e la finestra di osservazione ottima, in una sorta di apprendimento supervisionato avendo come riferimento eventi significativi in grado di portare ad un cambio di regime.

### Studio altre architetture

In questa tesi è stata proposta un'architettura basata sulle CNN e sono state testate diverse configurazioni per determinare la migliore. È tuttavia possibile una migliore esplorazione dell'architettura alla ricerca di configurazioni in grado di ottenere risultati migliori, visti gli ottimi risultati ottenuti anche con un basso livello di studio. Le CNN e la convoluzione sono strumenti molto potenti che richiedono uno studio approfondito.

Sono tuttavia possibili infinite architetture, da quelle specifiche per le serie temporali ad architetture generali come i MLP e la scelta e valutazione di queste è un compito continuo, grazie anche al contributo della ricerca, sempre in grado di proporre soluzioni innovative.

### Tecniche di Continual Learning

Sono state studiate, implementate e testate solamente le principali metodologie di Continual Learning, osservando risultati incoraggianti. Queste però non sono algoritmi che devono essere rispettati alla lettera ma offrono numerosi spunti per un continuo miglioramento, come dimostrato dalle varianti proposte per il GEM. Di fatto le metodologie potrebbero essere infinite, purtroppo non tutte applicabili. Tra queste è necessario citare le metodologie architetturali, scartate in questa tesi siccome richiedono un identificatore del task corrente. In un problema di *domain-IL*, dove non esistono boundaries tra i task e non viene richiesto al modello di fare inferenza su di esso, è tuttavia possibile costruire dei task descriptor contenenti informazioni di natura statistica, ottenute ad esempio durante la changepoint detection come media o varianza, per identificare e distinguere i task tra loro, permettendo dunque lo studio e la valutazione su questo tipo di tecniche.

# Bibliografia


[1] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115-133.

[2] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, 65*(6), 386–408.

[3] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems, 2*, 303-314.

[4] Ian Goodfellow and Yoshua Bengio and Aaron Courville (2016). Deep Learning. MIT Press. p. 326.

[5] Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*, *113*, 54-71.

[6] McCloskey, Michael; Cohen, Neal J. (1989). *Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem*. Psychology of Learning and Motivation. 24. pp. 109–165.

[7] Mermillod, M., Bugaiska, A., & Bonin, P. (2013). The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in Psychology, 4*.

[8] Ven, G.M., & Tolias, A. (2019). Three scenarios for continual learning. *ArXiv, abs/1904.07734*.

[9] Sutton, R. S., & Barto, A. G. (1998). *Introduction to reinforcement learning* (Vol. 135). Cambridge: MIT press.

[10] Delange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., ... &

Tuytelaars, T. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[11] Rebuffi, S. A., Kolesnikov, A., Sperl, G., & Lampert, C. H. (2017). icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (pp. 2001-2010).

[12] Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T. P., & Wayne, G. (2018). Experience replay for continual learning. *arXiv preprint arXiv:1811.11682*.

[13] Buzzega, P., Boschini, M., Porrello, A., Abati, D., & Calderara, S. (2020). Dark experience for general continual learning: a strong, simple baseline. *arXiv preprint arXiv:2004.07211*.

[14] Shin, H., Lee, J. K., Kim, J., & Kim, J. (2017). Continual learning with deep generative replay. *arXiv preprint arXiv:1705.08690*.

[15] Aljundi, R., Lin, M., Goujaud, B., & Bengio, Y. (2019). Gradient based sample selection for online continual learning. arXiv preprint arXiv:1903.08671.

[16] Lopez-Paz, D., & Ranzato, M. A. (2017). Gradient episodic memory for continual learning. *Advances in neural information processing systems*, *30*, 6467-6476.

[17] Chaudhry, A., Ranzato, M. A., Rohrbach, M., & Elhoseiny, M. (2018). Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*.

[18] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A., ... & Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, *114*(13), 3521-3526.

[19] Schwarz, J., Czarnecki, W., Luketina, J., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., & Hadsell, R. (2018, July). Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning* (pp. 4528-4537). PMLR.

[20] Zenke, F., Poole, B., & Ganguli, S. (2017, July). Continual learning through synaptic intelligence. In *International Conference on Machine Learning* (pp. 3987-3995). PMLR.

[21] Li, Z., & Hoiem, D. (2017). Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, *40*(12), 2935-2947.

[22] Sun, X., Panda, R., Feris, R., & Saenko, K. (2019). Adashare: Learning what to share for efficient deep multi-task learning. *arXiv preprint arXiv:1911.12423*.

[23] Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., ... & Hadsell, R. (2016). Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

[24] Mallya, A., & Lazebnik, S. (2018). Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (pp. 7765-7773).

[25] Aminikhanghahi, S., & Cook, D. J. (2017). A survey of methods for time series change point detection. *Knowledge and information systems*, *51*(2), 339-367.

[26] Facebook AI Research lab. *PyTorch*. https://github.com/pytorch/pytorch. 2016

[27] Weights & Biases. https://docs.wandb.ai/ . 2017

[28] TA-Lib. https://www.ta-lib.org/

[29] Adams, R. P., & MacKay, D. J. (2007). Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*.

[30] Tealab, A. (2018). Time series forecasting using artificial neural networks methodologies: A systematic review. *Future Computing and Informatics Journal*, *3*(2), 334-340.

[31] Schlaifer, R., & Raiffa, H. (1961). Applied statistical decision theory.

[32] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533-536.

[33] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[34] Buzzega, P., Boschini, M., Porrello, A., & Calderara, S. (2021, January). Rethinking Experience Replay: a Bag of Tricks for Continual Learning. In 2020 25th International Conference on Pattern Recognition (ICPR) (pp. 2180-2187). IEEE.

[35] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Dark knowledge. Presented as the keynote in BayLearn, 2, 2014

[36] Oil prices keep plummeting as OPEC starts a price war with the US. https://www.vox.com/2014/11/28/7302827/oil-prices-opec