**LISTA PAPER (FAKE NEWS DETECTION + CONTINUAL LEARNING**

PAPER 36

**Continual Learning for Large Language Models: A Survey**

* Tongtong Wu, Linhao Luo, Yuan-Fang Li, Shirui Pan, Thuy-Trang Vu, Gholamreza Haffari
* 2024
* <https://arxiv.org/abs/2402.01364>

***METODO***

Per applicare continual learning (CL) ai Large Language Models (LLMs), l’idea è distinguere il continual learning negli LLMs rispetto ai modelli più piccoli, introducendo una **struttura a più stadi**:

* **Continual Pre-training (CPT)**: aggiornamento delle basi linguistiche e delle conoscenze di base del modello, senza riaddestrarlo da zero:
  + **Aggiornamento di fatti**: LLM devono integrare continuamente informazioni recenti (nuovi eventi, dati, ecc.). Le tecniche principali sono:
    - **Dataset dinamici**: aggiornati continuamente da fonti come news, Wikipedia, articoli scientifici, social media
    - **Apprendimento incrementale dei fatti**: si addestrano i modelli su snapshot temporali di Wikipedia o Web data e si riduce il catastrophic forgetting combinando nuova conoscenza con quella già appresa;
  + **Aggiornamento di domini**: specializzazione in ambiti (medicina, finanza, e-commerce). Ci sono due modalità principali:
    - **Domain-incremental pre-training**: il modello accumula conoscenza da più domini progressivamente
    - **Domain-specific continual pre-training**: trasformare un LLM generale in uno specializzato.
  + **Espansione linguistica**: nuove lingue e dialetti, ma anche linguaggi di programmazione.
* **Continual Instruction Tuning (CIT)**: miglioramento della capacità di seguire istruzioni. Dopo il pre-training, il modello viene continuamente raffinato per seguire istruzioni o risolvere compiti specifici (in formato “instruction–output”), ed evitare il catastrophic forgetting dei compiti già appresi.
  + **Task-incremental CIT (nuovi compiti)**: aggiornare il modello con nuove task instructions.
    - Il problema principale è il catastrophic forgetting, e ci sono varie soluzioni come replay (memorizzare vecchi dati), regularization, parameter-efficient tuning (LoRA, adapters).
    - CIT su LLMs è difficile per costi computazionali; le tecniche PET sono quindi essenziali per ridurli.
  + **Domain-incremental CIT (nuovi domini)**: aggiornare le capacità istruttive in domini nuovi.
    - La sequenza di tuning su diversi domini influenza fortemente la qualità finale; l’ordine di addestramento conta.
  + **Tool-incremental CIT**: permette agli LLM di imparare continuamente nuovi strumenti esterni (API, motori di ricerca, calcolatrici, ecc.).
* **Continual Alignment (CA)**: allineamento continuo a valori, norme etiche e preferenze umane
  + **Value Alignment**: aggiornare norme etiche e culturali
  + **Preference Alignment**: adattare le risposte a gruppi demografici diversi o preferenze individuali.

Riguardo ai **benchmark** e le **metriche** usati nei vari studi.

* **Benchmark**: i principali benchmark usati nei lavori di continual learning applicati agli LLMs, organizzandoli secondo le tre fasi principali:
  + **CPT**: TemporalWiki, Firehose, CKL, TRACE.
  + **CIT**: Continual Instruction Tuning Benchmark (CITB), ConTinTin.
  + **CA**: dataset di preferenze umane come SHP e HH.
* **Metriche**: il paper distingue due livelli di valutazione:
  + **Target task performance**: misurare la performance “classica” del continual learning (quanto bene il modello impara sequenzialmente senza dimenticare)
    - Forward Transfer (FWT): quanto le conoscenze vecchie aiutano nei nuovi task.
    - Backward Transfer (BWT): misura del forgetting.
    - Average Accuracy.
    - **Forward Transfer (FWT)**: Misura se ciò che il modello ha imparato prima aiuta a imparare più velocemente nuovi task.
    - **Backward Transfer (BWT)**: Indica quanto il modello dimentica i task precedenti.
    - **Average Performance (Avg. ACC)**: Media generale delle performance su tutti i task dopo l’ultimo addestramento.
  + **Cross-stage forgetting**: misurare quanto il modello “dimentica” tra fasi diverse
    - **General Ability Delta (GAD)**: Misura la variazione nella performance su compiti generali dopo CL (quanto il modello “capisce” ancora il linguaggio generale).
    - **Instruction Following Delta (IFD)**: Misura la perdita di capacità di seguire istruzioni dopo CL.
    - **Safety Delta (SD)**: Valuta quanto la sicurezza del modello (evitare output dannosi) cambia dopo CL.

PAPER 37

**Continual Learning of Large Language Models: A Comprehensive Survey**

* Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, Zifeng Wang, Sayna Ebrahimi, Hao Wang
* 2025
* <https://dl.acm.org/doi/10.1145/3735633>

***METODO***

Le **principali famiglie di LLM** che sono stati studiati in scenari di apprendimento continuo:

* **Decoder-only (auto-regressive)**: GPT, LLaMA, Gemini, Pythia, GPT-NeoX, OPT, GLaM
* **Encoder-only / encoder-decoder**: BERT, RoBERTa, T5, ERNIE, DistilBERT
* **Multimodali**: CLIP, MLLMs (multimodal LLMs)

Il paper organizza le **tecniche di CL in 5 categorie principali**:

* **Replay-based**: Mantengono un buffer di esempi passati e lo riutilizzano durante l’addestramento di nuovi task.
* **Regularization-based**: Aggiungono un termine di regolarizzazione per penalizzare cambi drastici nei parametri rispetto alle versioni precedenti.
* **Architecture-based**: Estendono la rete dinamicamente aggiungendo moduli o parametri.
* **Optimization-based**: Modificano le strategie di training (es. RecAdam, mix-review). Consentono di bilanciare stabilità e plasticità.
* **Representation-based**: Lavorano sullo spazio delle rappresentazioni invece che sui parametri, mantenendo embedding stabili tra task.

Il paper distingue **tipi di apprendimento incrementale**:

* **TIL (Task-Incremental Learning)**: il modello sa quale task sta affrontando.
* **DIL (Domain-Incremental Learning)**: stessi formati input-output, ma domini diversi.
* **CIL (Class-Incremental Learning)**: nuovi concetti/classe da apprendere.

Introduce anche una distinzione originale:

* **Vertical continuity**: adattamento da dominio generale a domini specifici (es. LLM generalista → LLM medico).
* **Horizontal continuity**: aggiornamenti nel tempo o su domini diversi (es. GPT aggiornato con notizie 2021 → 2022).

PAPER 38

**Continual Pre-training of Language Models**

* Zixuan Ke, Yijia Shao, Haowei Lin, Tatsuya Konishi, Gyuhak Kim, Bing Liu
* 2023
* <https://arxiv.org/abs/2302.03241>

***METODO***

Il lavoro affronta il **continual domain-adaptive pre-training (continual DAP-training)**. Partendo da un LM pre-addestrato (es. RoBERTa), si vuole adattarlo progressivamente a una sequenza di domini differenti (Ristoranti, ACL papers, AI, Phone, PubMed, Camera) senza dimenticare le conoscenze pregresse (catastrophic forgetting) e favorendo knowledge transfer.

Il modello proposto è **DAS (Domain-Adaptive pre-training with Soft-masking)**, un metodo che aggiorna direttamente il modello pre-addestrato invece di usare moduli aggiuntivi (come adapters o prompt-tuning).

* **Soft-masking**: Ogni unità del modello (neuroni e attention heads) riceve un punteggio di importanza ∈ [0,1]. Questo punteggio controlla quanto un’unità può essere aggiornata durante l’apprendimento del nuovo dominio.
  + Nella fase forward, il soft-masking non viene applicato. Il modello sfrutta tutta la conoscenza accumulata.
  + Nella fase backward, il gradiente viene modulato. Si preservano le unità importanti per conoscenza generale e domini passati
* **Proxy per la conoscenza generale**: Poiché i dati di pre-training iniziale non sono disponibili, viene introdotto un proxy basato su robustezza e KL-divergence per stimare l’importanza delle unità alla conoscenza generale
* **Contrastive Learning per integrazione delle conoscenze**: DAS usa una perdita contrastiva che confronta le rappresentazioni del modello pre-addestrato + domini passati e le rappresentazioni del modello con dominio corrente incluso. L’obiettivo è di spingere il modello ad apprendere rappresentazioni complementari, integrando nuove informazioni senza separarle dalle vecchie

***RISULTATO***

Il paper confronta DAS con approcci classici di CL:

* Regularization-based: EWC (Elastic Weight Consolidation)
* Replay-based: DER++ (memorizzazione di esempi precedenti)
* Parameter-isolation: HAT (Hard Attention to Task), DEMIX, Adapters, Prompt-based
* Knowledge Distillation (KD)

Questi metodi soffrono di compromessi (o preservano conoscenza ma sacrificano performance, o trasferiscono conoscenza ma dimenticano). Invece, **DAS supera queste limitazioni** con soft-masking e contrastive integration.

L’ablation study mostra che ogni componente è cruciale:

* Rimuovere il soft-masking → degrado marcato delle performance.
* Rimuovere la contrastive loss → peggiora la rappresentazione.
* Rimuovere l’inizializzazione con proxy → perdita di conoscenza generale.

PAPER 39

**Continual Pre-Training of Large Language Models: How to (re)warm your model?**

* Kshitij Gupta, Benjamin Thérien, Adam Ibrahim, Mats L. Richter, Quentin Anthony, Eugene Belilovsky, Irina Rish, Timothée Lesort
* 2023
* <https://arxiv.org/abs/2308.04014>

***METODO***

Il lavoro affronta il problema del **continual pre-training** nei **modelli di linguaggio di grandi dimensioni** (LLM). Il paper esplora strategie di warm-up del learning rate come metodo semplice ed efficiente per rendere stabile e performante l’addestramento continuo.

* Il **modello** usato per lo studio è **Pythia 410M** (famiglia GPT-NeoX). Scelta per la scalabilità e perché è un modello open-source di media dimensione.
* I dataset usati sono:
  + Upstream (pre-training originale): Pile (300B tokens).
  + Downstream (continual pre-training): SlimPajama (297B tokens), dataset deduplicato e più esteso rispetto a RedPajama.

Le **tecniche di Continual Learning** considerate non hanno il focus su replay o regolarizzazione (troppo costosi per LLM), ma su **strategie di warm-up** **del learning rate**:

* **Warm-up length**
  + Hanno testato warm-up con 0%, 0.5%, 1%, 2% dei dati
  + Il risultato è che la lunghezza del warm-up non cambia molto le performance.
  + Se non si fa warm-up, c’è un “chaotic phase / stability gap”: un picco iniziale della perdita che però non ha conseguenze a lungo termine.
* **Maximum Learning Rate (MaxLR)**
  + Hanno testato diversi valori: 1.5e-4, 3e-4, 6e-4.
  + Il risultato è che
    - **MaxLR alto** porta a migliore adattamento al nuovo dataset (SlimPajama) ma più forgetting su Pile.
    - **MaxLR basso** preserva di più la conoscenza precedente (Pile) ma adatta peggio al nuovo dataset.
  + Esiste un trade-off tra stabilità e plasticità.
* **Confronto con training da zero**
  + I modelli continuati con warm-up ottengono migliori performance rispetto a quelli ri-addestrati da zero, anche quando il dataset downstream è grande quanto quello upstream.
* **Re-warming sullo stesso dataset**
  + Se si continua l’addestramento su Pile (invece che su SlimPajama), l’aumento della loss iniziale dovuto al re-warm rimane.
  + Quindi non è solo colpa del “distribution shift” tra dataset, ma anche delle dinamiche di ottimizzazione legate al ri-aumento del learning rate.
* **Checkpoint selection**
  + Testati checkpoint a diverse fasi dell’addestramento sul Pile (iniziale, intermedio, finale).
  + Il risultato è che partire dall’ultimo checkpoint (quello convergente) è meglio. Gli early checkpoint non migliorano l’adattamento e portano a performance peggiori.

***RISULTATI***

Risultati principali sono che:

* **Warm-up length non è cruciale**: la durata del warm-up non cambia molto le performance.
* **MaxLR è il fattore chiave**: più è alto e migliore è l’adattamento al nuovo dataset, ma con maggiore forgetting sul vecchio.
* **Continual pre-training > training da zero**: conviene sempre continuare da un checkpoint piuttosto che ri-addestrare da capo.
* **Problema del "chaotic phase"**: se si parte subito con MaxLR alto, c’è un picco iniziale della perdita, ma non danneggia i risultati finali.
* **Checkpoint finale migliore**: partire da un modello già convergente è la scelta più efficiente.

PAPER 40

**Memory Efficient Continual Learning with Transformers**

* Beyza Ermis, Giovanni Zappella, Martin Wistuba, Aditya Rawal, Cedric Archambeau
* 2022
* <https://arxiv.org/abs/2203.04640>

***METODO***

Il lavoro affronta il problema del Continual Learning (CL) con Transformers, in particolare come gestire nuove classi/task senza dimenticare quelle già apprese (catastrophic forgetting) e senza far esplodere l’uso di memoria.

La proposta è **ADA (Adaptive Distillation of Adapters)**, una tecnica che combina **adapters** e **knowledge distillation** per mantenere le prestazioni con un budget di memoria fisso

* **Base model**: un Transformer pre-addestrato (BERT, DistilBERT, RoBERTa per NLP; ViT e DeiT per visione) viene mantenuto congelato (parametri fissi).
  + ADA parte da un Transformer pre-addestrato (come BERT, RoBERTa per NLP o ViT, DeiT per visione).
  + I parametri principali del Transformer (Θ) vengono congelati, cioè non vengono aggiornati durante l’addestramento sui nuovi task.
  + Questo permette di evitare di riscrivere le rappresentazioni di base, prevenendo il catastrophic forgetting.
  + Inoltre, il Transformer pre-addestrato contiene già rappresentazioni linguistiche o visive molto generali, quindi basta “specializzarle” tramite piccoli moduli aggiuntivi.
* **Adapters**: piccoli moduli a bottleneck inseriti tra i layer del Transformer. Ogni nuovo task ottiene un proprio Adapter e un piccolo head di classificazione.
  + Un Adapter è un piccolo modulo neurale inserito tra i layer del Transformer.
  + Ogni nuovo task ottiene un proprio Adapter e una piccola testa di classificazione (head)
  + Durante l’addestramento sul task, i parametri del Transformer restano fissi, e si aggiornano solo l’adapter e l’head.
  + Questo riduce drasticamente i parametri addestrabili
  + Gli Adapters permettono di specializzare il Transformer su un nuovo task senza intaccare i parametri globali, mantenendo così la conoscenza dei task precedenti.
* **Pool di Adapters**: invece di accumularne uno per ogni task (come in AdapterFusion), ADA mantiene un numero fisso di Adapters K in memoria.
  + Nelle soluzioni precedenti come AdapterFusion, si tiene un Adapter per ogni task.
  + La soluzione di ADA è di mantenere un pool limitato di K Adapters.
  + Ogni Adapter nel pool rappresenta un insieme “consolidato” di più task precedenti.
  + Quando arriva un nuovo task, se c’è **ancora spazio** (meno di K Adapters nel pool), si aggiunge il nuovo Adapter, mentre se il **pool è pieno**, entra in gioco il consolidamento (spiegato sotto).
* **Consolidamento**: quando arriva un nuovo task, l’idea è di fondere un Adapter già presente nel pool con quello appena addestrato, così da mantenere un numero fisso di moduli.
  + **Addestramento del nuovo task**: Si addestra un nuovo Adapter e la head sul nuovo task usando i dati etichettati, mentre il Transformer rimane fisso.
  + **Selezione dell’Adapter da fondere**: ADA calcola per ciascun Adapter nel pool un punteggio di trasferibilità rispetto al nuovo task. L’Adapter con il punteggio più alto (cioè quello che “condivide” più conoscenza utile col nuovo task) viene scelto come partner di fusione.
  + **Distillazione (fusione)**: Si crea un nuovo Adapter consolidato. Per farlo, si usa una double distillation loss che misura la la perdita misura la differenza tra le predizioni dei due modelli, ossia tra l’output combinato del vecchio modello (insegnante) e l’output del nuovo modello (studente consolidato)

Le **tecniche di Continual Learning** usate stanno nella gestione degli Adapters:

* **Knowledge Distillation (Double Distillation Loss)**
  + I logit (soft targets) del nuovo modello e di quello selezionato vengono usati per addestrare un Adapter “consolidato”
  + In questo modo il nuovo Adapter impara sia dal task nuovo sia da quello vecchio.
* **Adapter Selection tramite Transferability Scores**
  + Per decidere quale Adapter sostituire, ADA stima quanto le sue rappresentazioni siano trasferibili al nuovo task.
  + Due metriche usate:
    - **TransRate**: misura la mutua informazione tra features dell’Adapter e i label del nuovo task.
    - **LEEP** (Log Expected Empirical Prediction): misura la bontà di un modello pre-addestrato sul nuovo task tramite distribuzioni condizionali.
  + L’Adapter con il punteggio più alto viene consolidato insieme al nuovo.
* **Pool Management (K controllato)**
  + Mantiene la memoria limitata. Se il numero di task cresce, ADA continua a consolidare senza accumulare parametri in modo lineare.

***RISULTATI***

Come **baselines** vengono considerate:

* Fine-tuning completo
* Solo head
* Adapters semplici
* AdapterFusion
* Experience Replay (ER)
* Elastic Weight Consolidation (EWC, per visione)

E come **risultati principali** è stato ottenuto

* **Prestazioni predittive**: ADA con TransRate/LEEP e **K=4** è comparabile ad AdapterFusion, ma usa molti meno parametri. Invece, ADA con **K=1** (distillazione pura) è competitivo nelle prime fasi ma poi degrada, mostrando che il consolidamento selettivo è cruciale.
* **Efficienza di memoria**: AdapterFusion richiede un Adapter per ogni task (es. 60 su Wiki-30K). Invece, ADA mantiene solo K=4–8 Adapters, risparmiando fino a un ordine di grandezza di parametri, pur mantenendo l’accuratezza.
* **Backward e Forward Transfer (BWT/FWT)**: ADA minimizza il negative backward transfer (non dimentica i task vecchi) e mostra forward transfer positivo (usa conoscenze passate per aiutare nuovi task).
* **Ablation Studies**:
  + Selezionare casualmente gli Adapters porta a risultati molto peggiori, e quindi la scelta tramite TransRate/LEEP è fondamentale.
  + TransRate tende a superare LEEP come metrica di selezione
  + L’aumento di K porta benefici decrescenti

PAPER 41

**Adapting BERT for Continual Learning of a Sequence of Aspect Sentiment Classification Tasks**

* Zixuan Ke, Hu Xu, Bing Liu
* 2021
* <https://arxiv.org/abs/2112.03271>

***METODO***

Il lavoro affronta il problema del **Continual Learning (CL)** per il compito di Aspect Sentiment Classification (ASC), cioè classificare la polarità (positiva, negativa o neutrale) di una frase rispetto a un certo aspetto.

Il paper confronta il modello proposto con 18 baseline, divise in:

* **Non-continual learning (NL)**: addestra un modello separato per ogni task
  + **BERT** fine-tuned
  + **Adapter-BERT** (Houlsby et al., 2019): piccoli moduli “adapter” inseriti in BERT, allenati senza modificare i parametri principali → meno rischio di forgetting.
  + **W2V** con embeddings word2vec.
* **Continual learning (CL):**
  + **WDF (Without Dealing with Forgetting)**: BERT, Adapter-BERT, W2V addestrati sequenzialmente senza meccanismi anti-forgetting.
  + **CL state-of-the-art methods** adattati dal computer vision o dal sentiment classification document-level:
    - **EWC (Elastic Weight Consolidation)**: regolarizzazione per preservare parametri importanti
    - **UCL (Uncertainty-based CL)**
    - **OWM (Orthogonal Weight Modification)**
    - **HAT (Hard Attention to the Task)**: mascheramento di neuroni rilevanti
    - **KAN** e **SRK**: progettati per sentiment classification a livello documento

Il modello proposto è **B-CL (BERT-based Continual Learning)**. Il contributo principale è l’integrazione di **capsule networks** e **adapter layers** in BERT.B-CL **sostituisce gli adapter di Adapter-BERT** con un nuovo modulo chiamato Continual Learning Adapter (CLA), che ha due componenti principali:

* **Knowledge Sharing Module (KSM)**: facilita condivisione di conoscenza. Il KSM serve a individuare quali task passati sono simili al nuovo task e a condividere la conoscenza utile tra di essi. Questo è ciò che permette al modello di **ottenere forward e backward transfer**.
  + KSM è basato su una variante del **Capsule Network**, che usa dynamic routing per decidere quanto una rappresentazione (capsula) di un task contribuisce a un’altra.
  + Ha due livelli di capsule:
    - **Task Capsule Layer (TCL)**: Ogni capsula di questo livello rappresenta un task appreso finora. Quando arriva un nuovo task, si aggiunge una nuova capsula dedicata a quel task. Ogni capsula è un piccolo MLP che trasforma le hidden states di BERT in una rappresentazione vettoriale
    - **Knowledge Sharing Capsule Layer (KCL)**: Qui le capsule rappresentano conoscenza condivisa tra più task. Ogni capsule di TCL (cioè ogni task) invia i propri vettori verso le capsule di KCL, ma con pesi di connessione dinamici, determinati da quanto due task sono simili.
      * Se due capsule (task) producono rappresentazioni molto “simili” (alto dot product), la loro connessione si rinforza.
      * Si forma così un cluster di task simili, che condividono conoscenza attraverso la capsula di conoscenza comune.
* **Task Specific Module (TSM)**: previene catastrophic forgetting. Il TSM serve a preservare le parti del modello specifiche di ciascun task, in modo che, quando si addestra un nuovo task, il modello non “dimentichi” quelli passati. Questa parte affronta direttamente il problema del **catastrophic forgetting**.
  + Utilizza il meccanismo **task masks** (ispirate a HAT) per proteggere i neuroni specifici di ciascun task già imparato.
    - Per ogni task e per ogni layer del TSM viene appresa una maschera neurale. È una soft mask (valori tra 0 e 1) che controlla quali neuroni sono attivi per un determinato task.
  + Durante il training di un nuovo task, i neuroni che in passato erano importanti per altri task (maschere con valore 1) vengono bloccati (gradienti azzerati). Invece, i neuroni “liberi” possono invece essere usati per apprendere il nuovo task.
  + In questo modo, durante l’addestramento su un nuovo task, i parametri cruciali per i task passati non vengono sovrascritti.
  + Le maschere sono soft (differenziabili) e permettono anche neuroni condivisi quando serve.

***RISULTATI***

B-CL supera tutte le baseline, con:

* Accuratezza media = 0.8829
* Macro-F1 = 0.8140

**Forward transfer**: già quando un task viene appreso per la prima volta, B-CL trasferisce conoscenza dai task precedenti, migliorando le performance.

**Backward transfer**: i task passati migliorano grazie all’apprendimento di nuovi task (grazie alla condivisione di capsule).

Per l’ablation study:

* Rimuovendo KSM (solo task masks): performance cala (MF1 ~0.71).
* Rimuovendo TSM (solo capsule sharing): performance cala (MF1 ~0.78).
* Senza entrambi (Adapter-BERT puro): performance molto bassa (~0.45 MF1).

PAPER 42

**BERT WEAVER: Using WEight AVERaging to enable lifelong learning for transformer-based models in biomedical semantic search engines**

* Lisa Kühnel, Alexander Schulz, Barbara Hammer, Juliane Fluck
* 2022
* <https://arxiv.org/abs/2202.10101>

***METODO***

Il paper tratta la capacità di continual/lifelong learning, cioè aggiornare i modelli con nuovi dati senza doverli riaddestrare da zero e senza “dimenticare” quanto già appreso (catastrophic forgetting).

Partendo da un modello di base BioBERT (dmis-lab/biobert-base-cased-v1.1), il nuovo metodo proposto è **WEAVER (WEight Averaging for lifElong leaRning)**

* L’idea nasce da un principio già noto nel federated learning, chiamato Federated Averaging (FedAvg). In FedAvg, più modelli vengono addestrati in parallelo su dati diversi (distribuiti tra “client”), e poi i loro pesi vengono aggregati (mediati) su un server centrale. Questo permette di ottenere un modello “globale” che rappresenta la conoscenza di tutti i nodi, senza mai condividere i dati grezzi.
* WEAVER riprende lo stesso principio, ma lo applica a un contesto sequenziale (continual learning) invece che parallelo (federated learning).
  + **Addestramento iniziale**
    - Considerando una sequenza di dataset per lo stesso task, si parte da un modello pre-addestrato, tipicamente BioBERT.
    - Si fine-tuna sul primo dataset e otteniamo un primo modello con i rispettivi pesi
  + **Nuovo dataset**
    - Quando arriva un nuovo dataset, invece di ricominciare da zero, si prende il modello precedente e lo si fine-tuna sul nuovo dataset, ottenendo il secondo modello con nuovi pesi
    - Questo step incorpora le nuove conoscenze, ma può “dimenticare” le vecchie (catastrophic forgetting).
  + **Media pesata dei pesi**
    - WEAVER aggiunge ora un passaggio di post-processing: la weight averaging.
    - Si calcola una media pesata tra i primi pesi e i secondi pesi. In questo modo, se un dataset è molto grande, i suoi pesi avranno maggiore influenza sul modello finale.
  + **Passaggi successivi**: si continua allo stesso modo
* Questa media funziona perché l’idea è che i pesi del modello rappresentano lo “spazio delle conoscenze” acquisite. Facendo una media pesata, si “infonde” nel nuovo modello parte della conoscenza del modello precedente, senza bisogno di rieseguire il training sugli stessi dati.

Tecniche di **continual learning** considerate:

* **Fine-tuning sequenziale** (baseline): semplice ri-addestramento su nuovi dati. Causa forte forgetting.
* **Regularization-based**: Elastic Weight Consolidation (EWC), che penalizza modifiche ai pesi “importanti” per i task precedenti.
* **Architecture-based**: AdapterFusion, ossia piccoli moduli aggiunti al modello, addestrati su ogni dataset e poi fusi.
* **Rehearsal-based** (upper bound): Replay buffer di una parte dei dati precedenti assieme ai nuovi.
* **Multi-task Learning** (MTL, upper bound): Addestramento congiunto su tutti i dataset contemporaneamente (impraticabile in molti scenari reali, ma usato come benchmark massimo).

***RISULTATI***

Analizzando le baseline su singolo dataset, ogni modello rende bene solo sul proprio dataset (es. BioBERT su NCBI → F1 ≈ 83%, ma cala a ≈ 65% su altri dataset).

Considerando il **confronto tra WEAVER e altri metodi**:

* Su tutte le entità (malattie, proteine, chimiche), WEAVER supera Fine-tuning, EWC e AdapterFusion.
* AdapterFusion si è rivelato il peggiore (F1 molto basso, ≈ 63% medio).
* WEAVER è solo leggermente peggiore di MTL (≈ 1–2% in meno di F1), nonostante non usi tutti i dati insieme.
* Backward Transfer: WEAVER mostra in vari casi valori positivi (i nuovi dati migliorano anche le prestazioni sui precedenti).
* Forgetting ridotto: con WEAVER la perdita di performance sul primo dataset è molto più contenuta rispetto a Fine-tuning o EWC (es. in disease NER: altri metodi scendono fino al 30% F1, WEAVER resta al 60%).

PAPER 43

**Lifelong Language Learning with Adapter based Transformers**

* Shadab Khan, Surbhi Agarwal, P. K. Srijith
* 2022
* <https://openreview.net/forum?id=4TZzNZSJzp>

***METODO***

Il paper propone un approccio diverso: **Adapter-based Transformers**, che sfruttano la crescita modulare della rete.

* Si parte da un Transformer pre-addestrato, nello specifico **GPT-2**, che viene usato come base fissa, ovvero i suoi pesi non vengono mai riaddestrati.
* Il modello viene poi esteso in modo modulare con **adapter layers**, ossia piccoli moduli neurali che vengono aggiunti dentro ogni layer del Transformer.
* Ogni volta che arriva un nuovo task, si creano due **nuovi adapter** per ogni layer del Transformer (quindi due moduli per layer, specifici per quel task). Questi nuovi adapter vengono inizializzati da zero e addestrati solo su quel task;
* I parametri del Transformer di base e quelli degli adapter dei task precedenti vengono congelati. In questo modo, le informazioni apprese in precedenza non vengono mai sovrascritte, e ogni task ha la sua memoria separata, isolata in un piccolo set di pesi aggiuntivi

Quello che porta di nuovo questo modello è:

* **Differenza rispetto ai metodi di replay**: Un punto centrale del paper è che non usa il “replay” (cioè la rigenerazione di esempi dai task passati per evitare di dimenticarli).
* **Indipendenza dall’ordine dei task**: Nel continual learning tradizionale, l’ordine dei task può influire sulle performance (per esempio, se si impara prima un task molto diverso, si può peggiorare la memoria dei precedenti). Nel paper, gli autori testano due ordini diversi di training (ad esempio SRL→SST→WOZ e SRL→WOZ→SST) e mostrano che le metriche rimangono pressoché identiche

***RISULTATI***

* **Dataset usati**:
  + SST (Stanford Sentiment Treebank): sentiment analysis binaria.
  + SRL (Semantic Role Labeling): assegnare ruoli semantici a frasi.
  + WOZ (Wizard of Oz dialogue): tracciamento stato di dialogo per prenotazioni.
* Metriche:
  + SST → Exact Match (EM)
  + SRL → Normalized F1 (nF1)
  + WOZ → Dialogue State Exact Match (dsEM)
* **Risultati Principali**
  + **Confronto con LAMOL** (baseline)
    - SST: Adapter ~90.9, LAMOL ~90.9 (simile)
    - SRL: Adapter ~68.0, LAMOL ~68.4 (leggermente inferiore)
    - WOZ: Adapter ~88.5, LAMOL ~85.8 (superiore)
  + **Esperimento con 4 task** (SST, SRL, WOZ): Migliore di LAMOL su 2 task (WOZ e Amazon).
  + **Indipendenza dall’ordine dei task**: Cambiando l’ordine di training, le performance restano stabili.
  + **Efficienza computazionale**
    - Training time: Adapter 2650s vs LAMOL 2919s (più veloce).
    - Parametri: crescita del ~13.9% per task, molto contenuta.

PAPER 44

**Continual Learning for Natural Language Generations with Transformer Calibration**

* Peng Yang, Dingcheng Li, Ping Li
* 2022
* <https://aclanthology.org/2022.conll-1.4/>

Il paper introduce un nuovo approccio basato su “**Transformer Calibration**”, che introduce **moduli leggeri di calibrazione neurale** per mitigare il forgetting.

Le **componenti principali** sono:

* **Attention Calibration Module (ACM):** Inserito nei blocchi di multi-head attention.Aggiunge una matrice appresa che funge da soft mask per pesare diversamente le connessioni tra token.Agisce sia nel forward pass (predizione) che nel backward pass (regolarizzazione dell’update dei parametri).
  + In continual learning, quando il modello affronta nuovi task, i pesi di attenzione cambiano troppo e le relazioni fra token imparate prima vengono “sovrascritte”, generando catastrophic forgetting.
  + Il paper introduce una matrice di calibrazione (una per ogni layer di attenzione), che modula l’importanza tra coppie di token.
  + **Soft mask dinamica**: questa matrice agisce come una “maschera morbida” che enfatizza o riduce certe connessioni token-token. Il modello può riusare parti di attenzione rilevanti per task precedenti, evitando di distruggerle.
  + **Task awareness**: Poiché la matrice di calibrazione viene appresa continuamente lungo i task, diventa implicitamente rappresentativa del task, sostituendo un task embedding esplicito.
  + **Doppio ruolo (forward e backward)**:
    - **Forward pass**: modifica i valori di attenzione usati per la predizione.
    - **Backward pass**: regolarizza la direzione dell’aggiornamento dei parametri, limitando variazioni drastiche dove la matrice di calibrazione segnala alta importanza.
* **Feature Calibration Module (FCM):** Inserito dopo ogni layer per scalare le feature maps.Implementato come un vettore moltiplicativo che regola l’importanza dei neuroni.
  + Dopo ogni blocco Transformer, si ottiene una feature map. Quando si passa a un nuovo task, queste rappresentazioni latenti possono essere troppo “adattate” ai dati nuovi, causando forgetting dei pattern vecchi.
  + Quindi, si introduce un vettore di calibrazione per ogni layer, che scala ogni dimensione della feature map. Il vettore “decide” quanto ogni dimensione delle feature deve essere adattata al nuovo task.
* **Interleaved Optimization:** Durante l’addestramento, alterna l’aggiornamento di parametri della calibrazione e parametri del modello base
  + Anche con moduli ACM e FCM, bisogna bilanciare quanto aggiornare i parametri del modello base e quelli dei moduli di calibrazione
  + Il paper usa una ottimizzazione interleaved, in cui si alternano i due aggiornamenti.

Riguardo alle tecniche di **Continual Learning**, il lavoro si differenzia dalle tre categorie classiche di Regularization (EWC, SI, LwF), Replay (buffer di esempi) o Architectural (nuovi parametri per ogni task).

* Qui l’approccio è un neuron calibration approach, che non usa memoria di replay, non introduce moduli specifici per task, è leggero e applicabile a diversi modelli Seq2Seq.
* In sostanza, bilancia stabilità (non dimenticare) e plasticità (imparare cose nuove) calibrando attenzione e feature maps.

***RISULTATI***

Il metodo riduce significativamente il forgetting rispetto a EWC e replay-based.

* Senza memoria: già superiore a molte alternative.
* Con memoria: ancora più competitivo.

Rispetto ai metodi classici, è generale (funziona per più task e modelli), efficace (migliora metriche di NLG e riduce il forgetting) e leggero (nessuna memoria di replay, nessun modulo task-specific).

PAPER 45

**TL-CL: Task And Language Incremental Continual Learning**

* Shrey Satapara, P. K. Srijith
* 2024
* <https://openreview.net/forum?id=xFGy2e7hjc&referrer=%5Bthe%20profile%20of%20Shrey%20Satapara%5D(%2Fprofile%3Fid%3D~Shrey_Satapara1)>

***METODO***

Il paper introduce un nuovo setup di Continual Learning in NLP chiamato **Task and Language Incremental Continual Learning (TLCL)**:

* **Task-Incremental CL (TICL)**: imparare nuovi task nella stessa lingua.
* **Language-Incremental CL (LICL)**: imparare la stessa task in nuove lingue.
* **Task & Language Incremental CL (TLCL)**: combinazione delle due: il modello deve imparare nuove task in lingue già viste, nuove lingue per task già viste, o task e lingue nuove in sequenza

Il problema nasce dal fatto che un modello multilingue deve adattarsi a nuove esigenze (nuovi task/lingue) senza dimenticare quelle precedenti (catastrophic forgetting).

Il modello proposto è **TLSA (Task and Language-Specific Adapters)**, una tecnica di parameter-efficient fine-tuning (PEFT). Il modello di base è mT5-small (encoder-decoder multilingue). TLSA aggiunge due tipologie di adapter nei layer del transformer:

* **Task-specific adapters**: condivisi tra lingue diverse → accumulano conoscenza del task indipendentemente dalla lingua
  + Allenati per ogni task ma condivisi tra tutte le lingue.
  + Catturano concetti astratti del compito (es. come ragionare per QA o classificazione).
  + Sono riutilizzati ogni volta che la stessa task compare in una lingua diversa.
  + Durante il training successivo vengono aggiornati con regolarizzazione EWC, per evitare di “sovrascrivere” ciò che hanno imparato prima.
* **Language-specific adapters**: condivisi tra task diversi → accumulano conoscenza della lingua indipendentemente dal task
  + Allenati per ogni lingua ma condivisi tra tutte le task.
  + Imparano le caratteristiche linguistiche (morfologia, sintassi, vocabolario).
  + Ogni volta che arriva una nuova task in una lingua già vista, questi adapter vengono riutilizzati e aggiornati con EWC.

Entrambi vengono regolarizzati con **Elastic Weight Consolidation (EWC)** per ridurre la catastrofica dimenticanza.

* Dopo aver finito di addestrare una coppia (task t, lingua l), si calcola la Fisher Information Matrix (F) per stimare quanto ciascun parametro sia importante per quella coppia.
* Quando si passa a un nuovo task o lingua, si aggiunge una penalità nella loss che limita le modifiche ai parametri più “importanti”.

Il paper introduce anche una variante: **TLSA-PI (Parameter Isolation)**, che sarebbe una versione “forte” di TLSA.

* TLSA-PI salva gli adapters di ogni coppia task-lingua dopo l’allenamento.
* In inferenza, carica direttamente quelli corrispondenti alla coppia desiderata.
* Così elimina completamente il forgetting, ma aumenta la memoria usata (torna alla crescita polinomiale), e non sfrutta la regolarizzazione EWC.

***RISULTATI***

Da i **risultati principali** ottenuti, si nota che:

* **TLSA-PI > tutte le altre tecniche** (miglior trade-off forgetting/performance, ma più memoria).
* **TLSA ≈ Experience Replay (ER)**: TLSA ottiene prestazioni comparabili a ER ma senza usare buffer di dati storici, quindi più realistico in scenari sensibili alla privacy o con vincoli di storage.
* **TLSA > altri PEFT** (AT, TSA, MAD-X) sia in complete che partial setup.
* In scenari con **single-language constraint**, ER rimane molto forte, ma TLSA è comunque competitivo.

PAPER 46

**Investigating Continual Pretraining in Large Language Models: Insights and Implications**

* Çağatay Yıldız, Nishaanth Kanna Ravichandran, Nitin Sharma, Matthias Bethge, Beyza Ermis
* 2024
* <https://arxiv.org/abs/2402.17400>

***METODO***

L’idea del paper è confrontare **strategie di continual domain-adaptive pretraining** con l’alternativa più classica del **domain-adaptive pretraining indipendente per ogni dominio**.

Gli autori testano sia modelli decoder-only che encoder-only:

* Decoder-only: GPT-2 (nelle varianti S, M, L, XL) e Llama2-7B.
* Encoder-only: RoBERTa-base e RoBERTa-large.

Per capire le dinamiche utilizza questo setup:

* Si parte da un modello pre-addestrato
* Si fa pretraining sequenziale su domini diversi
* Dopo ogni dominio, il modello è valutato su:
  + **CPT** (Continual PreTraining perplexity): prestazioni sul dominio appena appreso.
  + **LC** (Last Checkpoint perplexity): prestazioni finali su tutti i domini.
  + **FG** (Forgetting): differenza di performance rispetto al miglior checkpoint passato.
  + **Backward transfer**: effetto dell’apprendimento su domini passati.
  + **Forward transfer**: capacità di generalizzare su domini futuri non ancora visti.
* Due ordini di presentazione dei domini:
  + **Similar-order**: domini semanticamente vicini in sequenza.
  + **Random-order**: domini mescolati casualmente.

***RISULTATI***

* **Benefici del CL:**
  + Tutti i modelli GPT-2 migliorano grazie al continual pretraining
  + Llama2-7B peggiora sempre: i domini usati erano troppo piccoli rispetto alla sua scala (serve >100MB di dati per dominio per avere benefici).
* **CL vs Domain-Adaptive Pretraining (DAPT)**:
  + Il CL è sempre superiore al semplice adattamento dominio per dominio, perché favorisce la trasferibilità delle conoscenze.
* **Effetto della dimensione del modello**
  + **Modelli grandi** → meno forgetting, migliori performance finali.
  + **Modelli piccoli** → più sensibili, imparano di più ma dimenticano di più.
* **Ordine dei domini**
  + **Random-order** → meno forgetting, migliore performance complessiva, più forward transfer.
  + **Similar-order** → vantaggioso per la specializzazione (apprendimento più graduale e coerente).
* **Downstream tasks (BIG-Bench)**
  + Su GPT-2, il CL migliora le prestazioni su task coerenti con i domini visti (es. Physics dopo dominio Physics).
  + Su Llama2-7B, il CL degrada le performance (anche sotto GPT-1).
* **RoBERTa**
  + Risultati molto diversi dai decoder-only: non mostra forgetting e spesso ottiene forward transfer positivo.
  + Possibile che la diversa architettura (encoder con bottleneck) spieghi il comportamento.

PAPER 47

**Boosting Large Language Models with Continual Learning for Aspect-based Sentiment Analysis**

* Xuanwen Ding, Jie Zhou, Liang Dou, Qin Chen, Yuanbin Wu, Chengcai Chen, Liang He
* 2024
* <https://arxiv.org/abs/2405.05496>

***METODO***

L’obiettivo del paper è sviluppare un **modello LLM-CL (Large Language Model - Continual Learning)** che:

* sfrutti la conoscenza commonsense già presente negli LLM,
* eviti la perdita di prestazioni nei domini precedenti,
* distingua conoscenza invariata tra domini e conoscenza specifica di dominio.

Il framework proposto ha tre componenti principali

* **LLMs-based ABSA model**: Costruire un modello di base capace di svolgere tutti i sottocompiti ABSA (AE, ABSC, JOINT) in modo generativo, sfruttando la conoscenza generale già presente in un LLM.
  + Il backbone è LLaMA-7B, un large language model open-source.
  + Gli autori convertono ogni task in una istruzione testuale (instruction tuning).
  + Così il modello impara a generare direttamente la risposta testuale, senza architetture dedicate.
  + Per evitare di riaddestrare miliardi di parametri, usano LoRA (Low-Rank Adaptation)
  + Il risultato è un modello generativo ABSA, efficiente in memoria, che può apprendere nuovi domini aggiungendo soltanto piccoli adapter.
* **Domain Knowledge Decoupling (DKD)**: l’idea è quella di separare conoscenza invariata tra domini da quella specifica di dominio.
  + Gli autori introducono due tipi di adapter LoRA:
    - **Domain-invariant adapter**: cattura conoscenze generali di sentimento e linguaggio, addestrato su replay data da tutti i domini precedenti.
    - **Domain-variant adapters**: uno per ciascun dominio, addestrato solo sui dati di quel dominio.
  + Il tutto viene regolarizzato da un **vincolo ortogonale**, che forza gli adapter specifici a imparare direzioni nello spazio dei parametri indipendenti da quelle dell’adapter generale.
  + Il risultato è che il modello impara sia conoscenza condivisa che specifica, limitando il catastrophic forgetting e le interferenze
* **Domain Knowledge Warmup (DKW)**: il problema è che gli adapters specifici rimangono fissi, mentre quello invariato continua a cambiare, e quindi c’è un rischio mismatch. La soluzione è usare replay (pochi esempi salvati dai vecchi domini) per riallineare l’adapter invariato con ciascun adapter specifico.
  + Dopo aver addestrato tutti i domini, si esegue una fase di warmup:
    - Si unisce ogni adapter specifico con quello generale
    - Si fine-tuna solo l’adapter invariato (il resto resta fisso) usando i replay data da tutti i domini.
  + In questo modo l’adapter generale “si riallinea” a tutti gli adapters specifici, armonizzando le loro distribuzioni di parametri.
  + Si ha una maggiore coerenza tra le rappresentazioni, miglior generalizzazione cross-domain.

Ultima cosa da considerare è che nella fase di test non si conosce a priori il dominio di ogni esempio.

* **Domain Positioning (DP)**: in test non si conosce il dominio a cui appartiene il testo, e bisogna capire quale adapter specifico usare.
  + Gli autori progettano un modulo in due fasi:
    - **Domain Prototype Learning**: Durante l’addestramento, calcolano per ogni dominio una rappresentazione media (prototipo), e calcolano anche una covarianza condivisa.
    - **Nearest Domain Indexing:** Per un nuovo testo , trovano il dominio più simile usando distanza di Mahalanobis, e selezionano l’adapter variant del dominio con distanza minima.
  + Quindi il modello decide automaticamente quale conoscenza di dominio usare, senza bisogno di sapere da quale dataset proviene l’input.

***RISULTATI***

Considerando come **baselines**:

* Metodi PLMs-based (EWC, OWM, HAT, CLASSIC, ecc.)
* Metodi LLMs-based (SEQUENCE, REPLAY, O-LoRA, AdaLoRA, Multi-task).

Dai **risultati principali** si vede che:

* **Catastrophic Forgetting**: Anche gli LLM soffrono di forgetting, ma meno dei PLM tradizionali. LLM-CL riduce notevolmente il fenomeno, avvicinandosi alle prestazioni multitask (upper bound).
* **Prestazioni su ABSC**: LLM-CL → Accuracy 0.9491, Macro-F1 0.9143
* **Prestazioni su AE e JOINT**: LLM-CL supera nettamente le baseline:

PAPER 48

**Transformers for Supervised Online Continual Learning**

* Jorg Bornschein, Yazhe Li, Amal Rannen-Triki
* 2024
* <https://arxiv.org/abs/2403.01554>

***METODO***

Il lavoro esplora come usare i **Transformer per l’apprendimento continuo supervisionato** in streaming (Supervised Online Continual Learning), cioè in scenari dove i dati arrivano in sequenza non stazionaria e il modello deve adattarsi rapidamente ai cambiamenti (plasticità), conservare conoscenza a lungo termine, e ottimizzare la perdita predittiva cumulativa.

Gli autori vogliono combinare l’apprendimento in-context (rapido adattamento grazie al contesto) e l’apprendimento nei pesi (aggiornamento parametrico con SGD) in un unico modello online. Vengono proposte due varianti principali del Transformer:

* **2-Token Transformer**: Usa un modello decoder-only (simile ai language model). Ogni esempio (x,y) viene presentato come due token consecutivi. Il modello è addestrato a predire solo il token y (l’etichetta), ignorando la perdita su x
* **Privileged Information Transformer (pi-Transformer)**: È la proposta principale del paper. Ogni input x è accompagnato da una “informazione privilegiata” y (l’etichetta) solo come input ai meccanismi di attenzione:
  + y contribuisce alle key e value dell’attenzione (ma non alle query), e si impone una maschera causale che impedisce al token corrente di “vedere” la sua etichetta (solo quelle precedenti).
  + in questo modo il modello può usare la storia dei (x, y) precedenti per adattarsi, senza violare la causalità.

Per le **tecniche di Continual Learning**, il paper combina tre meccanismi chiave:

* **Online Training stile Transformer-XL**
  + Il modello viene aggiornato in modo incrementale con SGD su piccoli blocchi sequenziali di lunghezza S≈100.
  + Si mantiene una cache (KV-cache) di chiavi e valori per le ultime C osservazioni. Questo consente contesto a lungo raggio (fino a 1024 token).
  + Usa Multi Query Attention (MQA) per efficienza di memoria.
* **Replay-Streams: In-Order Replay**
  + Si creano E flussi di replay paralleli (num-streams). Uno “principale” che avanza sequenzialmente e misura la perdita reale, e altri E−1 che rieseguono segmenti passati della sequenza per simulare multi-epoch training in contesto strettamente online.
  + Periodicamente i flussi vengono resettati casualmente. Si ha un replay uniforme senza buffer espliciti.
  + Questo migliora la stabilità e induce meta-learning implicito, poiché i parametri devono funzionare bene sia per dati recenti che passati.
* **In-context + In-weight Learning**
  + Gli autori mostrano che il Transformer può unire due dinamiche:
    - **In-context learning**: rapido adattamento tramite il contesto (attivazioni interne).
    - **In-weight learning**: adattamento strutturale tramite i pesi (SGD).
  + L’interazione tra i due meccanismi produce miglioramenti persistenti e riduce il “catastrophic forgetting”

***RISULTATI***

* **Dati sintetici: Split-EMNIST e Split-CIFAR**
  + Gli autori costruiscono sequenze artificiali per simulare un ambiente piece-wise non stazionario, dove i dati cambiano improvvisamente (come accade nei problemi di continual learning). Vogliono osservare se e come i Transformer riescono ad adattarsi a cambiamenti improvvisi e diventare few-shot learners nel tempo.
  + Risultati osservati:
    - All’inizio (primi 10–20 task) il modello fatica: la distribuzione cambia e l’apprendimento in-context non è ancora “emerso”.
    - Dopo un certo numero di task, il modello impara a imparare. Mostra un comportamento tipico del meta-learning emergente
    - Il replay-stream è cruciale. Senza replay, il modello non riesce a stabilizzare le conoscenze né a imparare le regole di adattamento. Con replay, invece, la rete “vede” più volte gli stessi pattern e impara strategie generali per adattarsi.
    - Il pi-Transformer raggiunge performance più stabili e con meno variabilità rispetto al 2-token Transformer, che invece è più lento e sensibile agli iperparametri.
  + Quindi, nei dati sintetici, emerge chiaramente che:
    - il modello sviluppa abilità di few-shot learning (rapido adattamento).
    - il replay induce una forma di meta-learning implicito (il modello “impara a imparare”).
* **CLOC Benchmark (Continual Learning On Camera — Geo-localization)**
  + Esperimenti reali con il CLOC Benchmark (geo-localization). Il CLOC (Continual Learning On Camera) contiene circa 39 milioni di immagini cronologiche con etichette geografiche.
  + Due scenari sperimentali principali
    - **Feature extractor pre-addestrato e congelato**: Il Transformer lavora su feature estratte da reti pre-addestrate, e solo il Transformer viene addestrato online.
      * Il risultato è che quasi raddoppia l’accuracy rispetto al precedente stato dell’arte.
    - **Feature extractor appreso online (from scratch)**: Qui il feature extractor viene addestrato insieme al Transformer.
      * Anche in questo caso si ottiene 67% di accuratezza, superiore a tutti i baseline precedenti.

PAPER 49

**Learning to Route for Dynamic Adapter Composition in Continual Learning with Language Models**

* Vladimir Araujo, Marie-Francine Moens, Tinne Tuytelaars
* 2024
* <https://arxiv.org/abs/2408.09053>

***METODO***

Il paper affronta il problema del Continual Learning (CL) nei **modelli di linguaggio pre-addestrati (PLMs)**, ovvero la capacità di imparare nuovi compiti in sequenza senza dimenticare quelli precedenti. Gli autori si concentrano su un **approccio parameter-efficient**, cioè che evita di riaddestrare tutto il modello: usano adapters o moduli LoRA per aggiungere capacità specifiche per ogni nuovo task mantenendo congelato il backbone del PLM.

Il lavoro propone **L2R – “Learning to Route”**, un metodo per:

* **Addestrare nuovi moduli PEFT (adapters)** per ogni nuovo task in modo isolato (senza interferire con i precedenti);
* **Imparare una funzione di routing** che, durante l’inferenza, combina dinamicamente gli adapters più appropriati a seconda dell’input.

La **struttura generale** è data da:

* **Backbone**: un PLM congelato (BERT-base per MTL5 e WOS, AfroXLMR-base per AfriSenti);
* **Adapters**: moduli LoRA (Low-Rank Adaptation) aggiunti a ciascun layer del PLM e addestrati uno per volta per i diversi task;
* **Router network (R)**: una piccola rete neurale addestrata dopo il training dei moduli per decidere quali adapters attivare per un dato input.

Si tratta di un **processo in due fasi**

* **Training isolato degli adapters**
  + Per ogni nuovo task, si addestra solo l’adapter corrispettivo, mentre gli altri restano congelati.
  + Questo evita interferenza catastrofica con conoscenze precedenti.
  + Ogni adapter cattura la conoscenza specifica del task.
* **Learning to Route**
  + Si usa una memoria M con un piccolo insieme di esempi dei task precedenti.
  + Il router R è addestrato su questa memoria per imparare a combinare dinamicamente gli adapters a seconda dell’input.
  + La decisione di routing avviene tramite Gumbel-sigmoid, che permette di modellare scelte binarie (attiva/non attiva un adapter) in modo differenziabile.

L2R integra elementi di diverse **strategie di CL**:

* **Parameter Isolation**: Ogni task ha i propri moduli PEFT addestrati indipendentemente.
* **Replay/Memory-based methods**: Memoria non parametrica con esempi precedenti, usata non per replay ma per router training.
* **Dynamic Routing**: Router che seleziona o combina dinamicamente adapters in base all’input.
* **Local adaptation (MbPA)**: Il router è addestrato sulla memoria per adattarsi localmente alle distribuzioni viste, ma solo una volta prima dell’inferenza.

***RISULTATI***

Questi **Continual Learning setups**

* CIL (Class-Incremental Learning): il modello non conosce l’identità del task durante l’inferenza;
* TIL (Task-Incremental Learning): l’identità del task è nota durante l’inferenza (scenario più facile).

Vengono confrontati con queste **baselines**:

* Lower-/Upper-Bound: versioni limite di adapters indipendenti o ideali;
* DAM (Dynamic Adapter Merging)
* EPI (Efficient Parameter Isolation)
* ProgPrompt
* MoCL (Modular Compositional Learning)

Abbiamo che i **risultati principali** sono:

* Performance cresce con la dimensione della memoria (da 1% a 30% dei dati). Dimostra che una memoria più ampia migliora la qualità del routing.
* L2R-wavg e L2R-merge sono molto efficienti, vicini al costo computazionale di DAM e molto più leggeri di MoCL.

PAPER 50

**AdaPrefix++: Integrating Adapters, Prefixes and Hypernetwork for Continual Learning**

* Sayanta Adhikari, Dupati Srikar Chandra, P. K. Srijith, Pankaj Wasnik, Naoyuki Oneo
* 2025
* <https://ieeexplore.ieee.org/abstract/document/10943897>

***METODO***

Gli autori si concentrano su modelli **Transformer-based Pretrained Large Models (PLMs)** (come BERT o Vision Transformers) e propongono una nuova architettura che combini tecniche di **fine-tuning efficienti** in termini di parametri (PEFT) per il CL.

Gli autori introducono due modelli principali:

* **AdaPrefix**:
  + Una prima versione che combina due tecniche PEFT:
    - **Adapters**: piccole reti feed-forward aggiunte dopo i layer del Transformer, addestrate su nuovi task con pochi parametri.
    - **Prefixes**: vettori addestrabili concatenati alle chiavi e ai valori nei layer di attenzione del Transformer.
  + AdaPrefix integra prefixes nella Multi-Head Attention e adapters dopo la Feed-Forward Network del Transformer.
  + Ogni task ha un proprio insieme di parametri specifici, mentre i parametri del PLM restano congelati.
  + Questo approccio è basato su **parameter isolation**. Ogni task ha il suo spazio di adattamento, evitando interferenze.
* **AdaPrefix++** (proposta principale):
  + Estende AdaPrefix introducendo un **Hypernetwork** per generare automaticamente i prefixes di ogni task.
  + Componenti principali:
    - **Backbone**: PLM congelato (es. ViT-B, DeiT-S).
    - **Adapters**: rimangono task-specifici e addestrabili.
    - **Hypernetwork**: una rete condivisa che genera i prefixes di ogni task a partire da:
      * Task embedding (rappresenta il task)
      * Layer embedding ​(indica lo strato del PLM)

Tecniche di **Continual Learning** impiegate:

* **Parameter isolation** (per AdaPrefix): Ogni task ha parametri dedicati (no forgetting, ma nessun transfer)
* **Knowledge transfer via shared parameters** (per AdaPrefix++): Hypernetwork condivisa genera prefixes per tutti i task
* **Regularization-based CL**: Penalizza il cambiamento eccessivo dell’hypernetwork, simile a EWC o distillation
* **Task embeddings:** Permettono al modello di differenziare i task senza memorizzare dati passati

***RISULTATI***

* Backbone (PLMs usati)
  + Vision Transformers: ViT-L (307M), ViT-B (86M)
  + DeiT-S (21M) e DeiT-T (5M)
* Baseline confrontate
  + Classiche CL: EWC, Experience Replay
  + PEFT CL: Adapter, LAE-Prefix, LAE-Adapter
  + Prompt/Prefix-based: L2P, DualPrompt, CODA-Prompt, S-Prompt, HiDe-Prompt (SOTA 2024)
* Metriche
  + Average Accuracy (Aₜ) → accuratezza media su tutti i task alla fine del training
  + Forward Transfer (FWT) → misura quanto l’apprendimento di nuovi task migliora i precedenti

Dai risultati principali, si ottiene che:

* AdaPrefix++ supera tutti i baseline in TIL, CIL e DIL
* Per il forgetting, AdaPrefix ha zero forgetting (grazie alla separazione dei parametri), mentre AdaPrefix++ ha <1% di forgetting grazie alla regolarizzazione.
* AdaPrefix++ mostra FWT positivo. Dimostra trasferimento di conoscenza tra task.

PAPER 51

**HyperAdapter: Generating Adapters for Pre-Trained Model-Based Continual Learning**

* Qizhe Zhang, Ruichuan An, Bocheng Zou, Zhi Zhang, Shanghang Zhang
* 2024
* <https://openreview.net/forum?id=29sul3tAEa>

Il lavoro affronta il classico problema del catastrophic forgetting nei modelli neurali quando vengono addestrati in sequenza su più compiti (continual learning, CL). In particolare, propone un metodo senza buffer di rehearsal (rehearsal-free), basato sull’uso di modelli pre-addestrati e di una hypernetwork in grado di generare automaticamente adapter layers per ciascun compito.

Si tratta di **HyperAdapter**, e la struttura generale è composta da tre parti principali:

* **Backbone pre-addestrato** (es. Vision Transformer ViT-B/16): Rimane congelato per evitare dimenticanza catastrofica.
* **Hypernetwork (h)**: Genera dinamicamente i pesi degli adapter per ogni nuovo compito.
* **Task Dictionary (C)**: Contiene coppie di chiavi e embedding (k, z) che rappresentano “memorie episodiche” dei compiti appresi.

Il sistema funziona con questi passaggi:

* **Selezione del task embedding**:
  + Per ogni input x, si genera una query q(x) (feature estratta dal backbone).
  + Questa viene confrontata con le chiavi k nel task dictionary per trovare l’embedding più simile.
  + Durante il training, l’identità del task è nota, quindi si usa direttamente. In inference, invece, la selezione avviene automaticamente tramite matching.
* **Generazione degli adapter**
  + Una volta ottenuto il task embedding z, la hypernetwork genera i parametri dell’adapter.
  + ​L’adapter elabora quindi l’output del layer Transformer, e il risultato viene sommato al residuo del layer.

Ci sono **due varianti**

* **HAmodel**: una sola hypernetwork per l’intero modello.
* **HAblock**: una hypernetwork per ciascun blocco → più parametri ma migliori performance e minore forgetting.

Tecniche di **Continual Learning** coinvolte:

* **Rehearsal-free**: nessuna memoria dei dati passati.
* **Parameter isolation**: la knowledge del backbone resta congelata.
* **Dynamic expansion**: nuovi task → nuovi embeddings e nuovi adapter generati.
* **Meta-learning** (via hypernetwork): la hypernetwork apprende a generare pesi in base al contesto del task.
* **Task inference automatica**: grazie al task dictionary non serve sapere esplicitamente quale task si sta inferendo (class-incremental learning).

***RISULTATI***

HyperAdapter supera tutti i benchmark testati, anche i migliori metodi prompt-based e adapter-based.

* +1.06% su ImageNet-R rispetto a CODA-P.
* +4.8% su DomainNet rispetto a EASE.
* In alcuni casi supera perfino il multi-task upper bound, cioè il training con tutti i dati simultaneamente. Vision Transformer (ViT-B/16) pre-addestrato su ImageNet-21K.

Ablation studies

* Rimuovere il task dictionary causa un crollo di ~30% in accuracy.
* Rimuovere il layer normalization negli adapter → -60% performance.
* Le positional embeddings migliorano leggermente la performance.
* Più parametri → più accuratezza, ma con rendimenti decrescenti.

PAPER 52

**Experience Replay Addresses Loss of Plasticity in Continual Learning**

* Jiuqi Wang, Rohan Chandra, Shangtong Zhang
* 2025
* <https://arxiv.org/abs/2503.20018>

***METODO***

Il paper affronta la **perdita di plasticità nei modelli di deep learning** durante continual learning (CL).

Nel CL, un modello impara da un flusso continuo di compiti (tasks) senza conoscere i confini tra essi. **L’experience replay** (ER) può risolvere la perdita di plasticità nei modelli di deep learning, soprattutto se combinata con architetture Transformer.

Il paper confronta tre architetture principali:

* **MLP** (Multi-Layer Perceptron): rete feed-forward standard, con ReLU. Soffre rapidamente la perdita di plasticità.
* **RNN** (Recurrent Neural Network): include memoria temporale, ma anch’essa perde plasticità e apprende lentamente.
* **Transformer** (solo self-attention layers): una versione semplificata (senza feed-forward, positional encoding o layer norm). È l’unico modello che mantiene la plasticità e continua a migliorare col tempo.

Quindi **l’Experience Replay (ER)** è la tecnica principale analizzata.

* Il modello mantiene un buffer di memoria (di dimensione 100) contenente una piccola parte delle esperienze passate.
* Durante il training, ogni nuovo esempio viene combinato con il contenuto del buffer per formare un input esteso.
* Il buffer viene aggiornato in modo FIFO (First-In, Last-Out).
* Il modello impara quindi dai dati recenti e dal contesto del buffer, simulando una forma di in-context learning.

***RISULTATI***

Gli esperimenti coprono tre categorie di problemi di continual learning:

* **Continual** Regression: confronto tra MLP, ERMLP (MLP con replay), RNN, Transformer.
  + **Risultati**:
    - MLP → rapido collasso della plasticità, errore crescente.
    - Transformer + ER → nessuna perdita di plasticità; performance migliora con più task.
    - RNN e ERMLP → incapaci di apprendere efficacemente.
* **Continual Classification**: confronto tra MLP e Transformer con replay.
  + **Risultati**:
    - MLP → test accuracy in costante calo (perdita di plasticità).
    - Transformer + ER → accuratezza crescente; nessun degrado anche dopo migliaia di task.
    - Il Transformer aveva meno dell’1% dei parametri dell’MLP, ma prestazioni migliori nel lungo periodo.
* **Continual Policy Evaluation**: algoritmo di Semi-gradient TD-learning.
  + **Risultati**:
    - MLP → errore MSVE cresce con il tempo → perdita di plasticità.
    - Transformer + ER → errori decrescenti; nessun degrado.
    - RNN e ERMLP → non mostrano apprendimento significativo.

PAPER 53

**Efficient Continual Learning for Small Language Models with a Discrete Key-Value Bottleneck**

* Andor Diera, Lukas Galke, Fabian Karl, Ansgar Scherp
* 2024
* <https://arxiv.org/abs/2412.08528>

***METODO***

Gli autori propongono un nuovo meccanismo architetturale chiamato **Discrete Key-Value Bottleneck (DKVB)** per abilitare un apprendimento continuo efficiente, riducendo il rischio di dimenticanza catastrofica senza introdurre componenti specifici per compito o grandi costi computazionali.

Questo modello Discrete Key-Value Bottleneck (DKVB) si inserisce **tra l’encoder e il decoder** del modello linguistico e opera in tre fasi:

* **Encoding**: Il testo è codificato da un encoder (BERT, RoBERTa o DistilBERT) in una rappresentazione vettoriale continua
* **Discretizzazione tramite codebook**: Il vettore viene suddiviso in C teste (“heads”), ciascuna associata a un codebook di coppie key–value discreti.
  + Ogni key rappresenta un punto nello spazio delle feature, mentre la value associata è un vettore addestrabile.
  + Per ogni input, si cerca la key più vicina (L2 distance) e si recupera la value corrispondente.
  + Le chiavi sono inizializzate prima dell’addestramento (tramite Exponential Moving Average sui dati di input) e poi mantenute fisse, mentre i valori sono addestrabili.
* **Decoding**: Le values ottenute vengono combinate e passate a un decoder che può essere:
  + **Parametrico**, con pesi addestrabili (es. un layer lineare + softmax);
  + **Non parametrico**, semplicemente un softmax sui vettori mediati.

Riguardo alle **tecniche di Continual Learning**, gli autori collocano il loro approccio nel contesto delle 5 famiglie classiche di CL: Regularization-based (es. EWC), Replay-based (es. DER++), Optimization-based (es. OWM), Architecture-based (es. CTR), Instruction-based (solo per LLM). Tuttavia, il DKVB non appartiene pienamente a nessuna di queste categorie.

***RISULTATI***

Per l’analisi vengono usati questi **modelli**

* BERT-base, RoBERTa, DistilBERT (encoder-only).
* L’encoder è congelato nella maggior parte degli esperimenti; solo il DKVB e il decoder sono addestrati.

E vengono considerate queste **baseline** di confronto

* EWC (regularization), DER++ (replay), OWM (optimization), CTR (capsule-based architecture), NCL (naive continual learning, senza protezione contro la dimenticanza).

**Varianti di DKVB testate**:

* Parametric (DKVB-P) o Non-parametric (DKVB-NP) decoder.

**Key initialization**:

* **Incremental**: aggiornamento delle chiavi per ogni task
* **Oracle**: chiavi inizializzate su tutto il dataset (accesso completo)
* **Generic**: chiavi inizializzate su un corpus generale (Wikipedia)

Dai **risultati principali**, si vede che:

* **Prestazioni in CL**
  + CIL (20NG): DKVB-NP Oracle raggiunge 97.06% di accuratezza, superando EWC (96.8%) e CTR (95.5%).
  + TIL (4GLUE): DKVB-NP Oracle e Generic ottengono performance comparabili a CTR (≈70%).
  + DIL (DSC): DKVB meno efficace (≈82%) poiché la compartimentalizzazione delle key-value riduce il transfer tra domini simili.
  + Single-head CIL (R8/R52): Solo DKVB mantiene buone prestazioni (R8: 81.2%, R52: 47.8%) mentre altri modelli collassano per forgetting.
* **Efficienza computazionale**
  + DKVB ha tempi di training per epoca simili al semplice “frozen BERT” (≈2s per 20NG), molto inferiori a CTR o DER++.
  + L’inizializzazione delle chiavi è lineare nella dimensione del dataset ma avviene una sola volta.
* **Backward Transfer (BWT)**
  + DKVB-NP Oracle e Generic mostrano valori BWT prossimi a 0, indicando assenza di forgetting.
  + Al contrario, metodi come DER++ o NCL hanno BWT fortemente negativi.

PAPER 54

**HAM: Hierarchical Adapter Merging for Scalable Continual Learning**

* Eric Nuertey Coleman, Luigi Quarantiello, Samrat Mukherjee, Julio Hurtado, Vincenzo Lomonaco
* 2025
* <https://arxiv.org/abs/2509.13211>

***METODO***

Nei grandi modelli pre-addestrati, un approccio diffuso è il **Parameter-Efficient Fine-Tuning (PEFT)**, come **LoRA (Low-Rank Adaptation)**, che permette di adattare un modello introducendo pochi parametri addestrabili, mantenendo congelata la rete di base.

Il problema è che in scenari di CL lunghi, tenere un adapter separato per ogni task non è scalabile, e i metodi attuali non favoriscono abbastanza il transfer tra task simili, e spesso accumulano interferenze.

Il modello proposto è quindi **HAM (Hierarchical Adapter Merging)**, un nuovo approccio che combina **LoRA** e un **meccanismo gerarchico di raggruppamento** **e fusione degli adapter**.

* **Task-Specific LoRA Training**
  + Per ogni nuovo task, si addestra un adapter LoRA e un fattore di importanza
  + Durante questo addestramento, si aggiornano anche i fattori di importanza dei gruppi già esistenti, in modo da mantenere un equilibrio.
  + L’output complessivo è un modello base + adapter dei gruppi esistenti (pesati) + adapter del nuovo task.
* **Adapters Grouping**
  + Dopo aver appreso l’adapter LoRA, si calcola la cosine similarity con i gruppi esistenti.
  + Se sopra una soglia, allora viene unito al gruppo simile. Se no e non si è raggiunto il numero massimo di gruppi, allora si crea un nuovo gruppo.
  + Ogni gruppo ha un fattore di importanza (media dei valori dei membri).
  + Poi si fa un pruning selettivo, dove si conservano solo i pesi più significativi (top-k%).
  + E si fa una concatenazione intra-gruppo. Gli adapter di un gruppo vengono concatenati, aumentando la capacità di rappresentazione del gruppo.
* **Model Inference (fusione finale)**
  + Alla fine, tutti i gruppi vengono fusi in un singolo adapter tramite media pesata dei fattori di importanza
  + Così il modello può fare inferenza su tutti i task, senza necessità di ID del task o routing.

Tra le tecniche di **Continual Learning**, HAM si posiziona nella categoria **parameter-isolation/merging** del CL:

* Evita il catastrofico forgetting mantenendo e fondendo i LoRA invece di sovrascriverli.
* Favorisce il transfer tra task simili attraverso il raggruppamento per similarità.
* È scalabile perché il numero di gruppi è limitato (≪ numero di task).
* Gerarchia e pruning riducono interferenza e complessità, migliorando sia efficienza che stabilità.

***RISULTATI***

* **Accuracy**: HAM ottiene la media migliore. 70.08% su 50 task, superando SD-LoRA (66.22%) e nettamente meglio dei prompt-based. Su dataset complessi (CUB-200), HAM guadagna molto (55.17% vs 47.56% di SD-LoRA).
* **Forgetting**: HAM ottiene valori bassi (9.75% medio), migliori di InfLoRA o SD-LoRA.
* **Scalabilità**: Su sequenze molto lunghe (fino a 100 task), HAM resta stabile, mentre altri metodi collassano o esauriscono memoria GPU.
* **Efficienza**: quasi 2× più veloce di SD-LoRA sia in training che in inferenza. Inoltre, pruning + concatenazione riducono i parametri mantenendo o aumentando l’accuracy.