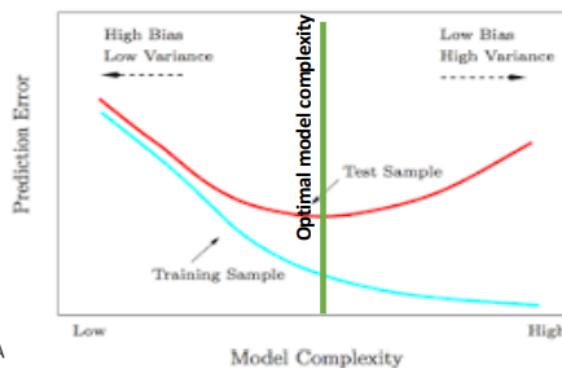
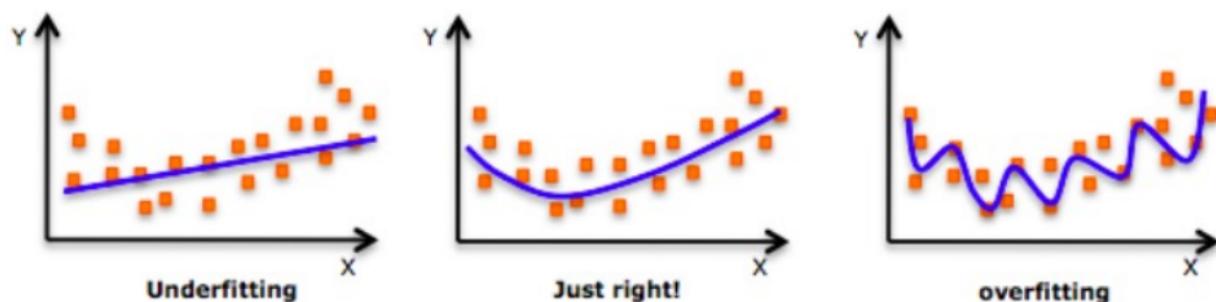


# Lezione 5 18/10/2024

## Regularization

Le funzioni di regolarizzazione sono importanti per ridurre il problema dell'overfitting.



CED MACHINE LEARNING - ENZA MESSINA

Per raggiungere buoni risultati, aumentando la complessità del modello, la funzione imparata potrà fittare bene i training data, ma avremo una varianza alta e quindi l'errore sui dati di training sarà minimizzato, mentre l'errore sui dati di test aumenterà (**overfitting**).

Al contrario, se il modello non è sufficiente a rappresentare la funzione che dobbiamo trovare, avremo **underfitting**.

Trovare la **complessità** ottimale del modello non è una cosa facile.

Serve quindi avere un modo di allenare il modello nella direzione giusta, riducendo la complessità.

Nel contesto del deep learning, le strategie di regolarizzazione sono basate sui **regularizing estimators**. Questo è fatto riducendo la varianza, al costo di aumentare il **bias** dell'estimatore. Un buon regolarizzatore è uno che riduce la varianza significativamente senza introdurre troppo bias.

Nel mondo reale, non abbiamo mai accesso alla vera distribuzione che genera i dati. Questo è un risultato diretto dei domini estremamente complessi con cui lavoriamo quando applichiamo algoritmi di deep learning (immagini, testo e sequenze audio).

Tutto ciò implica che controllare la complessità del modello non è una semplice questione di trovare la giusta dimensione del modello e il giusto numero di parametri. Quindi non alleniamo più modelli a livelli di complessità diversi, perché richiederebbe troppo tempo. Vogliamo forzare il modello che alleniamo a rimanere semplice.

Invece, il deep learning si basa sul trovare il miglior modello adatto come un grande modello che è stato **regolarizzato** correttamente. La regolarizzazione è un modo di minimizzare non solo la loss rispetto al training set, ma anche per gestire del **rumore** presente nei dati (variabilità dei dati).

**Regolarizzazione diretta:** introdurre conoscenze pregresse o esprimere una preferenza generica per una classe di modelli più semplice

- Cambiare i vincoli, imponendo alcune restrizioni sui valori dei parametri
- Cambiare la funzione obiettivo aggiungendo vincoli deboli che penalizzano certi valori dei parametri

**Regolarizzazione indiretta:**

- Aumento del set di dati
- Introduzione di rumore nei dati di addestramento

A volte forziamo la generalizzazione durante la fase di inferenza, ad es. "metodi di ensemble".

## Metodi di regolarizzazione diretta

### Parameters Norm Penalties

Questa è la forma di regolarizzazione più tradizionale. Limita la capacità del modello, per limitare l'overfitting. Questo viene fatto introducendo una funzione di penalità, che è pesata per  $\alpha$ , nella funzione obiettivo che diventa:

$$\tilde{J}(\theta) = J(\theta) + \alpha \Omega(\theta)$$

In linea di principio, alpha potrebbe cambiare per ogni livello.

- When the optimization procedure tries to minimize the objective function, it will also decrease some measure of the **size** of the parameters  $\theta$  (or some subset of parameters).

**Note:** usually  $\Omega$  penalizes only the weights of the affine transformation at each layer.

The bias terms in the affine transformations of deep models usually require less data to be fit and are usually left unregularized.

- Therefore, without loss of generality, we will assume we will be regularizing only the weights  $w$ .

Come possiamo scegliere il termine di penalizzazione? In teoria si può usare qualsiasi norma, e si raggiungono risultati diversi.

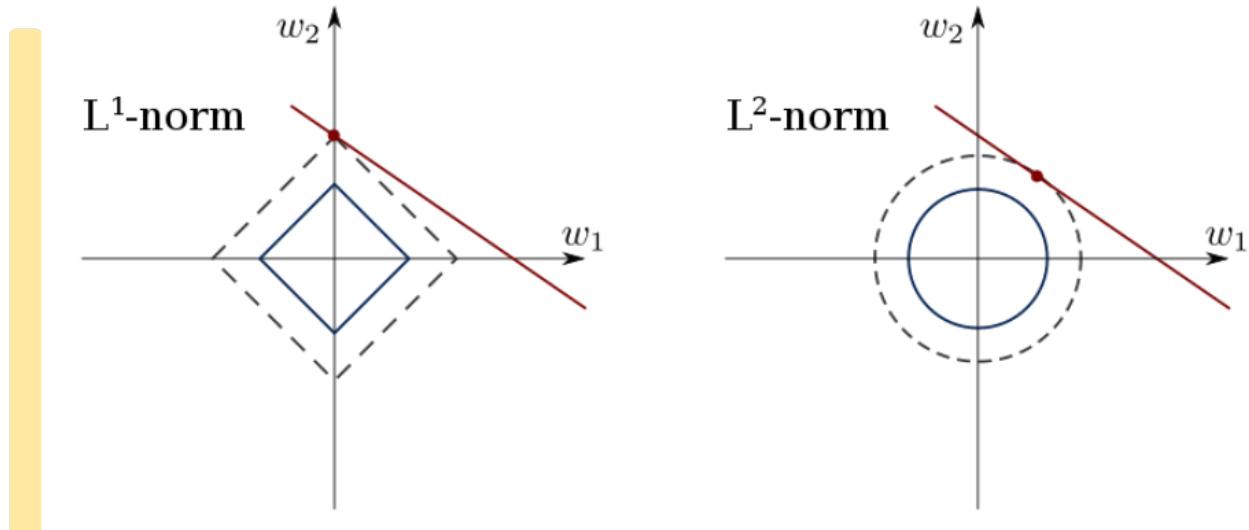
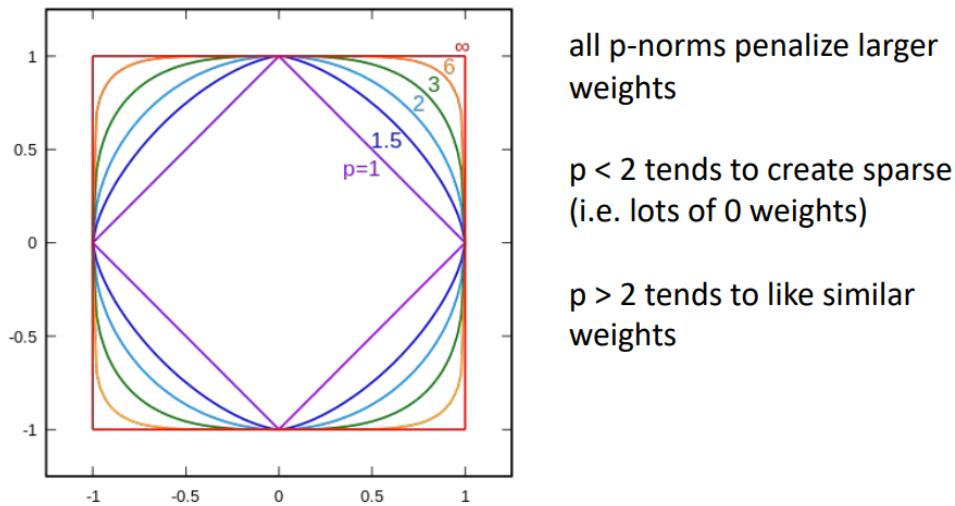
1 sum of the weights  $\Omega(w, b) = \sum_{w_j} |w_j|$

2 sum of the squared weights  $\Omega(w, b) = \sqrt{\sum_{w_j} |w_j|^2}$

Nella **somma dei pesi**, questa norma tende a mantenere il minor numero possibile di parametri  $\neq 0$ , forzandone quanti più possibile a 0, riducendo la complessità. Quindi **penalizza i pesi piccoli** di più. La **seconda norma** invece **penalizza i pesi più grandi** di più.

$$\text{p-norm} \quad \Omega(\theta) = \sqrt[p]{\sum_{w_j} |w_j|^p} = \|w\|^p$$

Esempio di un'altra norma che si potrebbe utilizzare (non è importante).



Nel primo caso quanti più pesi sono spinti a 0, nel secondo caso invece vengono ridotti tutti i pesi. ??

---

**AI:** La regolarizzazione L1 tende a produrre soluzioni sparse, ovvero soluzioni in cui molti pesi sono esattamente zero. Questo perché il diamante ha spigoli e la soluzione ottimale tende a trovarsi su uno di questi spigoli, dove uno o più pesi sono nulli.

Vantaggi:

- Selezione delle features: La sparsità indotta dalla L1 può essere interpretata come una forma di selezione automatica delle features, poiché i pesi nulli corrispondono a features che non contribuiscono alla previsione.
- Interpretabilità: Modelli con pochi pesi non nulli sono spesso più facili da interpretare.

Svantaggi:

- Non è differenziabile: La norma L1 non è differenziabile in corrispondenza dell'origine (quando tutti i pesi sono nulli), il che può rendere più complessa l'ottimizzazione.

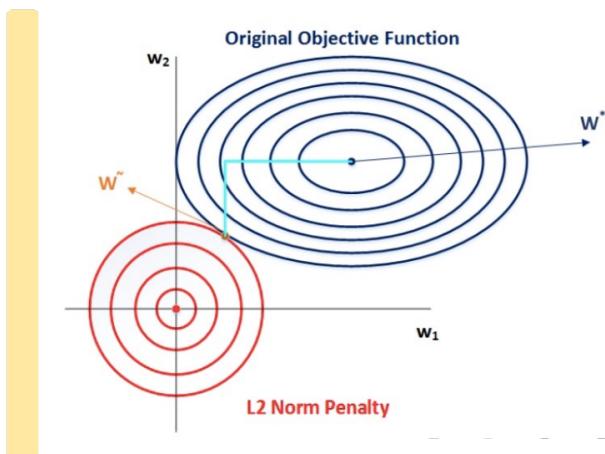
La regolarizzazione L2 tende a produrre soluzioni con pesi piccoli ma non necessariamente nulli. Questo perché il cerchio non ha spigoli e la soluzione ottimale tende a trovarsi all'interno del cerchio, lontano dall'origine.

- **Vantaggi:**

- **Stabilità:** La L2 è differenziabile ovunque, rendendo l'ottimizzazione più semplice.

- **Svantaggi:**

- **Non induce sparsità:** La L2 non produce soluzioni sparse, quindi non ha l'effetto di selezione delle features della L1.



Quindi il punto centrale blu è il minimo della funzione obiettivo.

Bisogna cercare un trade off

AI: Le curve blu rappresentano i valori della funzione obiettivo nello spazio dei pesi (parametri  $w_1 w_2$ ). Il punto blu più interno ( $w^*$ ) è il minimo di questa funzione, ossia l'insieme ottimale di pesi senza alcuna penalità.

I cerchi rossi rappresentano la penalità L2. Il centro di questi cerchi rossi è l'origine, il che significa che la penalità spinge i pesi a essere il più vicino possibile a zero. Il punto  $w^{**}$  è dove la penalità ha più effetto, spingendo i pesi verso lo zero.

Introducendo la penalità L2, l'obiettivo diventa trovare un equilibrio tra minimizzare la funzione obiettivo e minimizzare la penalità. Questo porta a una nuova soluzione (rappresentata dall'intersezione tra le curve blu e rosse), che è un compromesso tra i due obiettivi.

In pratica, questo compromesso è controllato da un parametro di regolarizzazione ( $\lambda$ ) che bilancia la funzione obiettivo e la penalità. Se  $\lambda$  è troppo grande, il modello viene regolarizzato eccessivamente (i pesi tendono a essere vicini a zero), mentre se  $\lambda$  è troppo piccolo, la regolarizzazione ha poco effetto e il modello potrebbe sovra-adattarsi ai dati.

Given the total objective function

$$\tilde{J}(\omega; X, y) = \frac{\alpha}{2} \omega^T \omega + J(\omega; X, y)$$

The gradient is:

$$\nabla_{\omega} \tilde{J}(\omega; X, y) = \alpha \omega + \nabla_{\omega} J(\omega; X, y)$$

The update rule of gradient decent using L2 (p=2)

$$\omega \leftarrow \omega - \varepsilon (\alpha \omega + \nabla_{\omega} J(\omega; X, y))$$

$$\omega \leftarrow (1 - \varepsilon \alpha) \omega - \varepsilon \nabla_{\omega} J(\omega; X, y)$$

The weights multiplicatively shrink by a constant factor at each step.

Riordinando i termini della funzione si ottiene l'ultima formula di update dei pesi.

Quindi qua anche la w precedente è penalizzata, in base alla grandezza di alpha.  
Quindi i pesi più grandi sono penalizzati di più.

---

La regolarizzazione **L2** causa l'algoritmo di learning di percepire l'input con una varianza più alta, e questo fa in modo che i pesi vengano ridotti per quelle features la quale covarianza relativa al target di output è bassa rispetto alla sua varianza.

Questo accade anche quando usiamo la regolarizzazione **L1**, ma L1 crea soluzioni che sono **sparse**, perchè forza alcuni parametri a 0, penalizzando tutti i parametri nello stesso modo, quindi i pesi più piccoli saranno i primi ad arrivare a 0. Questa proprietà della **sparsity** può essere vista come un meccanismo di **feature selection**.

---

Al posto di penalizzare le norme, potremmo vedere il problema da un punto di vista diverso.

Consider again the cost function

$$\tilde{J}(\theta) = J(\theta) + \alpha\Omega(\theta)$$

We can consider this as a function subject to constraints

$$\Omega(\theta) \leq k$$

where  $k$  is a small value

Now, we can minimize a function  $\mathcal{L}$  subject to constraints by constructing a generalized Lagrange function, consisting of the original objective function plus a set of penalties. Each penalty is a product between a coefficient, called a lagrangean multiplier, and a function representing whether the constraint is satisfied.

$$\mathcal{L}(\theta, \alpha; X, y) = J(\omega; X, y) + \alpha(\Omega(\theta) - k)$$

Vogliamo mettere un tetto massimo a questi parametri. Vogliamo che la funzione di regolarizzazione raggiunga i constraints. Questo significa che se aggiungiamo dei constraints allora ci ritroviamo con un problema a constraints non lineare (più difficile).

The Lagrangean function is,

$$\mathcal{L}(\theta, \alpha; X, y) = J(\omega; X, y) + \alpha(\Omega(\theta) - k)$$

and therefore, we need to solve:

$$\operatorname{argmin}_{\theta} \max_{\alpha \geq 0} \mathcal{L}(\theta, \alpha)$$

In order to solve this problem, we can use techniques that minimize  $J(\omega; X, y)$  and then project the solution obtained to the feasible region  $(\Omega(\theta) - k)$

Questa tecnica è usata in quei casi dove si ha "exploding gradient", con rates che vanno molto in alto e prevengono il learning. Ci sono delle tecniche che implementano questo metodo, per prevenire l'exploding gradient, limitandolo ad una regione data.

- Prevent to get stuck in local minima
  - Impose some stability on the optimization procedure
- 

## Esercizio

1. pick a model

$$y = b + \sum_{j=1}^n w_j f_j$$

2. pick a criteria to optimize (aka objective function)

$$\sum_{i=1}^n \exp(-y_i(w \cdot x_i + b)) + \frac{\lambda}{2} \|w\|^2$$

3. develop a learning algorithm

$$\operatorname{argmin}_{w,b} \sum_{i=1}^n \exp(-y_i(w \cdot x_i + b)) + \frac{\lambda}{2} \|w\|^2 \quad \text{Find } w \text{ that minimize}$$

$$\operatorname{argmin}_{w,b} \sum_{i=1}^n \exp(-y_i(w \cdot x_i + b)) + \frac{\lambda}{2} \|w\|^2$$



Loss function: penalizes examples where the prediction is different than the label

Regularizer: penalizes large weights

Key: this function is convex allowing us to use gradient descent

- pick a starting point ( $w$ )
- repeat until loss doesn't decrease in all dimensions:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)

$$w_i = w_i - \eta \frac{d}{dw_i} (\text{loss}(w) + \text{regularizer}(w, b))$$


---

$$\operatorname{argmin}_{w,b} \sum_{i=1}^n \exp(-y_i(w \cdot x_i + b)) + \frac{\lambda}{2} \|w\|^2$$

$$\frac{d}{dw_j} \text{objective} = \frac{d}{dw_j} \sum_{i=1}^n \exp(-y_i(w \cdot x_i + b)) + \frac{\lambda}{2} \|w\|^2$$

$$= - \sum_{i=1}^n y_i x_{ij} \exp(-y_i(w \cdot x_i + b)) + \lambda w_j$$


---

- pick a starting point ( $w$ )
- repeat until loss doesn't decrease in all dimensions:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)

$$w_i = w_i - \eta \frac{d}{dw_i} (\text{loss}(w) + \text{regularizer}(w, b))$$


---

$$w_j = w_j + \eta \sum_{i=1}^n y_i x_{ij} \exp(-y_i(w \cdot x_i + b)) - \eta \lambda w_j$$

$$w_j = w_j + \eta y_i x_{ij} \exp(-y_i(w \cdot x_i + b)) - \eta \lambda w_j$$

learning rate      direction to update      regularization  
 constant: how far from wrong

If  $w_j$  is positive, reduces  $w_j$   
 If  $w_j$  is negative, increases  $w_j$



moves  $w_j$  towards 0

## L1 regularization

$$w_j = w_j + \eta y_i x_{ij} \exp(-y_i(w \cdot x_i + b)) - \eta \lambda \text{sign}(w_j)$$

↑      ↑      ↑      ↑

learning rate    direction  
to update                  regularization

constant: how far from wrong

If  $w_j$  is positive, reduces by a constant  
 If  $w_j$  is negative, increases by a constant

} moves  $w_j$  towards 0  
*regardless of magnitude*

## Regularization with p-norms

**L1:**  $w_j = w_j + \eta(\text{loss\_correction} - \lambda \text{sign}(w_j))$

**L2:**  $w_j = w_j + \eta(\text{loss\_correction} - \lambda w_j)$

**Lp:**  $w_j = w_j + \eta(\text{loss\_correction} - \lambda c w_j^{p-1})$

How do higher order norms affect the weights?

L1 is popular because it tends to result in sparse solutions (i.e. lots of zero weights)  
However, it is not differentiable, so it only works for gradient descent solvers

L2 is also popular because for some loss functions, it can be solved directly

L<sub>p</sub> is less popular since they don't tend to shrink the weights enough

## Metodi indiretti di regolarizzazione

Ci sono diversi modi di regolarizzare, alcuni metodi non sono diretti.

Il modo migliore sarebbe di avere più dati di training (**data augmentation**), perché anche se la funzione si applica molto nello specifico ai dati, è comunque generalizzata. Questo metodo è molto utile se è semplice ottenere più dati, per esempio nell'image processing (per esempio basta specchiare un'immagine, non serve mettere labels sull'immagine).

Questo metodo è molto utile, però non bisogna introdurre troppo bias nei dati generati (i dati generati devono avere la stessa distribuzione dei dati reali).

Bisognerebbe quindi generare un modello con e uno senza data augmentation, per valutare qual è il migliore.

Il **noise injection** (è una forma di regolarizzazione) è un metodo che introduce un rumore con varianza infinitesimale negli input. Questo è equivalente a mettere una penalità alla norma dei pesi.

Il rumore può essere iniettato a **livelli diversi** di un modello di deep learning.

Per rumori piccoli, si può dimostrare che la minimizzazione di  $J$  con il rumore aggiunto è equivalente a minimizzare  $J$  con un termine di regolarizzazione aggiunto.

Questa forma di regolarizzazione incoraggia i parametri a muoversi verso regioni dello spazio dei parametri in cui piccole perturbazioni dei pesi hanno un'influenza relativamente piccola sull'output.

In altre parole, spinge il modello verso regioni in cui il modello è relativamente insensibile a piccole variazioni dei pesi, trovando punti che non sono

semplicemente minimi, ma minimi circondati da regioni piatte.

La maggior parte dei dataset ha un certo numero (MOLTI!) di errori nelle etichette  $y$ . Minimizzare la nostra funzione di costo su etichette errate può essere estremamente dannoso. Un modo per rimediare a questo è modellare esplicitamente il rumore sulle etichette. Questo si fa impostando una probabilità  $\epsilon$  per la quale riteniamo che le etichette siano corrette.

An example is **label smoothing**, based on a softmax with  $k$  output values where the outputs 0 and 1 are replaced by

$$\frac{\epsilon}{k-1} \text{ and } 1 - \epsilon$$

Usually, we have output vectors provided to us as

$$y_{label} = [1, 0, 0, 0, \dots, 0]$$

Softmax output is usually of the form

$$y_{out} = [0.87, 0.001, 0.04, 0.1, \dots, 0.03]$$

L'apprendimento con maximum likelihood, con un classificatore softmax e "hard targets" potrebbe effettivamente non convergere mai, poiché il softmax non può mai prevedere una probabilità esattamente pari a 0 o esattamente pari a 1, quindi continuerà a imparare pesi sempre più grandi, facendo previsioni sempre più estreme all'infinito.

Il label smoothing ha il vantaggio di prevenire l'inseguimento di probabilità rigide senza scoraggiare la corretta classificazione.

## Multitask learning

Multitask Learning is a way to improve generalization by pooling the examples arising out of several tasks.

Usually, the most common form of multitask learning is performed through an architecture which is divided to two parts:

- Task-specific parameters (which only benefit from the examples of their task to achieve good generalization).
- Generic parameters, shared across all the tasks (which benefit from the pooled data of all the tasks).

Multitask learning is a form of parameter sharing.

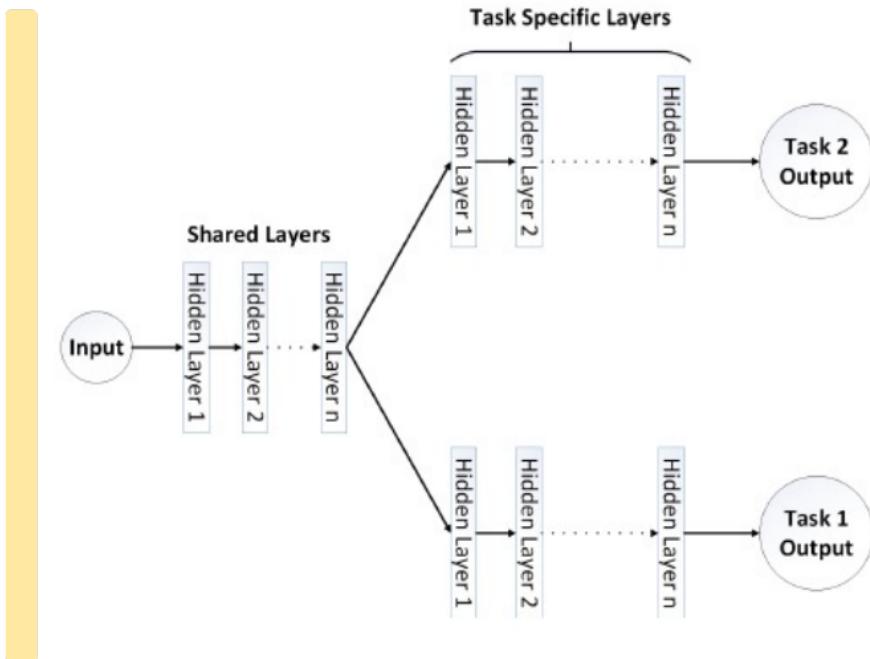
L'apprendimento multi-task è una tecnica che migliora la capacità di generalizzazione di un modello, combinando esempi provenienti da task diverse.

La forma più comune di apprendimento multi-task si basa su un'architettura divisa in due parti:

- **Parametri specifici per il compito:** questi parametri traggono beneficio solo dagli esempi del loro compito specifico per ottenere una buona generalizzazione.
- **Parametri generici condivisi:** questi parametri sono condivisi tra tutti i compiti e beneficiano dei dati combinati di tutti i compiti.

L'apprendimento multi-task è, in sostanza, una forma di condivisione dei parametri.

Visto che dobbiamo specializzare 2 task diverse, i layer in comune vengono forzati ad essere più generali.



La maggior generalizzazione è raggiunta grazie ai parametri condivisi, che forzano la generalizzazione.

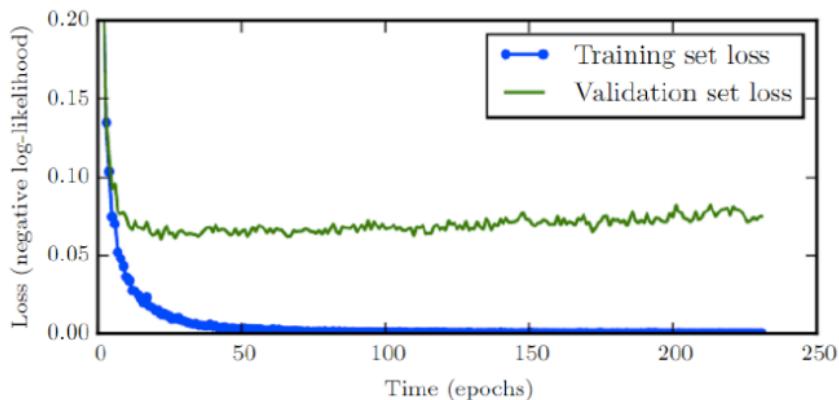
Una migliore generalizzazione può essere raggiunta grazie ai parametri condivisi, per i quali la "statistical strength" può essere notevolmente migliorata in proporzione all'aumento degli esempi per i parametri condivisi, rispetto allo scenario di modelli a singolo compito.

Il compito aggiuntivo impone vincoli ai parametri negli strati condivisi, prevenendo l'overfitting.

Un miglioramento nella generalizzazione si verifica solo quando c'è qualcosa di condiviso tra i compiti in questione.

## Early stopping

Quando addestriamo modelli di grandi dimensioni con una capacità rappresentativa sufficiente a overfittare, spesso osserviamo che l'errore sull'insieme di addestramento diminuisce costantemente nel tempo, mentre l'errore sull'insieme di convalida inizia a risalire.



Possiamo ottenere un modello con un migliore errore sul validation set, ritornando ai parametri che erano stati raggiunti nel punto del tempo con il validation set error più basso.

L'early stopping è una delle strategie di regolarizzazione più usate nel deep learning. Può essere considerato come un metodo di selezione degli iperparametri, dove il tempo di addestramento è l'iperparametro da scegliere.

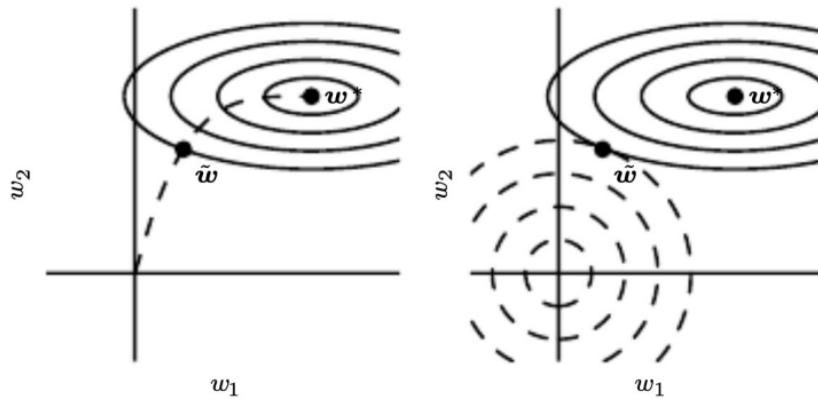
La scelta automatica del tempo di addestramento può essere effettuata con una singola run della fase di addestramento, con l'unica aggiunta della valutazione dell'errore sull'insieme di convalida ad ogni n iterazioni. Questo viene solitamente fatto su una seconda GPU.

Una seconda strategia per sfruttare completamente l'intero dataset di addestramento sarebbe:

- Applicare l'arresto precoce come descritto in precedenza.
- Continuare l'addestramento con i parametri determinati dall'arresto precoce, utilizzando i dati del validation set.

Questa strategia evita l'alto costo di riaddestrare il modello da zero, ma non si comporta bene. Poiché non abbiamo più un insieme di validazione, non possiamo sapere se l'errore di generalizzazione sta migliorando o meno.

La migliore opzione è interrompere l'addestramento quando l'errore sull'insieme di addestramento non diminuisce più significativamente.



Quest'immagine mostra perchè l'early stopping può essere visto come una tecnica di regolarizzazione. A destra c'è la tecnica diretta vista prima. A sinistra abbiamo la traiettoria del parametro, fermandosi prima possiamo raggiungere lo stesso risultato della funzione di regolarizzazione, prima di raggiungere il minimo.

## Parameter sharing

Fino ad ora, abbiamo discusso della regolarizzazione come l'aggiunta di vincoli o penalità ai parametri all'interno di una regione fissa.

Tuttavia, potremmo voler esprimere le nostre conoscenze a priori sui parametri in modo diverso. In particolare, potremmo non sapere in quale regione specifica si troveranno i parametri, ma piuttosto che ci siano delle dipendenze tra di loro.

Il tipo di dipendenza più comune è che alcuni parametri dovrebbero essere vicini tra loro.

## Parameter tying

Potremmo avere 2 modelli diversi, per diverse task (ma simili). Possiamo imporre che i pesi dei due modelli siano simili. Questo approccio viene usato per informazioni nei grafi, dove l'informazione è su nodi.

Parameter Tying refers to explicitly forcing the parameters of two models to be close to each other, through the norm penalty:

$$||w(A) - w(B)||$$

Here,  $w(A)$  refers to the weights of the first model while  $w(B)$  refers to those of the second one.

Parameter Sharing imposes much stronger assumptions on parameters through forcing the parameter sets to be equal.

Examples would be Siamese networks, convolution operators, and multitask learning.

## Bagging

Il bagging (abbreviazione di bootstrap aggregating) è una tecnica che serve a ridurre l'errore di generalizzazione combinando diversi modelli.

Il bagging è definito come segue:

- Addestra **k modelli diversi** su **k diversi sottoinsiemi** dei dati di addestramento. Questi sottoinsiemi vengono creati dal dataset originale in modo da avere lo stesso numero di esempi del dataset originale, facendo random sampling dal dataset "with replacement" (AI: La particolarità è che gli elementi vengono campionati con rimpiazzo, il che significa che lo stesso elemento può apparire più volte in un sottoinsieme e altri potrebbero non esserci affatto).
- **Fai votare tutti i modelli** sull'output per gli esempi di test.

Le tecniche che utilizzano il bagging sono chiamate **modelli ensemble**.

Il motivo per cui il bagging funziona è che diversi modelli solitamente non commettono tutti gli stessi errori sul set di test. Questo è un risultato diretto dell'addestramento su k diversi sottoinsiemi dei dati di addestramento, dove ogni sottoinsieme manca di alcuni esempi del dataset originale.

Altri fattori come differenze nell'inizializzazione casuale, selezione casuale di mini-batch, differenze negli iperparametri, o diversi risultati di implementazioni non deterministiche delle reti neurali sono spesso sufficienti per far sì che diversi membri dell'ensemble commettano errori parzialmente indipendenti.

## Ensemble models

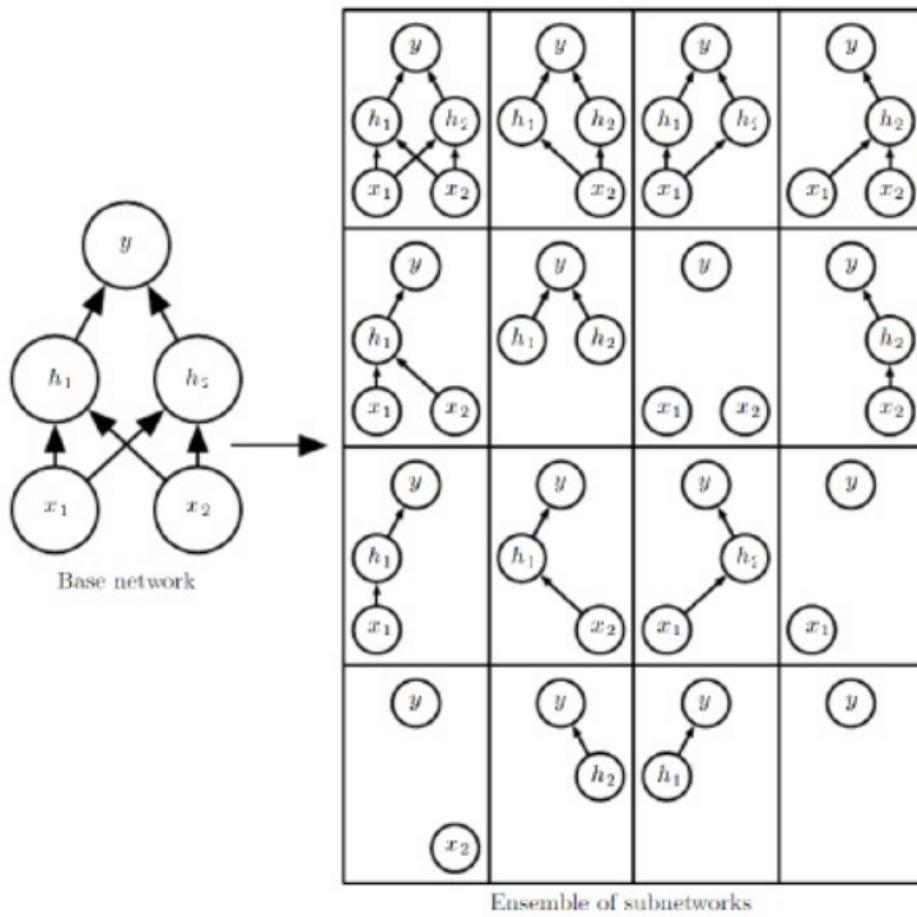
In media, l'ensemble avrà prestazioni almeno pari a quelle di uno qualsiasi dei suoi membri e, se i membri commettono errori indipendenti, l'ensemble avrà prestazioni significativamente migliori dei suoi membri.

L'unico svantaggio dei modelli ensemble è che non ci forniscono un modo scalabile per migliorare le prestazioni. Solitamente, modelli ensemble con più di 2-3 reti diventano troppo complessi da addestrare e gestire.

## Dropout

Il dropout fornisce un'approssimazione economica all'addestramento e alla valutazione di un ensemble di bagging di un numero esponenzialmente grande di reti neurali. Il dropout addestra l'ensemble costituito da tutte le sotto-reti che possono essere formate rimuovendo unità non di output da una rete di base sottostante.

Quindi si fa training tramite subnetworks (in teoria tutti quelli possibili, in realtà sono limitati in modo random).



L'immagine non è corretta, è solo un esempio di cancellazione nodi random, ovviamente ci sono dei limiti.

Per addestrare con il dropout, utilizziamo un algoritmo di apprendimento basato su minibatch che compie piccoli passi, come la discesa del gradiente stocastico. Ogni volta che carichiamo un esempio in un minibatch, campioniamo casualmente una maschera binaria diversa da applicare a tutte le unità di input e le unità nascoste nella rete. La maschera per ciascuna unità è campionata in modo indipendente dalle altre.

Tipicamente, la probabilità di includere un'unità nascosta è 0,5, mentre la probabilità di includere un'unità di input è 0,8.

Il dropout ci consente di rappresentare un numero esponenziale di modelli con una quantità gestibile di memoria, ed elimina la necessità di accumulare i voti dei modelli nella fase di inferenza.

Il dropout può essere intuitivamente spiegato come il forzare il modello a imparare con unità di input e nascoste mancanti.

Complessità dell'addestramento con dropout:

Durante l'addestramento, è necessario dividere l'output di ciascuna unità per la probabilità della maschera di dropout di quell'unità.

L'obiettivo è assicurarsi che l'input totale atteso per un'unità durante il test sia approssimativamente lo stesso dell'input totale atteso per quell'unità durante l'addestramento, anche se metà delle unità sono assenti in media durante l'addestramento.

Il dropout è computazionalmente economico. Il dropout non limita in modo significativo il tipo di modello o la procedura di addestramento che può essere utilizzata. Funziona bene con quasi qualsiasi modello che utilizza una rappresentazione distribuita e può essere addestrato con la discesa del gradiente stocastico.

Anche se il costo per passaggio nell'applicare il dropout a un modello specifico è trascurabile, il costo di utilizzo del dropout in un sistema completo può essere significativo.

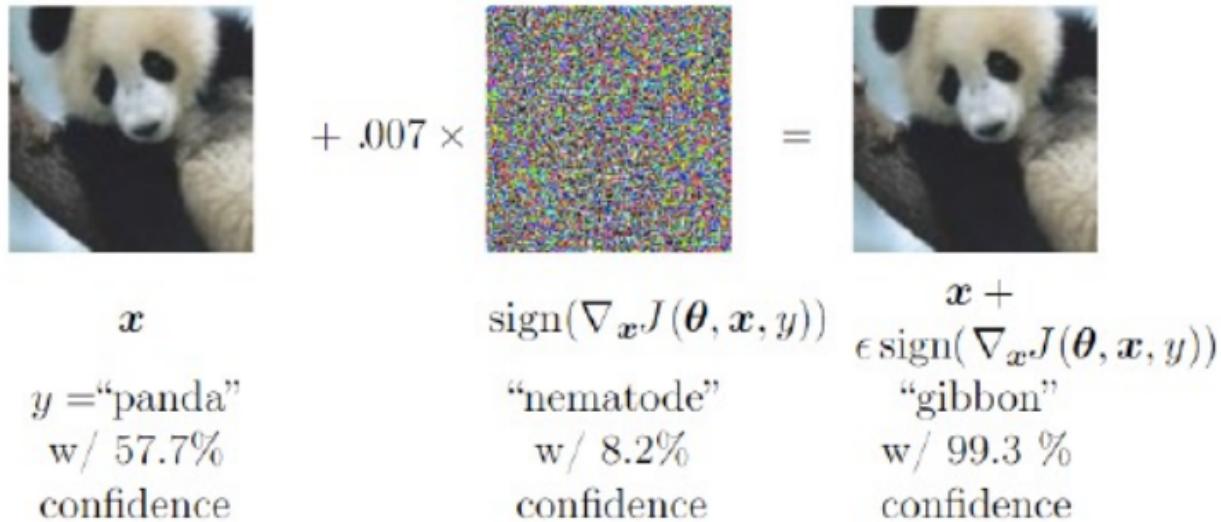
L'applicazione del dropout ci costringe indirettamente a progettare sistemi più grandi per preservare la capacità. Sistemi più grandi sono solitamente più lenti durante la fase di inferenza.

Bisogna tenere presente che per dataset molto grandi, la regolarizzazione conferisce una piccola riduzione dell'errore di generalizzazione. Il costo computazionale dell'utilizzo del dropout e di modelli più grandi potrebbe superare il beneficio della regolarizzazione.

## Adversarial training

Si generano degli esempi che fanno fallire la rete. Questi vanno trovati e vanno inseriti nel training set, per generalizzare il modello. In molti casi, l'esempio modificato è simile a quello originale, una persona non può vederne la differenza.

Per costruire un adversarial example bisogna risolvere un problema di ottimizzazione, e trovare il minimo parametro che fa fallire la predizione. Questo non è semplice.



In questo esempio viene aggiunta noise, e il modello fallisce sulla nuova immagine.

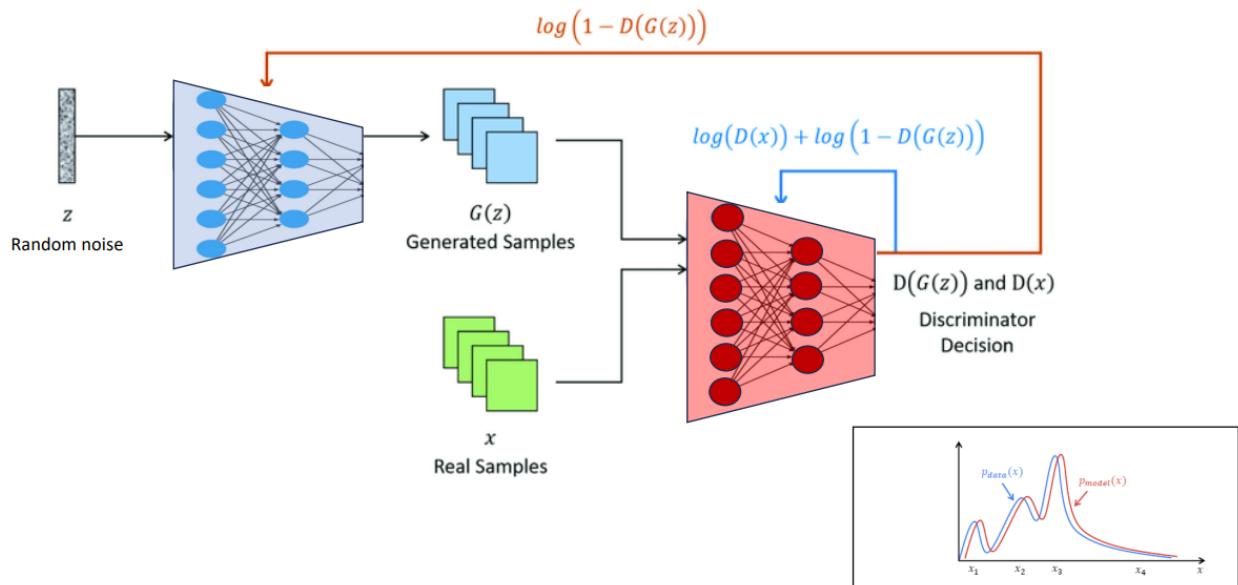
## Generative adversarial models

Sono modelli di deep learning generativi che imparano la distribuzione dei dati, usando un "zero-sum game" tra due reti neurali che competono: **discriminator (D)** e **generator (G)**.

Durante il training il discriminator deve migliorare nel capire gli elementi finti del generator, e il generator deve diventare migliore a generare esempi simili.

GANs game:

$$\min_G \max_D V_{GAN}(D, G) = \underbrace{\mathbb{E}_{x \sim p_{data}(x)} [\log D(x)]}_{\text{real samples}} + \underbrace{\mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]}_{\text{generated samples}}$$



**In a neural network the nonlinearity causes the most interesting loss function to become non convex**

### ⌚ Results

You answered:

True

False



The correct answer was

True

Le immagini delle loss functions hanno minimi e massimi locali, non è convessa

**The loss function produces a numerical score that also depends on the set of parameters  $\theta$  which characterizes the FFN model**



Congratulations!



True



False

**Regularization functions are added to the loss functions to reduce their training error**

True

False

Il training error è solo sugli esempi di training, se aggiungiamo questo termine è per ottimizzare l'errore sugli esempi non visti nel training set, quindi per non fare

overfitting.

**The gradient can be estimated through an iterative procedure that uses at each iteration only a sample of training examples**

True

False

## Which of the following statements are true ?

 You can select multiple choices

When using SGD with mini-batches the model updates do not depend on the number of training examples

When using SGD with mini-batches the number of updates to reach convergence does not depend on the number of training examples

Once the SGD converges it is still useful to add more training examples

The choice of cost functions is tightly coupled with the choice of the output unit

risposta: 1 e 4. 1: Quando abbiamo tanti esempi, se usiamo mini batch potremmo convergere prima di usare tutti gli esempi, l'update dipende solo dal mini batch e non rispetto a tutto il training set perchè l'update accade rispetto al mini batch.

2: la risposta non è chiara, dipende dal tipo di dati di training che abbiamo. Non ho capito il resto della spiegazione

3: se il gradiente ha raggiunto la convergenza non abbiamo bisogno di altri esempi