INDICE

12 Model Compression

- Pruning NN
- Lottery Ticket Hypothesis
- Weight Quantization

13 Meta Learning

- Meta Learning Problem, Few Shot
- Supervised vs meta learning
- Metric-Based Meta Learning
- Model-Based Meta learning
- Optimization-Based Meta learning
- Learning to Learn
- MAML

14 RAG

15 Noisy Label

CAPITOLO 1

INTRODUCTION

Mean Squared Error Used for regression

$$L(\theta) = \sum_{i=1}^{I} \left[f(x,\theta) - y_i \right]^2 = \sum_{i=1}^{I} \left[(\theta_0 + \theta_1 x_i) - y_i \right]^2$$

Binary Cross Entropy : For classification Problems

$$L(\theta) = \sum_{i=0}^{I} -(1-y_i) \log\left[1 - \text{sig}\left(f(x,\theta)\right)\right] - y_i \log\left[\text{sig}\left(f(x,\theta)\right)\right]$$

$\text{sig}(h) = \frac{1}{1+e^{-h}}$

Optimization of BCE, computation of gradient

$h = w^T x \qquad z = \text{sig}(h) = \frac{1}{1-e^{-h}}$

$$J(w) = -\left(y \log(z) + (1-y)\log(1-z)\right)$$

The Chain Rule

$$\frac{\partial J(w)}{\partial w} = \frac{\partial J(w)}{\partial z} \frac{\partial z}{\partial h} \frac{\partial h}{\partial w}$$

$$\text{Gradient Descent} = w - \alpha \frac{\partial J(w)}{\partial w}$$

Discriminative vs Generative

$y = f(x,\theta)$

$x = g(y,\theta)$

In this setting the inference to get y would require the inversion of the equation

$y = g^{-1}(x,\theta)$

CAPITOLO 2

SHALLOW AND DEEP NEURAL NETWORKS

- Shallow Network
- Universal Approximation Theorem
- Activation Functions
- Deep Networks
- UAT : Depth Version

BASICS OF NEURAL NETWORKS

$f[x, \phi]$ (ϕ: parameters, x: input) x input scalar and y (output) is a scalar

ϕ is composed of 10 parameters $(\phi_0, \phi_1, \phi_2, \phi_3, \theta_{10}, \theta_{11}, \theta_{20}, \theta_{21}, \theta_{30}, \theta_{31})$

Then we can express $y = f(x, \phi)$

Now is a multivariate (3) linear function

$= \phi_0 + \phi_1 \square + \phi_2 \square + \phi_3 \square$

By using the parameters $= \phi_0 + \phi_1 a(\square) + \phi_2 a(\square) + \phi_3 a(\square)$

LINEAR LAYER — ACT. FUNCTION

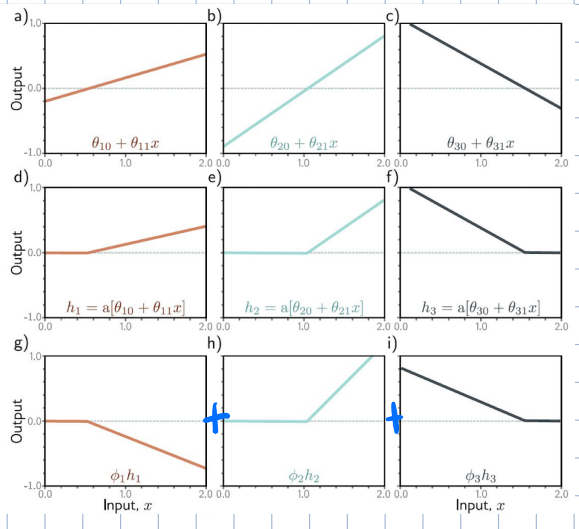
$= \phi_0 + \phi_1 a(\theta_{10} + \theta_{11} x) + \phi_2 a(\theta_{20} + \theta_{21} x) + \phi_3 a(\theta_{30} + \theta_{31} x)$

Now let's see this function by the use of hidden units

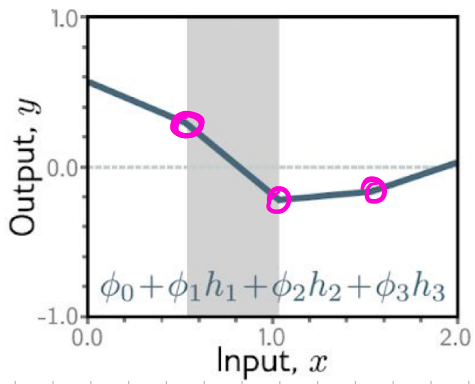
$= \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$

$h_i = [\theta_{i0} + \theta_{i1} x]$

Let's take $a = \text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$



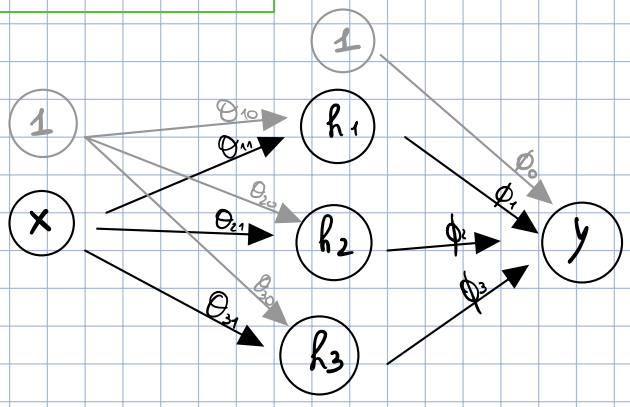
Sum of the three functions



3 hidden units = 3+1 splits

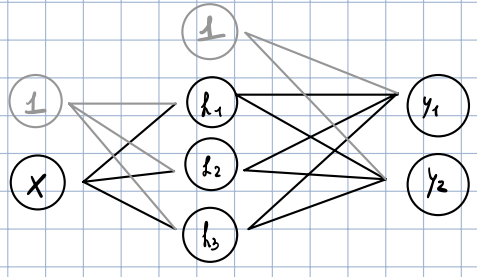
Now let's see the function with the neuron scheme

$$y = f(x, \phi) =$$
$$= \phi_0 + \phi_1\, a\left(\theta_{10} + \theta_{11} x\right) + \phi_2\, a\left(\theta_{20} + \theta_{21} x\right) + \phi_3\, a\left(\theta_{30} + \theta_{31} x\right)$$



Considering a multivariate output function

$$y_1 = \phi_{10} + \phi_{11}\, a\left(\theta_{110} + \theta_{111} x\right) + \phi_{12}\, a\left(\theta_{120} + \theta_{121} x\right) + \phi_{13}\, a\left(\theta_{130} + \theta_{131} x\right)$$
$$y_2 = \phi_{20} + \phi_{21}\, a\left(\theta_{210} + \theta_{211} x\right) + \phi_{22}\, a\left(\theta_{220} + \theta_{221} x\right) + \phi_{23}\, a\left(\theta_{230} + \theta_{231} x\right)$$

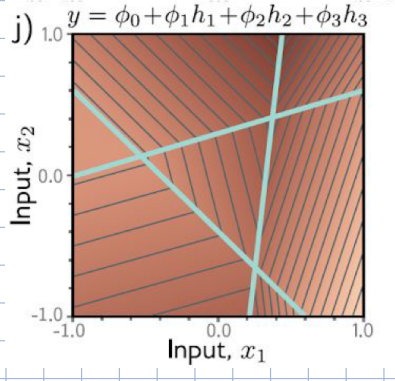


Multivariate Input function

Now $x = [x_1, x_2]$

then $y = \phi_0 + \phi_1\, a\left(\theta_{10} + \theta_{11} x_1 + \theta_{12} x_2\right) + \phi_{12}\, a\left(\theta_{20} + \theta_{21} x_1 + \theta_{22} x_2\right) + \phi_{13}\, a\left(\theta_{30} + \theta_{31} x_1 + \theta_{32} x_2\right)$

linear function of the multivariate input $\} h_i$



3 hidden units (3 lines)

2 input (2 dimension - plot)

Multivariate input and output

$x \in \mathbb{R}^{d_i}$ $\quad y \in \mathbb{R}^{d_o}$ $\quad h \in \mathbb{R}^{d}$ $\quad h_d = a\left(\theta_{d0} + \sum_i^{d_i} \theta_{di} x_i\right)$ $\quad y = \phi_0 + \sum_i^{d_o} \phi_i h_i$

UNIVERSAL APPROXIMATION THEOREM

D hidden units $h_1 \dots h_D$ $\quad h_i = a\left(\theta_{i0} + \theta_{i1} x\right)$

The non linear function will be $y = \underline{\theta_0} + \sum_i^D \underline{\theta_i}\ \underline{h_i}$

θ_0: bias; θ_i: weight; h_i: non linearity

With D hidden units we have D+1 split

Make $D \to \infty$ and you can approximate (any) complex functions

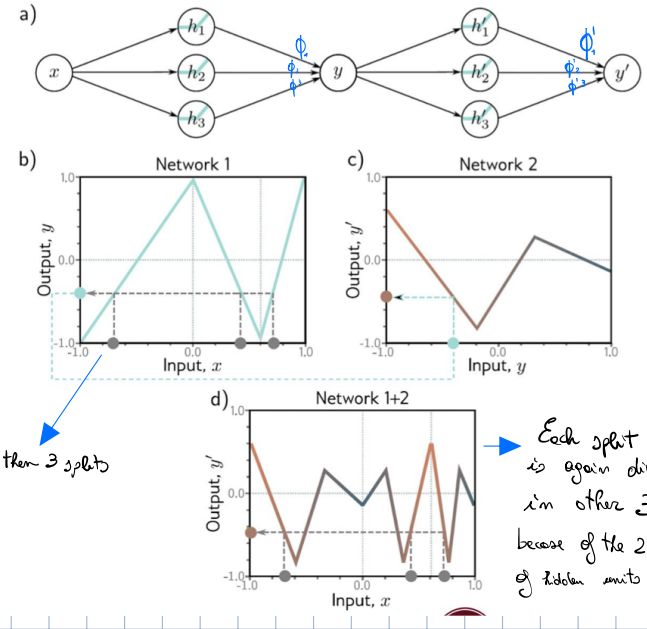
THEOREM: For any continuous function ∃ a shallow NN that approximate that function

DEEP NEURAL NETWORKS

$y = \phi_0 + \phi_1 a\left(\theta_{10} + \theta_{11} x\right) + \phi_2 a\left(\theta_{20} + \theta_{21} x\right) \dots$

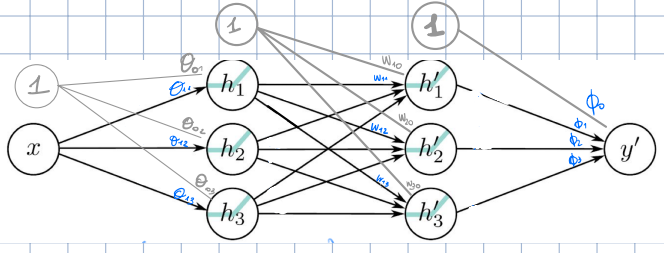
$y' = \phi'_0 \dots$

Case 1



h then 3 splits

Each split is again divided in other 3 splits because of the 2° layer of hidden units



CASE 2 : MLP

$y' = \phi_0 + \phi_1 h'_1 + \phi_2 h'_2 + \phi_2 h'_3$

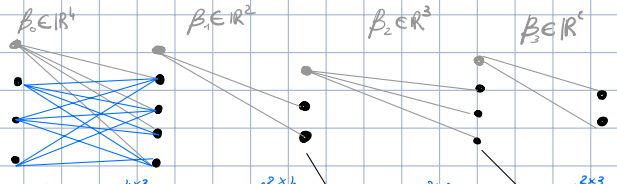
$= \phi_1 a\left(w_{10} + w_{11} h_1 + w_{12} h_2 + w_{13} h_3\right) + \phi_2 a\left(w_{20} + w_{21} h_2 + w_{22} h_2 + w_{23} h_3\right) + \phi_3\left(w_{30} + w_{31} h_1 + w_{32} h_2 + w_{33} h_3\right)$

$= \phi_1 a\left(w_{10} + w_{11} a\left(\theta_{10} + \theta_{11} x\right) + w_{12} a\left(\theta_{20} + \theta_{21} x\right) + w_{13} a\left(\theta_{30} + \theta_{31} x\right)\right)$

$+ \phi_2 a\left(w_{20} + w_{21} a\left(\theta_{20} + \theta_{21} x\right) + \dots\right)$

Matrix Notation

$x \in \mathbb{R}^3$ $\quad y \in \mathbb{R}^2$ $\quad H = \{H_1, H_2, H_3\}$ $\quad A_1 \in \mathbb{R}^4$, $A_2 \in \mathbb{R}^2$, $H_3 \in \mathbb{R}^3$



$$h_1 = [\beta_0 + W_0 x] \quad h_2 = [\beta_1 + W_1 h_1] \quad h_3 = [\beta_2 + W_2 h_2] \quad y = [\beta_3 + W_3 h_3]$$

Exercise MLP Parameters Count

$x \in \mathbb{R}^{6220800}$ $\quad D_1 = 512$ $\quad D_2 = 1024$ $\quad D_3 = 512$ $\quad y \in \mathbb{R}^1$

$$\#\text{ parameters} = (\beta_0 + W_0) + (\beta_1 + W_1) + (\beta_2 W_3) + (\beta_3 W_3)$$

$$= (512 + (512 \cdot 6220800)) + (1024 + (1024 \cdot 512)) + (512 + (512 \cdot 1024)) + (1 + 512) = \_ \_$$

CAPITOLO 3

TRAINING & LOSS FUNCTION

- Loss Function, Maximum Likelihood
- Log Likelihood, Negative Log Likelihood
- Recipe for Loss
  - Least Square Loss
  - Heteroscedastic vs Homoscedastic
- Multiclass Classification, Multiclass Cross Entropy
- Cross Entropy

Loss

Is a function that returns a measure of the mismatch between prediction of the model and ground truth

Training objective is to find the best parameters of $f(x, \theta)$, minimizing the mismatch

$$\tilde{\theta} = \underset{\phi}{\operatorname{argmin}}\ L(\phi)$$

MAXIMUM LIKELIHOOD

We can see the ML model as a method to omit a conditional probability over the possible output, given the x.

$P(y|x)$, then we have to consider the prob. to get an output y given the input and the parameter of the model

$$P_z\{y|x;\theta\} \rightarrow \text{Likelihood}$$

What the model should do is to find the parameters θ^* such that the likelihood is maximum: the combined probability over all the dataset is maximum

Maximum Likelihood Criterion

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \left(\prod_{i=1}^{D} P_z(y_i | f(x, \theta)) \right)$$

→ the model will generate an output probability for each possible output value given each input x in D

Maximum Log Likelihood (Transform Π in Σ)

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \left[\sum_{i=1}^{D} \log \left(P_z(y_i | f(x, \theta)) \right) \right]$$

Negative Log Likelihood (Convert the problem into minimization problem)

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \left[- \sum_{i=1}^{D} \log \left(P_z(y_i | f(x_i, \theta)) \right) \right] = \underset{\theta}{\operatorname{argmin}}\ L(\theta)$$

Inference (Return y with highest prob. according to the model) $\hat{y} = \underset{y}{\operatorname{argmax}}\ P_z(y | x, \hat{\theta})$

BUILD A LOSS in 4 steps

1. Choose a prob. distribution $P_Z(y|\theta)$ θ parameters of the distribution
2. Set the ML model $g(x,\phi)$ ϕ parameters of the model predict $\theta = g(x,\phi)$
   and $P_Z(y|\theta) = P(y|g(x,\phi))$
3. Train the model finding ϕ such that minimize the loss (the NLL Loss)
   $$\hat{\phi} = \underset{\phi}{\operatorname{argmin}} \left[-\sum_{i=1}^{D} \log\left(P(y_i|g(x,\phi))\right)\right]$$
4. Inference return $P_Z(y|g(x,\phi))$ or $\underset{y}{\operatorname{argmax}}\, P_Z(y|g(x,\phi))$

DERIVATION OF THE MSE LOSS

1. Consider Gaussian distribution with μ, σ^2 $P_Z(y|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) =$
2. Set 2 ML models compute both μ, σ^2
   then $\mu = g_1[x,\phi]$ $\sigma^2 = g_2[x,\phi]^2$
3. $$\hat{\phi} = \underset{\phi,\sigma^2}{\operatorname{argmin}} \left[-\sum_{i=1}^{D} \log\left[\frac{1}{\sqrt{2\pi g_2(x|\phi)^2}} \exp\left(-\frac{(y-g(x,\phi))^2}{2 g_2(x|\phi)^2}\right)\right]\right]$$

MSE Loss
$$\underset{\phi}{\operatorname{argmin}} \left[\sum_{i=1}^{D} (y_i - g(x|\phi)^2)\right]$$

or using one model to get both (set as differentiable also σ^2) $\hat{\phi} = \underset{\phi,\sigma^2}{\operatorname{argmin}} \left[-\sum_{i=1}^{D} \log\left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-g(x,\phi))^2}{2\sigma^2}\right)\right]\right]$

BINARY CROSS ENTROPY

$$P_Z(y|\lambda) = \begin{cases} 1-\lambda & y=0 \\ \lambda & y=1 \end{cases} \longrightarrow P_Z(y|\lambda) = (1-\lambda)^{1-y} \cdot \lambda^y$$

1. Choose prob distribution Bernoulli
2. Set $\lambda = g(x,\phi)$
3. Likelihood $\rightarrow$ $P_Z(y|x) = \left(1 - \operatorname{sig}\left(g(x,\phi)\right)\right)^{1-y} \cdot \operatorname{sig}\left(g(x,\phi)\right)^y$

Negative log likelihood
$$L(\phi) = \sum_{i=1}^{D} -(1-y_i)\log\left(\operatorname{sig}\left(g(x,\phi)\right)\right) - y_i \log\left(\operatorname{sig}\left(g(x_i,\phi)\right)\right)$$

MULTICLASS CLASSIFICATION

1. $y \in \{1 \dots K\}$ there is a prob. for each class, then K prob.
   $P_Z(y=k) = \lambda_k$
   λ_i has to be in $[0,1)$ and $\sum_{i=1}^{K} \lambda_i = 1$
2. We set λ_i $i=1\dots K$ $= g(x,\phi)$
   Set the constraint using $\operatorname{softmax}_k(z) = \frac{\exp(z_k)}{\sum_{k'=}^{K} \exp(z_{k'})}$
3. NLLLoss
   Likelihood $P_Z(y=k|x) = \operatorname{softmax}\left(g(x,\phi)\right)$

NLL $L(\phi) = -\sum_{i=1}^{D} \log\left[\operatorname{softmax}_{y_i}\left(g(x,\phi)\right)\right] =$
$$-\sum_{i=1}^{D} \left(g_{y_i}(x_i,\phi) - \log\left(\sum_{k'=1}^{K} \exp\left[g_{k'}(x_i,\phi)\right]\right)\right)$$

CAPITOLO 4

MODEL - TRAINING

- Gradient Descent
- Gabor Model and SGD
  - SGD Scheduler LR, Momentum, Nesterov Momentum, Adam
- Backpropagation
- Computation Graph
- Parameter Initialization
- Vanishing & Exploding gradient

GRADIENT DESCENT

Find parameters that leads to lower loss values

① Derivative of loss wrt ϕ

$$L(\phi) = \sum_{i=1}^{D} \ell_i \qquad \frac{\partial L}{\partial \phi} = \begin{bmatrix} \frac{\partial L}{\partial \phi_0} \\ \vdots \\ \frac{\partial L}{\partial_N} \end{bmatrix}$$

② Update the parameters

learning rate

$$\phi = \phi - \alpha \frac{\partial L}{\partial \phi}$$

Linear regression

$$y = f(x, \phi) = \phi_0 + \phi_1 X \qquad Loss(\phi) = \sum_{i=1}^{D} \ell_i = \sum_{i=1}^{D} (f(x, \phi) - y_i)^2 = \sum_{i=1}^{D} (\phi_0 + \phi_1 x - y_i)^2$$

Derivative of the loss

$$\frac{\partial L}{\partial \phi} = \frac{\partial}{\partial \phi} \sum_{i=1}^{D} \ell_i = \sum_{i=1}^{D} \frac{\partial \ell_i}{\partial \phi}$$

this means $\frac{\partial \ell_i}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2 x_i (\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$

STOCHASTIC GRADIENT DESCENT

Compute the gradient descent on a minibatch

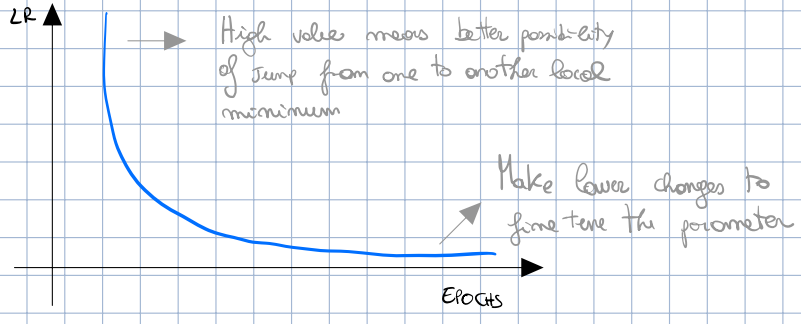
Pros
- Less expensive
- Escape local minima
- Allows generalization over unseen data

Update Rule

$$\phi_{t+1} = \phi_t - \alpha \sum_{i=1}^{B_t} \frac{\partial \ell_i [\phi_t]}{\partial \phi}$$

Batch instead that the entire dataset

SGD and Scheduler of LR



Remember LR $= \alpha$ $\quad \phi_{t+1} = \phi_t - \alpha \cdot \frac{\partial L}{\partial \phi_t}$

SGD with MOMENTUM

Effect: Smooth out the trajectory to the minimum and makes the convergence fast

Add a Momentum Term m instead that ∂L

The new update rule

$$\phi_{t+1} = \phi_t - \alpha \boxed{m_{t+1}}$$

the overrall direction depends from the previous momentum

$$m_{t+1} = \beta m_t + (1-\beta) \sum_{i=1}^{B_t} \frac{\partial \ell_i [\phi_t]}{\partial \phi}$$

$\beta \in [0,1)$ weight that gives more power to the previous or current update rule

Gradient Computed on the current batch

SGD NESTEROU ACCELERATED MOMENTUM

$$\phi_{t+1} = \phi_t - \alpha m_{t+1} \qquad m_{t+1} = \beta m_t + (1-\beta) \sum_{i=1}^{B_t} \frac{\partial \ell_i [\phi_t - \alpha m_t]}{\partial \phi}$$

Effect provide a better path to reach the minimum

SGD - ADAM

Problem until now:
- big gradient lead to big parameters update
- choose a LR that makes the model stable is difficult

Solution 1:
Normalize the momentum by the scale of the gradient to avoid huge updates, then:

$$m_{t+1} = \frac{\partial L(\phi_t)}{\partial \phi} \qquad v_{t+1} = \left(\frac{\partial L(\phi_t)}{\partial \phi} \right)^2$$

$$\phi_{t+1} = \phi_t - \alpha \frac{m_{t+1}}{\sqrt{v_{t+1}} + \varepsilon}$$

Adam: Have an even smoother trajectory

Convergence

$$m_{t+1} = \beta \cdot m_t + (1-\beta) \frac{\partial L(\phi_t)}{\partial \phi}$$

$$v_{t+1} = \gamma v_t + (1-\gamma) \left(\frac{\partial L(\phi_t)}{\partial \phi} \right)^2$$

At each timestep m_{t+1} is $\tilde{m}_{t+1} = \frac{m_{t+1}}{1-\beta^{t+1}}$

v_{t+1} is $\tilde{v}_{t+1} = \frac{v_{t+1}}{1-\gamma^{t+1}}$

Update Rule $\phi_{t+1} = \phi_t - \alpha \frac{\tilde{m}_{t+1}}{\sqrt{\tilde{v}_{t+1}} + \varepsilon}$

FITTING THE MODEL WITH BACK PROPAGATION

Let's consider

X : Multivariate input $(x_1 \dots x_m) \in \mathbb{R}^m$

ϕ : Parameters

h_1, h_2, h_3 Hidden layers

$$h_1 = a(\beta_0 + \phi_0 X)$$

$$h_2 = a(\beta_1 + \phi_1 h_1)$$

$$h_3 = a(\beta_2 + \phi_2 h_2)$$

$$f(x, \phi) = \beta + \phi_3 h_3$$

Loss for each sample ℓ_i

Loss for the entire dataset $L(\phi) = \sum_{i=1}^{n} \ell_i$

Parameter update using SGD $\phi_{t+1} = \phi_t - \alpha \sum_{i=1}^{B_t} \frac{\partial \ell_i(\phi_t)}{\partial \phi}$

Computing the derivatives $\frac{\partial \ell_i}{\partial \beta_k}$ and $\frac{\partial \ell_i}{\partial \phi_k}$

This algorithm is called **Back propagation**

It includes a

→ forward pass

← A backward pass that uses the derivatives to update the weights

An Example 8 parameter for a function $f(x, \phi)$

$\phi = \{\beta_0, \omega_0, \beta_1, \omega_1, \beta_2, \omega_2, \beta_3, \omega_3\}$ 3 non linear functions sin, exp, cos

$$f[x, \phi] = \beta_3 + \omega_3 \cdot \cos\left[\beta_2 + \omega_2 \cdot \exp\left[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x]\right]\right] \qquad \ell_i = (f[x_i, \phi] - y_i)^2$$

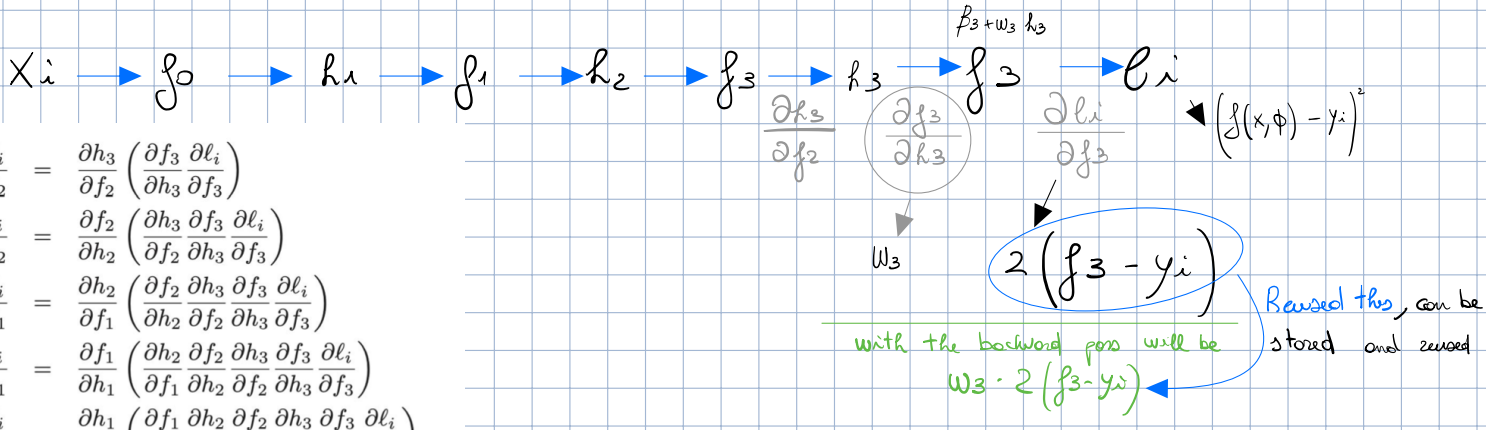
compute $\frac{\partial \ell_i}{\partial \beta_0, \partial \beta_1, \partial \beta_2, \partial \beta_3}$ and $\frac{\partial \ell_i}{\partial \omega_0, \partial \omega_1, \partial \omega_2, \partial \omega_3}$

$$\frac{\partial \ell_i}{\partial \omega_0} = \frac{\partial \ell_i}{\partial \omega_0} = -2\left(\beta_3 + \omega_3 \cdot \cos\left[\beta_2 + \omega_2 \cdot \exp\left[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x_i]\right]\right] - y_i\right) \cdot \omega_1\omega_2\omega_3 \cdot x_i \cdot \cos[\beta_0 + \omega_0 \cdot x_i] \cdot \exp\left[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x_i]\right] \cdot \sin\left[\beta_2 + \omega_2 \cdot \exp\left[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x_i]\right]\right].$$

Computing derivatives you face a lot redundant computation

COMPUTATION GRAPH

Avoid the redundant computation in the computation of derivative using a graph



$$\frac{\partial \ell_i}{\partial f_2} = \frac{\partial h_3}{\partial f_2}\left(\frac{\partial f_3}{\partial h_3}\frac{\partial \ell_i}{\partial f_3}\right)$$
$$\frac{\partial \ell_i}{\partial h_2} = \frac{\partial f_2}{\partial h_2}\left(\frac{\partial h_3}{\partial f_2}\frac{\partial f_3}{\partial h_3}\frac{\partial \ell_i}{\partial f_3}\right)$$
$$\frac{\partial \ell_i}{\partial f_1} = \frac{\partial h_2}{\partial f_1}\left(\frac{\partial f_2}{\partial h_2}\frac{\partial h_3}{\partial f_2}\frac{\partial f_3}{\partial h_3}\frac{\partial \ell_i}{\partial f_3}\right)$$
$$\frac{\partial \ell_i}{\partial h_1} = \frac{\partial f_1}{\partial h_1}\left(\frac{\partial h_2}{\partial f_1}\frac{\partial f_2}{\partial h_2}\frac{\partial h_3}{\partial f_2}\frac{\partial f_3}{\partial h_3}\frac{\partial \ell_i}{\partial f_3}\right)$$
$$\frac{\partial \ell_i}{\partial f_0} = \frac{\partial h_1}{\partial f_0}\left(\frac{\partial f_1}{\partial h_1}\frac{\partial h_2}{\partial f_1}\frac{\partial f_2}{\partial h_2}\frac{\partial h_3}{\partial f_2}\frac{\partial f_3}{\partial h_3}\frac{\partial \ell_i}{\partial f_3}\right).$$

PARAMETER INITIALIZATION

Depend from where you start from a place where the variance of the params is small to do not oscillate too much or to low

Consider $f_k = \beta_k + \Omega_k h_k$

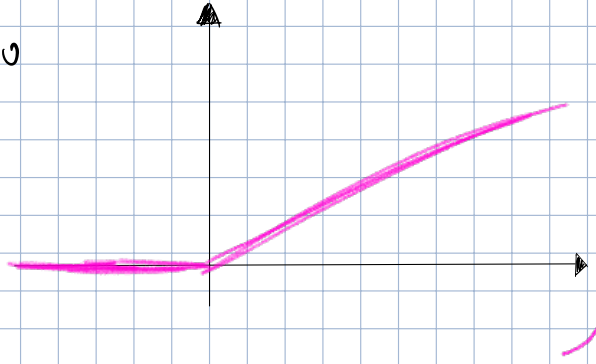
$= \beta_k + \Omega_k a(f_{k-1})$

With Ω very small variance there will be no update.

IMPOSE SOME VARIANCE : a good balance

Ω initialized $\in N(\mu, \sigma^2)$

Because RELU



→ Going through layers becomes much smaller

SMALL WEIGHTS leads to VANISHING GRADIENT

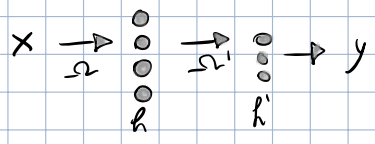
High variance σ^2 = Big Weights, huge jump in update this leads to EXPLODING GRADIENTS

VARIANCE CONSTRAINT : He initialization

$$\sigma^2 \Omega = \frac{2}{D_h}$$

$$\sigma^2 \Omega' = \frac{2}{D_{h'}}$$

If we have 2 adjacent hidden layer



Consider on initialize.

the total initialization

$$\sigma^2 \Omega = 4 \left(D_h + D_{h'} \right)$$

CAPITOLO 5

PERFORMANCE & REGOLARIZATION

- Overfitting
- Regolarizz
  - Explicit
    - As a prior
    - L2
  - Early Stopping
  - Dropout
  - Data Augmentation

SOURCE OF ERRORS

NOISE
Noise on the data

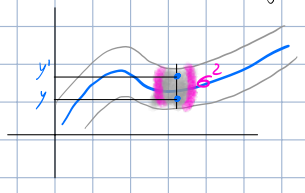
BIAS
Choosen model not flexible/powerful enough for learning the true function

VARIANCE
Variance in the learning process leads to different local minimum. Non enough variant dataset will not be a good set for the model to learn the ground truth distribution

$$\mathbb{E}_{\mathcal{D}}\Big[\mathbb{E}_y[L[x]]\Big] = \underbrace{\mathbb{E}_{\mathcal{D}}\Big[\big(\mathrm{f}[x,\phi[\mathcal{D}]] - \mathrm{f}_\mu[x]\big)^2\Big]}_{\text{variance}} + \underbrace{\big(\mathrm{f}_\mu[x] - \mu[x]\big)^2}_{\text{bias}} + \underbrace{\sigma^2}_{\text{noise}}.$$

NOISE, BIAS, VARIANCE, MATEMATHICAL PROOF

Consider 1D Regression problem. Data x have noise with variance σ^2.
For each x there might be multiple y value. We then consider a probability for x to have a certain y



$P(y|x)$. The expected mean for x: $y[x] = \mathbb{E}_y[y[x]] = \int y[x] \cdot P_x(y|x)\, dy$

fixed noise $\sigma^2 = \mathbb{E}_y\Big[\big(y[x] - y[x]\big)^2\Big]$

REDUCE ERRORS

NOISE:
Nothing to do.
Noise cannot be removed, is intrinsic of the data generation process

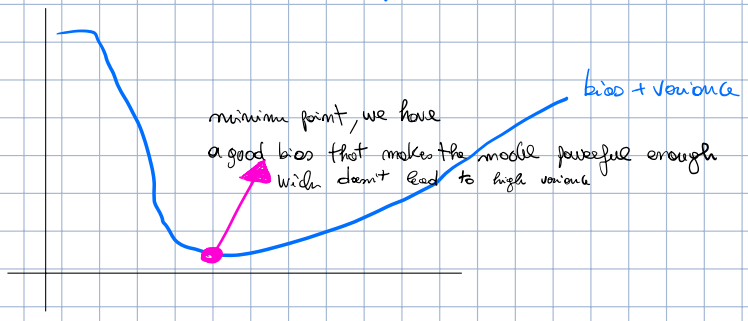
REDUCE THE VARIANCE
- Add data to learn from makes the model more confident on the prediction that is giving. Less data, more uncertainty.
- Noise is symmetric, add data makes the model even more robust wrt data noise

THE BIAS-VARIANCE TRADE OFF
Choosen a certain type of model to fit on training data, we are imposing a bias.
We are essentially limiting the flexibility of the model. To reduce the bias we can use a more powerful model.
With a lot of sample the model will try to get the lowest error for each of them.

Simple Model, biased
small variance

Increasing the picking
We have many more functions to pick from = We are increasing the variance

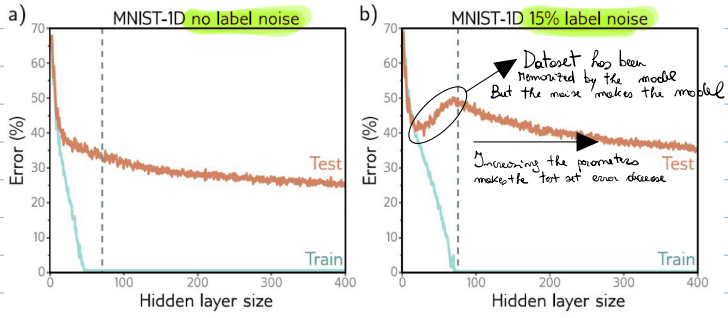
The capacity issue pick: How much data to use to fit the model?



DOUBLE DESCENT

The double descent

1. Training, the model grows
2. Adding parameters, the model memorize the data, but make mistake because of the noise
3. Mistake grows
4. Still add parameter
5. Mistake decrease



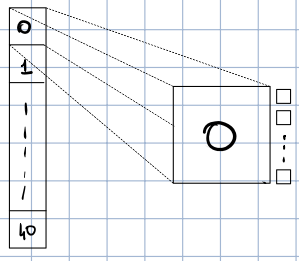
mistake on the test set which does not contain the noise

COURSE OF DIMENSIONALITY

In low dimensionality is easy to know if two object are close or far
Increasing the dimension everything starts to get complicated

Add hidden unit let the model to be powerful.

BUT still, we have to consider high dimensional data (40 dim)

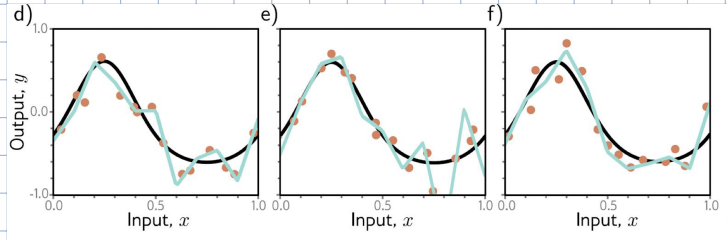


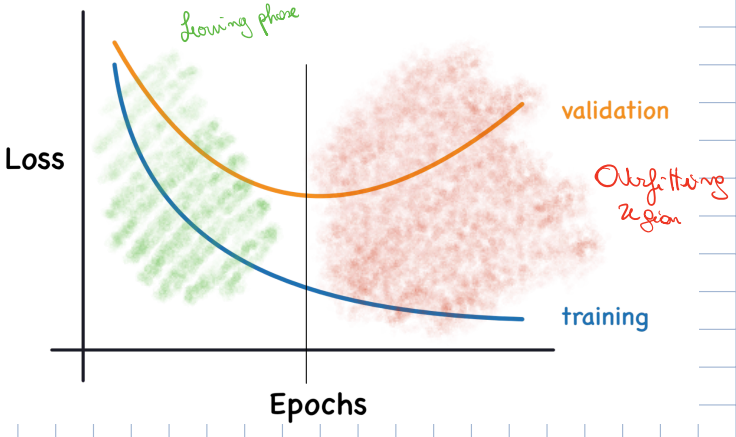
→ With 10k example (10^5)
we are observing a small portion of this high dimensional space w.r.t 10^{40}

There will be a sample every 10^{35} bins

OVERFITTING

Increasing the number of hidden units (Universal theorem approximator) lead the model to perfectly represent the data. But this doesn't mean that the model is actually learning the true distribution underlying the data.





REGULARIZATION

Explicit Regularization

Importance to give to this

$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}} \left(\sum_{i=1}^{D} l_i + \lambda \cdot g(\phi) \right)$$

given a weight value returns a value to influence then the loss. Big value when ϕ has value that is not preferred

Regularization as a Prior

Remember the Maximum Likelihood

$$\hat{\phi} = \underset{\phi}{\operatorname{argmax}} \prod_{i=1}^{D} P(y_i | x_i)$$

$$= \underset{\phi}{\operatorname{argmax}} \prod_{i=1}^{D} P_2(y_i | g[x, \phi])$$

With regularization term

$$\hat{\phi} = \underset{\phi}{\operatorname{argmax}} \left[\prod_{i=1}^{D} P_2(y_i | g(x, \phi)) P_2(\phi) \right]$$

L2 Regularization

Explicit (L2 Regularization)

$g(\phi)$ here is penalizing weights with high values

$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}} \sum_{i=1}^{D} l_i(x_i, y_i) + \lambda \boxed{\sum_{j} \phi_j^2}$$

As a prior L2 Regularization

Consider prior dist$_2$ over the weights $\in N(\mu=0, \sigma^2)$ Then:

$$\hat{\phi} = \underset{\phi}{\operatorname{argmax}} \prod_{i=1}^{D} P_2(y_i | g(x, \phi)) \cdot \prod_{j=1}^{J} \text{Norm}_{\phi_j}[0, \sigma_\phi^2]$$

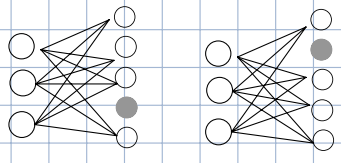
EARLY STOPPING

Stop the training procedure before that overfits the data

DROPOUT

Some hidden layer's unit weight are set to 0, both in comig and outgoing

We may see this as an ensemble of subnetworks



Note: dropout to works need a lot of labeled training example

CAPITOLO 6

CNN - RESNET

- Principles Idea
- CNN Geometric view
- Convolution Theorem
- Discrete Conv
- Conv Layer, Receptive Field
- Resnet : the depth
- Exploding gradient & Both Norm
- U - Net
- Highway Networks

PRINCIPLES IDEA

MLP are universal approximator BUT

- Complexity
- Difficult optimization
- Limited Generalization

We would like to leverage structural similarities

- SELF SIMILARITY
- TRANSLATION INVARIANCE
- DEFORMATION INVARIANCE
- HIERARCHY COMPOSITIONALITY

CNN : GEOMETRIC VIEW

Let's define a Continuous Convolution $(f * g)(x) = \int_{-\infty}^{+\infty} f(t)\, g(x-t)\, dt$

Discrete time Convolution (2 vector x, w) $(x * w)_i = \sum_{k=0}^{n-1} w_{i-k \bmod n}\, x_k$

$y = C \cdot x$

Or as a linear operator $y = C(w)\, x$ where $C(w)$ is a Circulant Matrix (stack the vector w n times with shift)

Let's say something regarding Circulant Matrix

With $W_{vector} = [0, 1, 0 \dots 0]$ we have a Shift-Matrix (a type of Circulant Matrix)

$$\begin{matrix} y \\ 2 \\ 3 \\ 4 \\ 1 \end{matrix} = \begin{bmatrix} & & & \bullet \\ \bullet & & & \\ & \bullet & & \\ & & \bullet & \end{bmatrix} \begin{matrix} x \\ 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

Shift right by 1 → $x \Rightarrow 1,2,3,4$; $y \Rightarrow 2,3,4,1$

PROPRETY

Matrix is circulant if it commutes with shift

C is circulant if $C(w) \cdot S(w) = S(w) \cdot C(w)$

FACT : If Matrices commute, they have the same Eigenvector : they are joint diagonalizable

the eigen values of $C(w)$ is the fourier Transform of w

THE CONVOLUTION THEOREM

Formally : Convolution in the time domain (sliding dot product between kernel and signal) is the same of multiplication in the frequency domain

Compute the convolution between $x * w$ in 2 ways :

Spatial Domain

Take a Circulant Matrix $C(w)$ and apply it to x

Spectral Domain

Compute Fourier Transform of x (signal)

compute FT of w (kernel)

Multiply FT x with FT w

Compute the inverse FT of the Multiplication (go back in time domain)

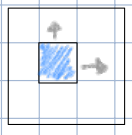
Discrete 2D Convolution

$$(f * g)[m,n] = \sum_k \sum_\ell f[k,\ell] \cdot g[m-k, n-\ell]$$

f: an image : $m \times n$ matrix

As in 1D case we would have $f(t) \cdot g(x-t)$

this is like a sliding window over the image



- Padding
  - Zero padding (Valid)
  - Full Padding (Same) The size of the output = Size of input
- Stride
- Dilation (skip within a convolution certain offset of the current position)
- Dimension after convolution

W_{in} H_{in} F Size of kernel P padding S stride

$$W_{out} = \frac{W_{in} - F + 2P}{S} + 1$$

$$H_{out} = \frac{H_{in} - F + 2P}{S} + 1$$

Convolutional Layer & Receptive Field

Convolutional Layer: Input, Convolve Input, Add Bias, Activation function

$$h_i = a\left[\beta + w_1 x_{i-1} + w_2 x_i + w_3 x_{i+1}\right] = a\left[\beta + \sum_3 w_j x_{i+j-2}\right]$$

F = kernel size = 3

Fully connected VS convolution

6 size of input, 6 hidden unit

$(6 \times 6)_{W} + 6_{Bias} = 42$ pesi

F kernel size = 3

$(3 \times 1 + 6)_{\beta} \times 1 = 9$

Number of parameters

2D $(F \times F \times C_{in} \times C_{out}) + C_{out}$

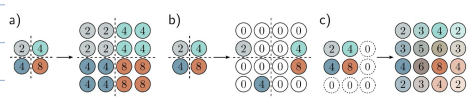
1D $(F \times C_{in} + Bias) \times$ Numb of filters β

Convolutional Network Structure

(Conv Layer → Relu → Activation) $\times n$ - - - → Flatten → MLP

Downsampling
- By Stride
- Pooling: Max, Average

Upsampling
- Bilinear interpolation
- Max Unpooling
- Chanel Duplication
- TRANSPOSED CONVOLUTION



PARAMETERS OPTIMIZATION AND GRADIENT FLOW

RESNET - BATCH NORM

The problem: More layer, better approximation (Universal Approximation Theorem)

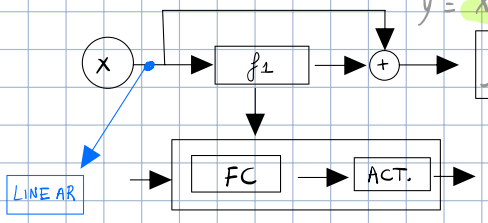
BUT more layer = more difficult search of parameters

- Network do not Converge
- Exploding / Vanishing Gradients

RESIDUAL BLOCK

The introduction of residual connection make the problem of parameters optim. easier and faster

$y = x + f_1(x, \phi_1)$



$$\frac{\partial y}{\partial \phi_1} = \frac{\partial y}{\partial f_1} \cdot \frac{\partial f_1}{\partial \phi_1} = 1 \cdot \frac{\partial f_1}{\partial \phi_1}$$

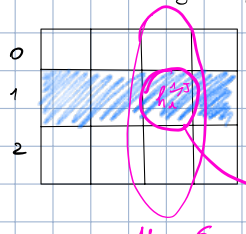
$\frac{\partial y}{\partial f_1} = 1$

BATCH NORMALIZATION

Adaptive reparametrization

$x \xrightarrow{W_0} h_1 \xrightarrow{W_1} h_2 \cdots \xrightarrow{W_e} h_e$ $\quad h_i = (h_{i-1} W_i)$ non linear funct. of weights

Now, let's suppose $H_i = \langle h_i^1 \cdots h_i^k \rangle$ the activation of layer i for the minibatch of samples from 1 to k



$$H_i' = \frac{H - \mu}{\sigma}$$

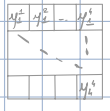
The normalization $\tilde{h}_i^{1,j} = (h_i^{1,j} - \mu_j) / \sigma_j$

Element 1 of the batch, j element of the hidden vector for the hidden layer i

Both μ and σ are trainable parameters

While training $\mu = \frac{1}{k} \sum_i H_i$

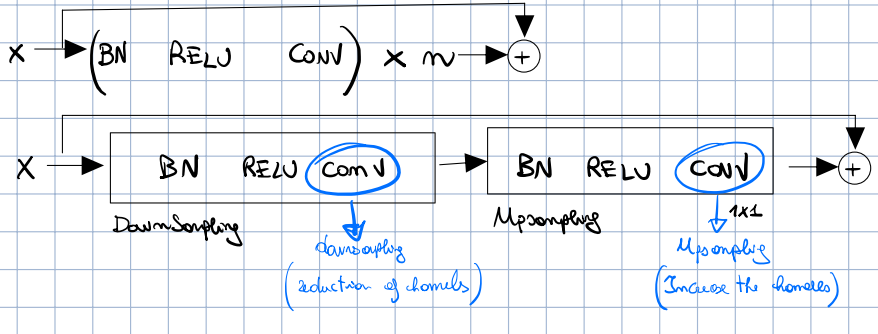
Mean vector μ computed for each layer i



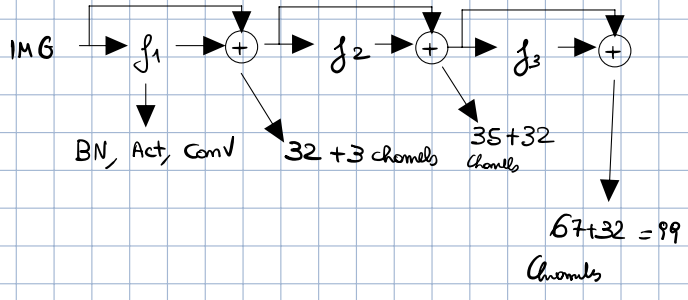
$$\sigma = \sqrt{\delta + \frac{1}{k} \sum_i (H - \mu)^2_i}$$

RESNET + BATCH NORM ARCHITECTURE

RESNET

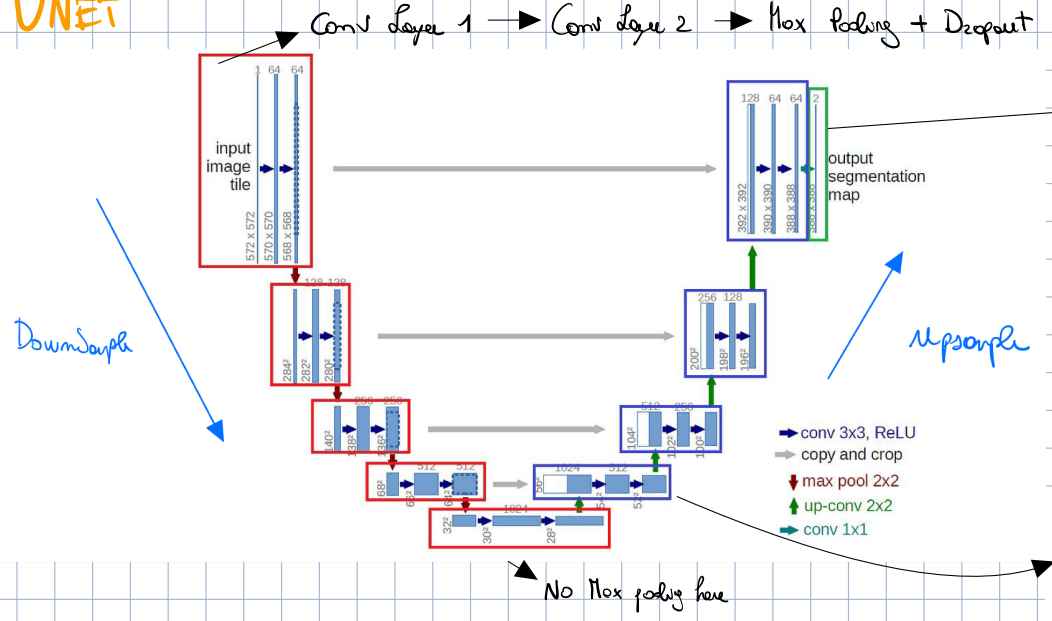


DENSENET

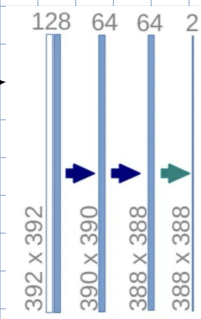


UNET

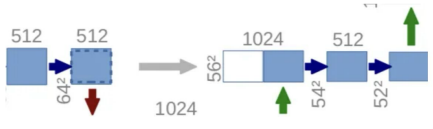
Conv Layer 1 → Conv Layer 2 → Max Pooling + Dropout



No Max pooling here




```
conv_2d_transpose -> concatenate -> conv_layer1 -> conv_layer2
```



HIGHWAY NETWORKS

given $y = f(x, W_f)$

on highway nn also have other 2 transformation of the input x : $T(x, W_T)$ $C(x, W_C)$

$$y = f(x, W_f) \cdot \underbrace{T(x, W_T)}_{\text{Transform Gate}} + x \cdot \underbrace{C(x, W_C)}_{\text{Carry gate}}$$

Usually set as $1 - T(x, W_T)$

CAPITOLO 7

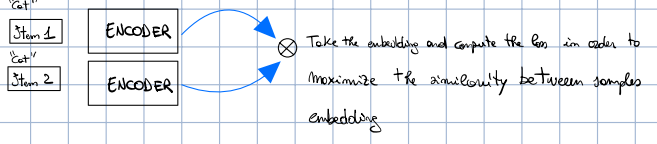
SELF SUPERVISED LEARNING

- Siamese NN
- Neural Collaborative Filtering
- Self Supervision / Proxy Task
- Contrastive Learning / Triplet Loss
- SIMCLR
- BYOL + Evaluation
- BARLOW TWINS + Evalu.
- Deep Metric

SIAMESE NEURAL NETWORK

Two or more identical subnetworks.

IDEA: Some items should have some representation



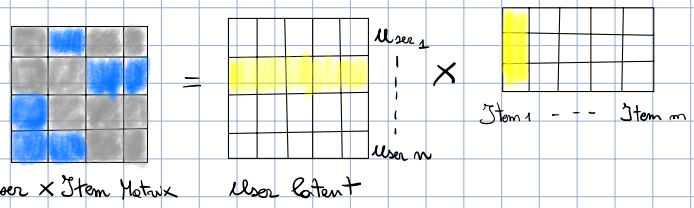
Use data as label for themself to learning their represent.

NEURAL COLLABORATIVE FILTER

User $= \{u_1 \dots u_m\}$ Item $= \{i_1 \dots i_n\}$ Preferences $= \{(u_i, i_j, y) \dots\}$

Goal: get a ranked list of items (unseen) for a certain user

$\hat{y}_{ui} = f(u, i \mid \theta)$ predict the score of interaction for (u, i)

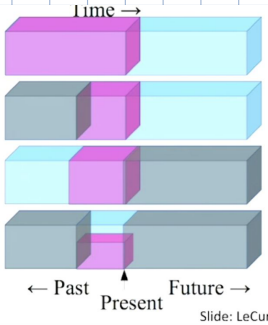


User x Item Matrix — User Latent

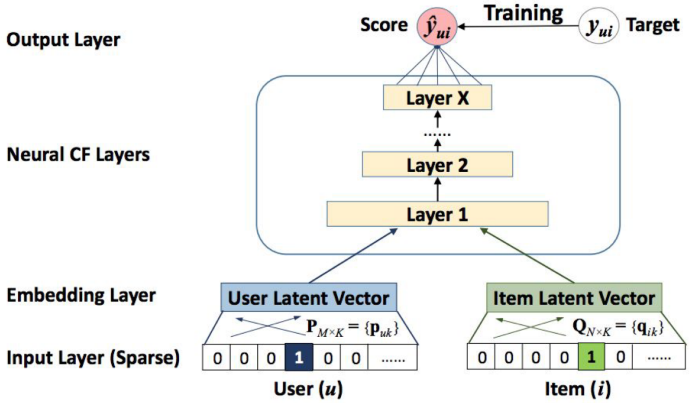
SELF SUPERVISION PROXY TASK

We don't care about performance of specific application, we want to learn input representation such that we can then use them in downstream task. We learn by the use of PROXY TASK.

- Predict any part of the input from any other part.
- Predict the future from the past.
- Predict the future from the recent past.
- Predict the past from the present.
- Predict the top from the bottom.
- Predict the occluded from the visible
- Pretend there is a part of the input you don't know and predict that.



Slide: LeCun



CONTRASTIVE LEARNING

IDEA: similar samples close, while dissimilar ones far apart

Sample from same class → Similar embeddings

$x_1, x_2 \in$ class X → minimize embedding distance

$x_1 \in X$, $x_2 \in Y$ → maximize embedding distance with a lower bound distance between different classes

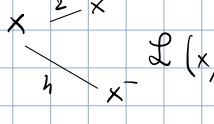
CONTRASTIVE LOSS (x_1, x_2)

$$\mathcal{L}(x_i, x_j, \theta) = \mathbb{I}[y_i = y_j] \left\| f_\theta(x_i) - f_\theta(x_j) \right\|_2^2 + \mathbb{I}[y_i \neq y_j] \max\left(0, \varepsilon - \left\| f_\theta(x_i) - f_\theta(x_j) \right\|_2\right)$$

TRIPLET LOSS

$$\mathcal{L}_{\text{Triplet}}(x, x^+, x^-) = \sum_{x \in \mathcal{D}} \max\left(0, \|f(x) - f(x^+)\|_2^2 - \|f(x) - f(x^-)\|_2^2 + \varepsilon\right)$$

To make this loss decrease the positive distance has to be at least the negative distance $+\varepsilon$



$\mathcal{L}(x, x^+, x^-) = 2 - 4 + \varepsilon = -2 = \max(0, -2) = 0$ God

IF x — 4 — x^+, x — 2 — x^-: $\max(0, 4-2) = 2$ Bad

INGREDIENTS

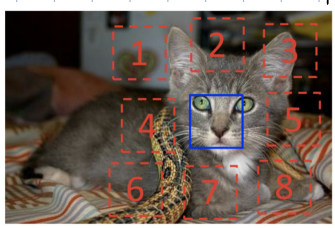
- Create positive samples with data augmentation
- Large Batch size to get a good negative-samples coverage
- Create a proxy strategy to get hard negative (by labels or by an algorithm)

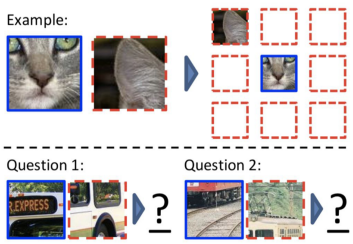
IMAGE BASED SELF-SUPERVISED LEARNING

WORKFLOW

- Training with pretext task (a proxy task)
- Use a layer to train a logistic regression over the ImageNet classes Evaluating the representation given

→ Have some representation for patches corrupted but starting from an original one (Exemplar-CNN)
→ Given the degree of rotation of an image
→ Predict Relation between patches



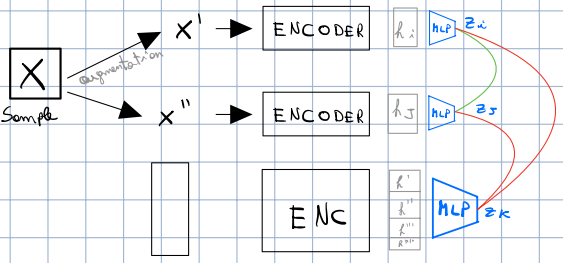


X = (,); Y = 3

→ Use the 9 patches to guess which permutation of the patches has been applied

SIMCLR

(x, x'), (x, x'') positive pairs all the rest are negative



$$\mathcal{L}_{SIMCLR} = -\log \frac{\exp\left(\text{sim}(z_i, z_j)/t\right)}{\sum_{k=1}^{2N} \mathbb{I}_{[k \neq i]} \exp\left(\text{sim}(z_i, z_k)/t\right)}$$

Weight to give to the similarity

do not consider sim with same image

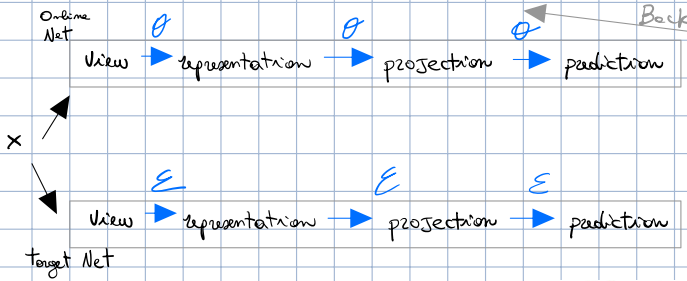
BYOL

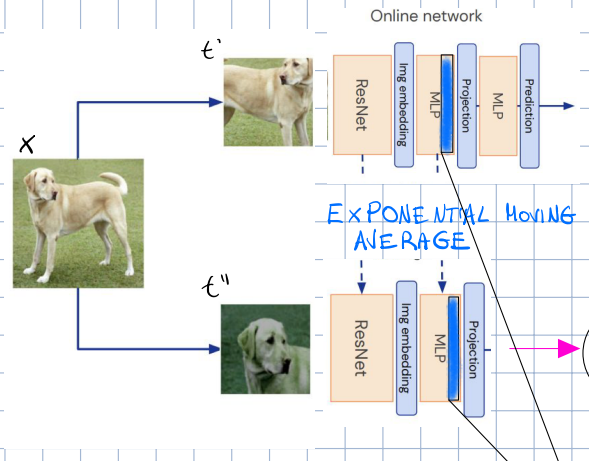
Bootstrap your own latent

Can we get rid of negatives?

Giving 2 identical network with 3 stages
Train online net and update the θ parameter with GD
But update the target with MOVING AVERAGE

→ This avoid the 2 represent. embedding from online and target network to not be equal hence the function to be constant





$q_\theta(z_\theta)$

EXPONENTIAL MOVING AVERAGE

$$\xi = \tau\,\xi + (1-\tau)\,\theta$$

importance of current parameters ξ New θ parameters

$$\frac{\langle q_\theta(z_\theta), z'_\xi \rangle}{\|q_\theta(z_\theta)\|_2 \cdot \|z'_\xi\|_2}$$

z'_ξ

this is a prediction

BATCH NORM
Create contrastive samples

LOSS: Minimize the distance between represent. given by online network with represent. given by target network

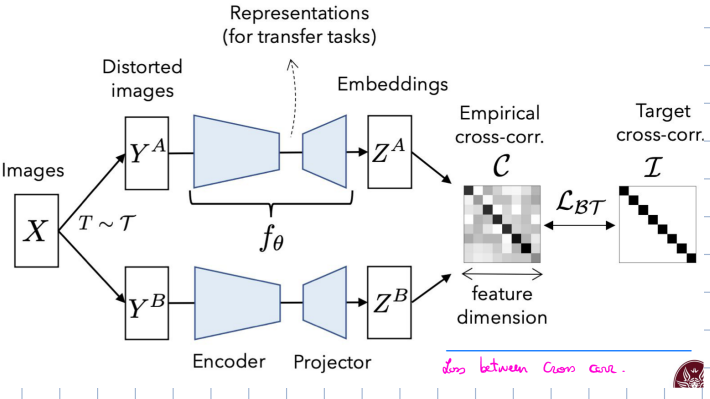
$$\mathcal{L}_{\theta,\xi} = \left\| q_\theta(z_\theta) - z'_\xi \right\|_2^2 = 2 - 2 \cdot \frac{\langle q_\theta(z_\theta) - z'_\xi \rangle}{\left\| q_\theta(z_\theta) \right\|_2 \cdot \left\| z'_\xi \right\|_2}$$

Magnitude of 2 vectors → a kind of cos similarity

BARLOW TWINS

Measure cross-correlation matrix between the embeddings of two identical networks. GOAL → Make this matrix close to the identity

EFFECT: the embedding of distorted version of the same sample have to be similar



LOSS

Cross Correlation matrix between the output of two identical networks along the batch dimension

$$\mathcal{L}_{BT} = \sum_i \left(1 - C_{ii}\right)^2 + \lambda \sum_i \sum_{j \neq i} C_{ij}^2$$

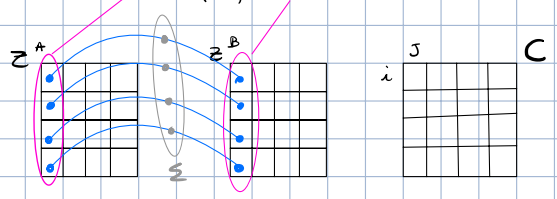
Invariance term: Try to make the diagonal elements of the cross-correlation matrix = 1 (Makes the embeddings invariant to distortion)

Redundancy Reduction term: Try to make the off-diagonal elements equal to 0 → output units contain non redundant information for the sample

Cross Correlation Matrix computed as:

$$C_{ij} = \frac{\sum_b z^A_{b,i} \cdot z^B_{b,j}}{\sqrt{\sum_b \left(z^A_{b,i}\right)^2} \sqrt{\sum_b \left(z^B_{b,j}\right)^2}}$$

along batch dim



SUPERVISED DEEP METRIC LEARNING

Again, put sample of the same class near in the latent space using a feature extractor $f_\theta : X^D \to \mathbb{R}^N$.

TECHNIQUE → Contrastive learning

Expansion issue: it is not guaranteed that $\forall$ sample $j \in y_i$ the distance between sample is small

Sampling issue: How to define negative samples?

Center Loss: Taken the representation, use a classifier using softmax. Add a term to loss to ensure that samples are pushed toward the centroid of the class

$$\mathcal{L}_{center} = \mathcal{L}_{softmax} + \frac{\lambda}{2} \sum_i^N \left\| z_i - c_{y_i} \right\|_2^2$$

distance from the centroid of class of sample i

Sphere Face: Modify the softmax to allow a different choice of the "center" in the center loss

Consider θ_{y_i} as the angle between z_i and centroid of class y

Starting from the Softmax

$$\mathcal{L}_{modified\ softmax} = -\frac{1}{N}\sum_{i}^{N} \log \frac{\exp\left(W_{y_i}^T z_i + b_{y_i}\right)}{\sum_{j}^{m} \exp\left(W_j^T z_i + b_j\right)} = -\frac{1}{N}\sum_{1}^{N} \log \frac{\exp\left(\|z_i\| \cos\left(\theta_{y_i}\right)\right)}{\sum_{j}^{m} \exp\left(\|z_i\| \cos\left(\theta_{j_i}\right)\right)}$$

We assign to sample i the class y if the projection z_i to the center of class y is the largest or if the cosine angle is the smallest between all the centroids

CAPITOLO 8

TRANSFORMERS

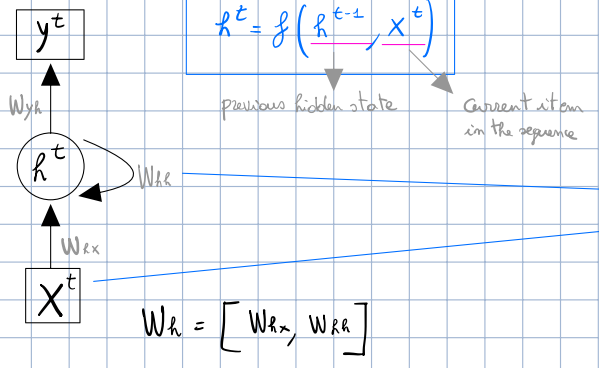
- RNN
- Seq 2 Seq
- Autoregressive Models
- ELMo, Word 2 Vec
- Attention, Seq-Attention, MHA
- Transformer
- BERT
- Unimodal Transformer
- Transfer learning, Neural IDBS, T5

RNN

X = Input, composed of sequence data (Text, Audio, Video)

Only one architecture to each item of the sequence. The weight are shared for all the tokens.

RNN can in theory deal with an arbitrary long sequences



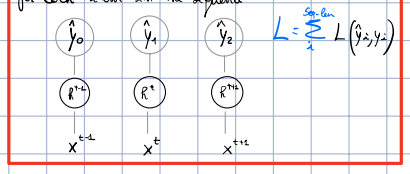
An Example of RNN

Starting from $h^t = f(h^{t-1}, x^t)$

$$h^t = g_\theta(h^{t-1}, x^t)$$
$$h^t = \tanh(W_{hh} h^{t-1} + W_{xh} x^t)$$
$$y^t = \text{softmax}(W_{yh} h^t)$$

Note on Back propagation

Each sample is a sequence ($\langle x_0, y_0 \rangle \langle x_1, y_1 \rangle \dots$)
this means we can compute the loss for each item in the sequence

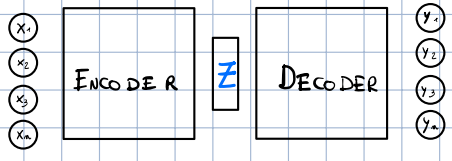


SEQUENCE TO SEQUENCE

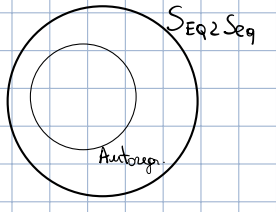
Many to one → Classification

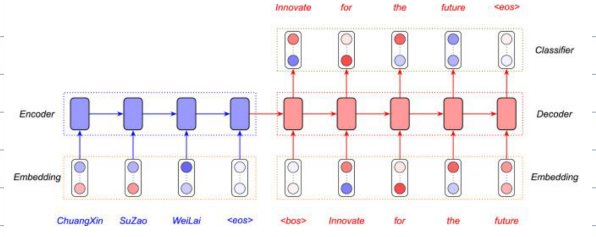
Many to many

One to Many

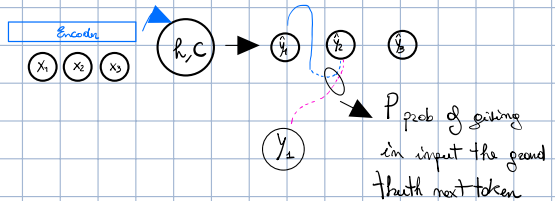


AUTOREGRESSIVE MODELS





TEACHER FORCING



WORD 2 VEC

CBOW

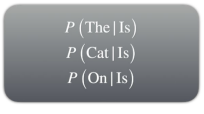
Predict target word from the context

$$P(w_t \mid w_{t-h} \dots w_{t-1}, w_{t+1} \dots w_{t+h}) = \text{softmax}(w_t U V^T) \quad U, V \in \mathbb{R}^{m \times d}$$

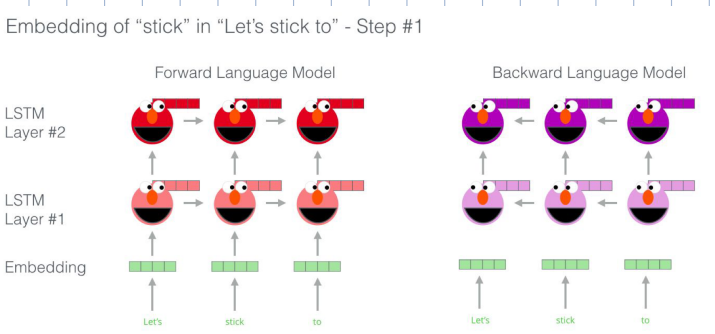
ISSUE
Static word embedding: the embedding vector does not change for ambiguous words

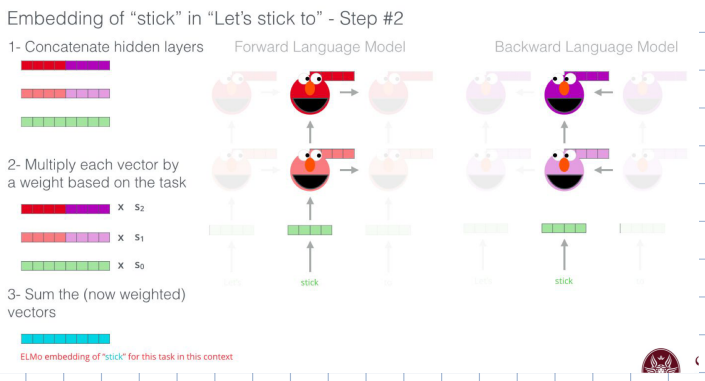
Skipgram

Predict the context from the target



ELMO

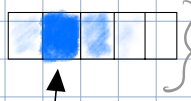




THE ATTENTION MECHANISM

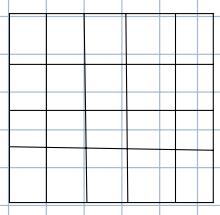
THE ATTENTION: Association for each output a vector long as the input sequence

Let's take a vector for one output



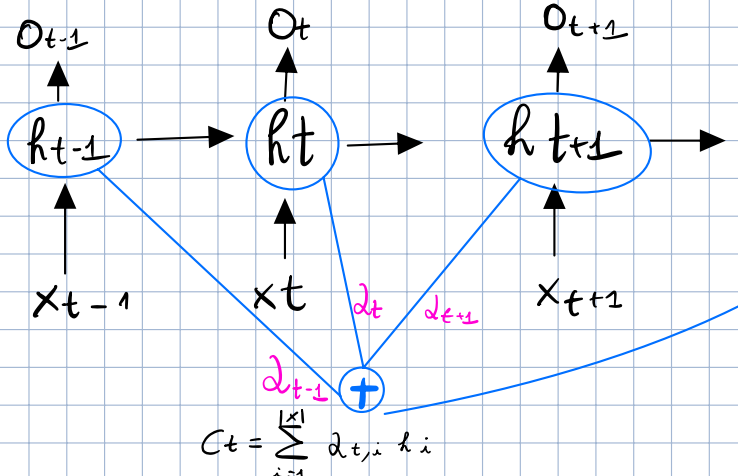
Formally Input $x = [x_1 \dots x_n]$ $y = [y_1 \dots y_m]$

The attention is a matrix



$|x| = n$ $|y| = m$

RNN ENCODER



$C_t = \sum_{i=1}^{|x|} \alpha_{t,i} h_i$

DECODER

the decoder works on s_t that is a function of s_{t-1}

C_t context = sum of hidden states of the encoder weighted by the attention score α

α represent the alignment

"Align the output to the input" of the output token

α is a kind of multiclass classification

$\text{softmax}(y_t, x_i)$

the scores α are learnt by another NN

$$\text{SCORE}(s_t, h_i) = \underbrace{v_\alpha^T \tanh\left(W_\alpha [s_t, h_i]\right)}_{\text{to be learnt}}$$

SCALED DOT PRODUCT ATTENTION

$$SCORE(s_t, h_i) = \frac{s_t^T h_i}{\sqrt{n}}$$

$\sqrt{n}$ → normalized by seq. lenght

THE SELF ATTENTION

Input $X = [x_1 \dots x_n]$ x_i is a vector (input embedding)

The self-attention: given each word which are the other word in the same sentence that contribute the most to explain the semantic of that word

$$Attention(Q, k, V) = softmax\left(\frac{Qk^T}{\sqrt{n}}\right)V$$

scaled dot product

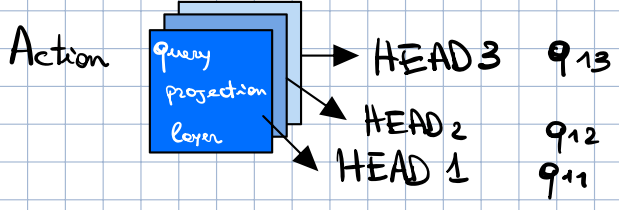
//

Example of self attention "Action gets result

	query	key	value	score	score/$\sqrt{n}$	Softmax	Softmax * V	Sum
Action	q_1	k_1	V_1	$q_1 \cdot k_1$	$q_1 \cdot k_1 / 8$	X_{11}	$X_{11} \cdot V_1$ →	Z_1
gets		k_2	V_2	$q_1 \cdot k_2$	$q_1 \cdot k_2 / 8$	X_{12}	$X_{12} \cdot V_2$ →	
result		k_3	V_3	$q_1 \cdot k_3$	$q_1 \cdot k_3 / 8$	X_{13}	$X_{13} \cdot V_3$ →	

SCALED DOT Product

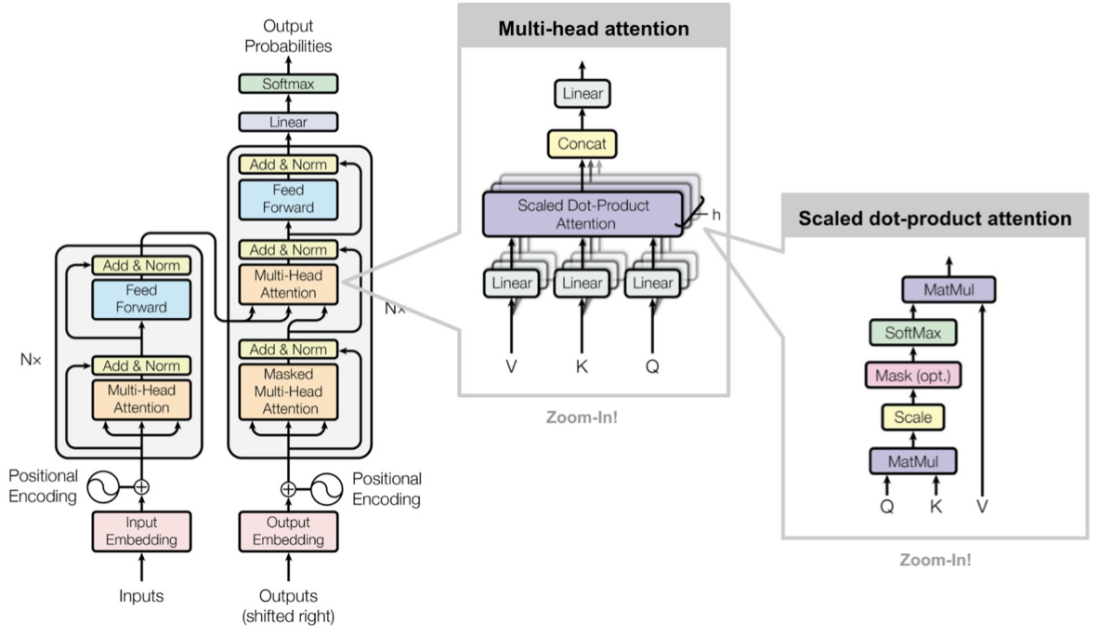
MULTI HEAD ATTENTION



$$Multihead(Q, k, V) = [head\ 1 \dots head\ h]$$

$$head\ i = Attention[QW_i^q, kW_i^k, VW_i^v]$$

TRANSFORMER



BERT Bidirectional Encoder Representation from Transformer

Multi layer bidirectional transformer Base 110 M / Large 340 M

Proxy Tasks

- Masked Language Modeling — some input is masked and the model has to predict the right token
  - 80% [MASK] token
  - 10% Random token
  - 10% Same token
- Next Sentence Prediction

PRE-TRAINING

[CLS] + Input Embedding + Positional Emb + Segment embedding (Sentence 1) + [SEP] + Input Embedding + Positional Emb + Segment embedding (Sentence 1)
(Wordpiece Embedding) (Wordpiece Embedding)

FINE-TUNING

(Named Entity recognition, Grounded common sense inference, Textual Entailment, Question Answering) → Sentence 1 + Continuation → FFN → Softmax → Label

∀ head, ∀ token, 1 Hidden state → Hidden state is a tensor (For Bert Base / Large 12/24, seq_len, 768/1024) → Pooling → FFN → Label

TRANSFER LEARNING : Neural IDBS

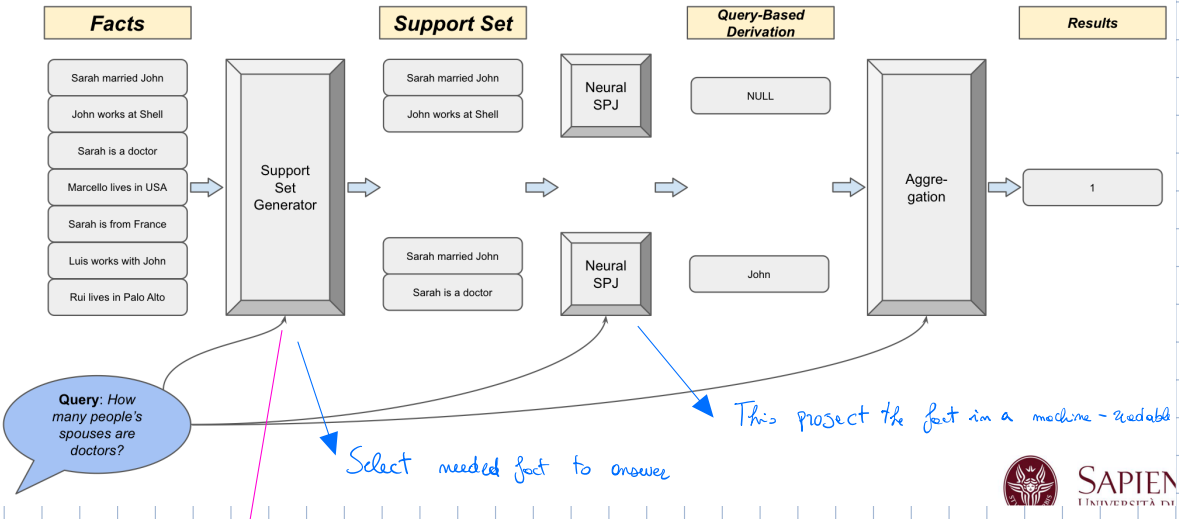
Make use of large pretrained model to adopt its ability on performing downstream task.

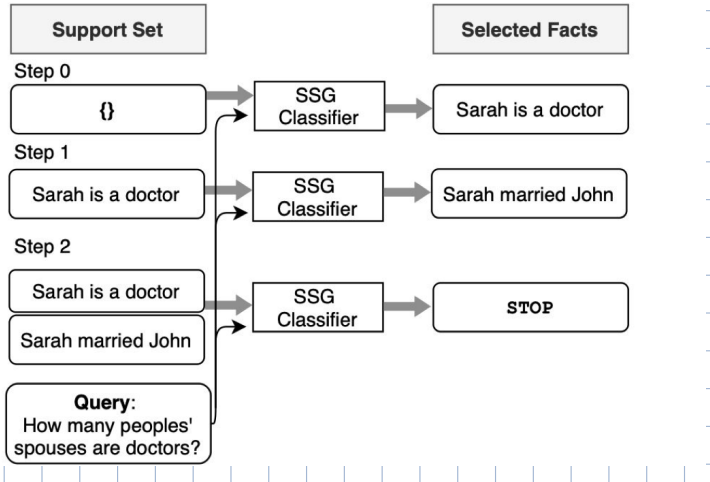
Consist in basically replace the SCHEMA of the database with the only unstructured content.

This can be used by a Neural Processor which can interprete the content and reply to answer based on that content.

Neural Query Processing

Given a query and a small number of facts from the database, can the T5 accurately answer queries that are posed in natural language, whose answer may require projection (i.e., extracting part of a sentence), join, and aggregation?





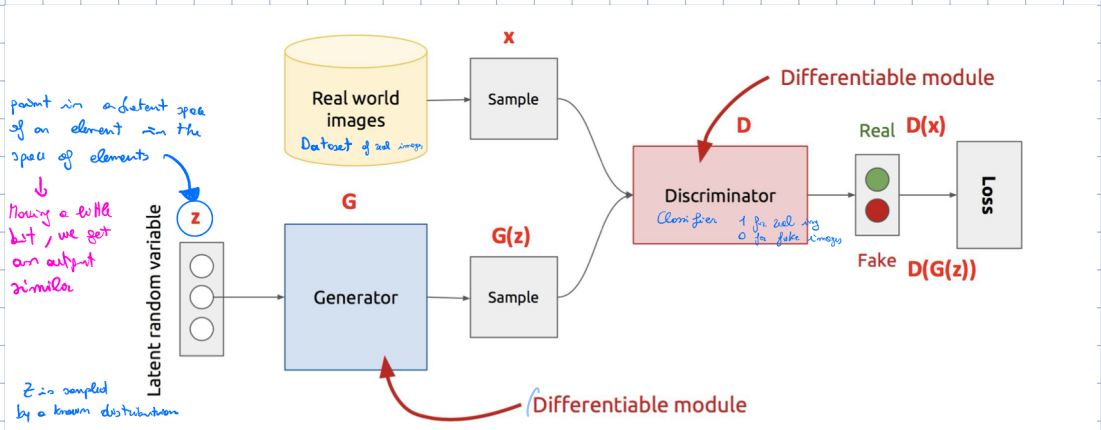
CAPITOLO 9

GENERATIVE AI

- Discriminative vs Generative
- GAN, Loss, Convergence, Mode collapse
- Wasserstein GANS
- Autoencoders, Denoising Autoencoders
- VA, Elbo, Reparametrization Trick

GAN

Considering the prob. of X $P_\theta\{x\}$. We assume the generation of $x \in X$ conditioning on a variable z, $P_\theta\{x|z\}$ and z comes from a normal distrib. $z \sim N(0, \sigma)$



TRAIN GAN

1. Freeze Generator, Train discriminator
2. Freeze Discriminator, Train generator

GAN TRAINING: SETUP

for epoch

Train discriminator, while the generator is freezed

for k step:

- obtain $P_{data}(x)$ sampling m samples from Dataset
- obtain m samples generated

$$\max_D \frac{1}{m} \sum_{i=0}^{m} \log D(x_i) + \frac{1}{m} \sum_{i=0}^{m} \left[\log\left(1 - D\left(G(z_i)\right)\right)\right]$$

Train generator

- obtain m generated samples

$$\min_G \frac{1}{m} \sum_{i=1}^{m} \log\left[1 - D\left(G(z_i)\right)\right]$$

this is why the generator want to minimize

if $D(G(z_i)) = 1$: the discriminator says that the image is fake, then $\log \to \infty$

while if $D(G(z_i))$ miss classifies the sample, saying that is real then $\log \to 0$

ALL MATHEMATICAL PROOF FOR GAN

- The value of generator depends on the discriminator
  - When the value of the discriminator is low (is committing errors, the value of the generator is high)
  - When the value of the discriminator is high means that the generator is no able to fool it, hence the discriminator detect all the fake images

This is a ZERO-SUM GAME (if one win, the other lose)

$$\min_G \max_D V(D,G)$$

NASH Equilibrium: the moment of convergence of a GAN. When is the moment in which the generator is working at best?

Basically when is totally uncertain about a sample

$$D(x) = \frac{1}{2} \quad \forall x$$

$\frac{1}{2}$ Real / $\frac{1}{2}$ Fake — MAX UNCERTAINTY

this means that the generator has learnt the distribution of the data

$$P_{data}(x) = P_{gen}(x) \quad \forall x$$

there is a possible set of strategies (in this case the set of weight for the discriminator and for the generator) such that changing one strategy will make worse one of the strategy in the set.

Obtain the best Discriminator

$$\max_D V(G,D) = \mathbb{E}_{x \sim P_{data}(x)}\left[\log D(x)\right] + \mathbb{E}_{z \sim q(z)}\left[\log\left(1 - D(G(z))\right)\right]$$

- $\max_D$: Maximization Post
- $\mathbb{E}_{x \sim P_{data}(x)}$: expresses the prob. of x to come from P_{data} or P_z
- $z \sim q(z)$: A pre defined distrib such as $N(0, \sigma)$
- $1 - D(G(z))$: Minimization Post

Basically this means that we want to maximize the probability for the discriminator to predict $x \sim p_{data}$ and minimize the probability that discriminator predicts $G(z) \sim p_{data}$

Obtain the best generator

We want for the best generator that is able to fool the discriminator: is able to maximize the probability that $D(G(z)) = \text{True}$

$$\max_G V(D,G) = \mathbb{E}_{z \sim q(z)}\left[\log D(G(z))\right]$$

Equivalent to

$$\min_G V(D,G) = \mathbb{E}_{z \sim q(z)}\left[\log\left(1 - D(G(z))\right)\right]$$

Then we can reach the Nash Equilibrium merging the best discriminator and the best generator in min max settings

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}} \log\left[D(x)\right] + \mathbb{E}_{z \sim q(z)} \log\left[1 - D(G(z))\right]$$

Law of Unconscious Statistician (LotUS)

$z \sim X$ sampled from X

$$E(g(X)) = \int_{-\infty}^{\infty} g(z)\, f_X(z)\, dx$$

- $E(g(X))$: a random variable
- $f_X(z)$: probability density function of X

1 Let's rewrite the second part of the loss

$$\mathbb{E}_{z \sim p(z)} \log\left[1 - D(G(z))\right] = \mathbb{E}_{z \sim p_G(z)} \log\left[1 - D(G(z))\right]$$

Consider samples coming out from the generator directly

2 Apply Lotus for the two terms of the GAN loss

$$\int_x \frac{P_{data}(x)}{\text{Prob. density function}} \log D(x)\, dx + \int_z P_G(z) \log\left[1 - D(G(z))\right] dz$$

- $\int_x$: over the random variable x
- $\int_z$: over random variable z
- $P_G(z)$: PDF

unifying the two integrals $=$

$$\int_x P_{data}(x) \log D(x) + P_G(x) \log\left[1 - D(G(x))\right] dx$$

3 Use the Leibniz Rule to obtain the optimal D given G

$$\frac{d}{dx}\left(\int_a^b f(x,t)\, dt\right) = \int_a^b \frac{\partial}{\partial x} f(x,t)\, dt$$

take the derivative inside and because x is multivariate we must take the partial derivate ∂

- a point where the previous integral is minimized
- hence where the derivative is 0

Let's write again the integral using a function $f(y)$

$f(y) = a \log y + b \log(1-y)$ and the derivative of $f'(y)$

- a: P_{data}
- y: $D(x)$
- b: P_G

$$f'(y) = \frac{a}{y} - \frac{b}{(1-y)}$$

4 to minimize we set the derivative to 0

$$f'(y)=0 \Rightarrow \frac{a}{y} - \frac{b}{(1-y)} = 0 \Rightarrow y = \frac{a}{a+b} \Rightarrow D(x) = \frac{p_{data}}{p_{data}+p_G}$$

Knowing that that the Nash Equilibrium if $p_{data} = p_G$ we have $D(x)=\frac{1}{2}$ $D(x) = \frac{1}{2}$ lead to the best generator, then

Theorem: the global minimum of the criterion $C(G) = \max_D V(G,D)$ is reached when $p_G = p_{data}$

Proof:

- Assumption : we know $D^* = \frac{1}{2}$ and $p_G = p_{data}$

Just plug $\frac{1}{2}$

$$V(G,D^*) = \int_x p_{data}(x) \log \frac{1}{2} + p_G(x) \log\left(1-\frac{1}{2}\right) dx$$

$$= -\log 2 \underbrace{\int_x p_{data}(x)\,dx}_{1} - \log 2 \underbrace{\int_x p_G(x)\,dx}_{1} = 2 - \log 2 = -\log 4$$

CANDIDATE FOR GLOBAL MINIMUM

- Assumption $C(G) = \max_D V(G,D)$

$D_G^* = p_{data} / (p_{data}+p_G)$

Let's plug D_G^* into $C(G) = \max_D V(G,D)$

$$1 - \frac{p_{data}}{p_{data}+p_G} = \frac{p_{data}+p_G-p_{data}}{p_{data}+p_G}$$

$$C(G) = \int_x p_{data}(x) \log\left(\frac{p_{data}}{p_{data}+p_G}\right) + p_G(x) \log\left(\frac{p_G}{p_{data}+p_G}\right) dx$$

Now let's write again $C(G)$ using a trick

$$C(G) = \int_x \left(\log 2 - \log 2\right) p_{data}(x) + p_{data}(x) \log\left(\frac{p_{data}}{p_{data}+p_G}\right) + \left(\log 2 - \log 2\right) p_G(x) + p_G(x) \log\left(\frac{p_G}{p_{data}+p_G}\right) dx$$

$$C(G) = \int_x \log 2\, p_{data}(x) - \log 2\, p_{data} + p_{data}(x) \log\left(\frac{p_{data}}{p_{data}+p_G}\right) + \log 2\, p_G(x) - \log 2\, p_G(x) + p_G(x) \log\left(\frac{p_G}{p_{data}+p_G}\right)$$

$$= \int_x -\log 2 \left(p_{data}(x)+p_G(x)\right) dx + \int_x p_{data}(x)\left(\log 2 + \log\left(\frac{p_{data}}{p_{data}+p_G}\right)\right) + \int_x p_G(x)\left(\log 2 + \log\left(\frac{p_G}{p_{data}+p_G}\right)\right) dx$$

$$= \boxed{-\log 2 \int_x p_{data}(x)+p_G(x)\,dx} + \int_x p_{data}(x)\left(\log 2 + \log\left(\frac{p_{data}}{p_{data}+p_G}\right)\right) + \int_x p_G(x)\left(\log 2 + \log\left(\frac{p_G}{p_{data}+p_G}\right)\right) dx$$

by definition of probability distributions the integration $=1$

$-\log_2(1+1) = -2\log 2$

$\log\left(2\,\frac{p_{data}}{p_{data}+p_G}\right)$

$$-\log 4 + \int_x p_{data}(x) \log\left(\frac{p_{data}(x)}{(p_{data}+p_G)/2}\right) dx \quad \int_x p_G(x) \log\left(\frac{p_G}{(p_{data}+p_G)/2}\right) dx$$

the 2 integrals are KL divergences

$$C(G) = -\log 4 + KL\left(p_{data} \,\middle|\, \frac{p_{data}+p_G}{2}\right) + KL\left(p_G + \,\middle|\, \frac{p_{data}+p_G}{2}\right)$$

Because this are non-negative we see that $-\log 4$ is the global minimum of this function

or written as $C(G) = -\log 4 + 2 \cdot JSD\left(p_{data} \,|\, p_G\right)$

CONVERGENCE OF GAN

In the $\min_G \max_D$ formulation with $D^*_G = \frac{1}{2}$ the p_G converges to p_{data}

$$\min_G \max_D V(G,D) = \mathbb{E}_{x\sim p_{data}(x)}\left[\log D^*_G(x)\right] + \mathbb{E}_{x\sim p_G}\left[\log 1 - D^*_G(x)\right]$$

(And we know that $p_G = p_{data}$ is when the nash equilibrium is reached and when $D(x) = \frac{1}{2}$, the discriminator is totally uncertain)

PROBLEMS OF GANS

Saturation of loss G

In beginning of learning, G is poor D reject sample with high confidence because samples are really different from real ones instead of minimize $\log[1 - D(G(z))]$ that saturate we can maximize $\log[D(G(z))]$

Mode Collapse

The generator learns the subset (of very similar) samples that fools the discriminator. If the discriminator get stuck in a local minimum and do not find that the subset of generated is fake, then the G will never learn to generate other samples.

When we have $\nabla_{\theta_g} \frac{1}{n}\sum_{i=1}^{n} \log\left[1 - D(G(z_i))\right]$

we will obtain

$x^* = \text{argmax}_x D(x)$

and x^* will be almost independent from z.

Considering the gradient $\frac{\partial Loss}{\partial z} \approx 0$

and no update will be performed

D is too powerful

that G's gradient vanish and doesn't learn

WASSERSTEIN GAN

$P_1 = 3$ $P_2 = 2$ $P_3 = 1$ $P_4 = 4$

$Q_1 = 1$ $Q_2 = 2$ $Q_3 = 4$ $Q_4 = 3$

} How do I match this 2 distributions?

Move $(2, P_1, P_2)$ $P_1 = Q_1$ $P_1 = 1$ $P_2 = 4$ $P_3 = 1$ $P_4 = 4$

Move $(2, P_2, P_3)$ $P_2 = Q_2$ $P_1 = 1$ $P_2 = 2$ $P_3 = 3$ $P_4 = 4$

Move $(1, Q_3, Q_4)$ $P_3 = Q_3$, $P_4 = Q_4$ $Q_1 = 1$ $Q_2 = 2$ $Q_3 = 3$ $Q_4 = 4$

$W = \sum |\delta_i| = 2 + 2 + 1 = 5$ This is the discrete case

For the continuous case

$$W(P_r, P_g) = \inf_{\gamma \sim \Pi(P_r, P_G)} \mathbb{E}_{(x,y)\sim\gamma}\left[\|x - y\|\right]$$

take the smallest amount of sand to move

distance between element in the 2 distrib

γ: all the possible sand movement PLAN between the 2 distr.

We can use the wasserstein distance as a gan loss function

This distance is more smoothly, not like KL or JSD

If we want to use W distance in the gan,
we have to consider that we cannot enumerate γ (all the possible set of plans of dust to move)

We can use $$W(p_z, p_g) = \frac{1}{k} \sup_{\|f\|_L \leq k} \mathbb{E}_{x \sim p_r}[f(x)] - \mathbb{E}_{x \sim p_g}[f(x)]$$

f should be k-lipschitz continuous

Instead of take infimum over joint distribution, now we take sup of a function

Now we parametrize f with a set of parameters that we optimize. f will be a Neural Net

We will have an objective that directly maximize the similarity between p_{data} and p_G

$$Loss(p_z, p_g) = W(p_z, p_g) = \max_{w \in W} \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_z(z)}[f_w(g_\theta(z))]$$

TRAIN A GAN with WASSERSTEIN DISTANCE

While θ not converged

for iteration

update the params to enforce $p_{data} = p_g$ using W. distance

- Sample $\{x_i\}_{i=1}^m$ Real data
- Sample $\{z_i\}_{i=1}^m$ Noise Samples
- $g_w = \nabla_w \left[\frac{1}{n} \sum_{i=1}^m f_w(x^i) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z_i)) \right]$ gradient given by W. distance

  this is to enforce $p_{data} = p_g$ by the using of a function f_w

  using the parameter of generator
- $w = w + \alpha \, RMSProp(w, g_w)$
- $w = clip(w, -c, c)$ this is to enforce k-Lipschitz

Update G params

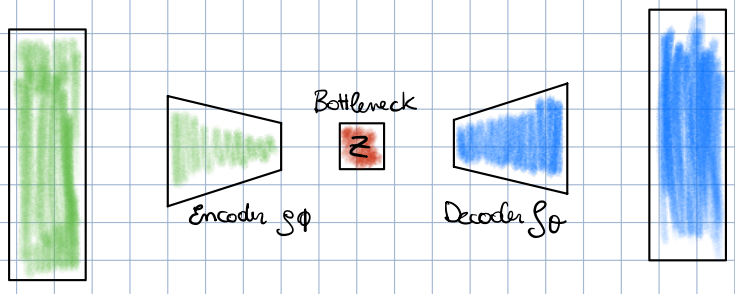
Sample $\{z_i\}_{i=1}^m$ from $q(z)$ a Noise Samples

$g_\theta = -\nabla_\theta \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z_i))$

gradient w.r.t θ params of generator

$\theta = \theta - \alpha \, RMSProp(\theta, g_\theta)$

AUTOENCODERS



INPUT X → MSE ← Output x'

$$L(\phi,\theta) = \sum_{i=1}^{m} (x_i - x'_i)^2 = \sum_{i=1}^{m} (x_i - f_\theta(g_\phi(x_i)))^2$$

Note: this has a problem, might learn an identity function

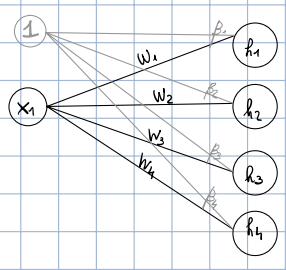
Denoising autoencoder

Make the input noisy $\tilde{x} = M_{\mathcal{D}}(x)$ (noise)

$x' = f_\theta(g_\phi(\tilde{x}))$

SPARSE AUTOENCODER

Avoid overfitting, improve robustness.
Enforce activation of few neurons.



$h_1 = a(\beta_1 + w_1 x_1)$

$h_2 = a(\beta_2 + w_2 x_1)$

$h_3 = a(\beta_3 + w_3 x_1)$

$h_4 = a(\beta_4 + w_4 x_1)$

$\sum_4 = \hat{\rho} \Rightarrow \frac{1}{m} \sum_{i=1}^{m} h_i$

Enforce small activations

with a term in the loss

$$L(\phi,\theta) = L_{MSE}(\phi,\theta) + \beta D_{KL}(\rho \| \hat{\rho}^{(l)})$$

ρ: target activation

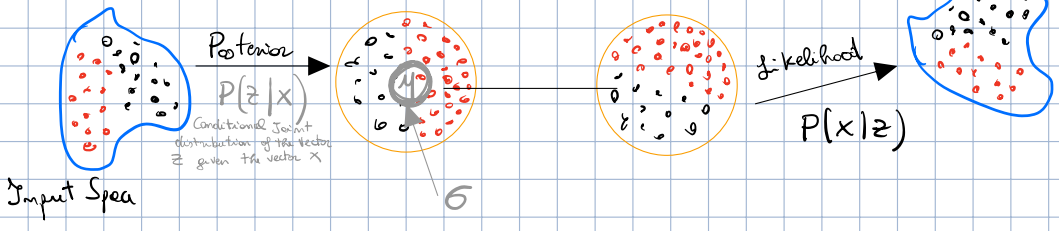
$\hat{\rho}^{(l)}$: for a certain layer

VARIATIONAL AUTOENCODERS

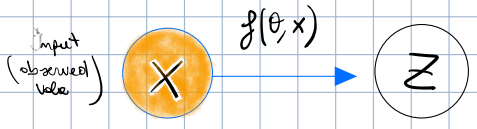
Idea Behind: the input space follows a distribution. We want to learn a representation for the whole input space

Instead of learn a **fixed** vector for a given sample x, we learn **a distribution** on a latent space. (How samples are distributed in a lower dimension latent space)

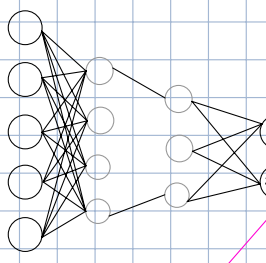
- $P_{\text{prior}\,\theta}(z)$ Prob. of latent space
- $\text{Likelihood}_\theta(x|z)$ Prob. of getting a sample x from a latent variable z)
- $\text{Posterior}_\theta(z|x)$ Prob. of obtaining the latent variable z from x sample



Again, we want to learn the distribution of the input data. We do it by observe the input and use it to estimate the parameters of another latent variable z.



f is a NN



Let's suppose z is a single dimension Gaussian variable. We then Estimate the parameter μ and σ^2 by the use of a NN

we choose this $\log \sigma^2$ then to $2 \log \sigma$ to have the output of the net to be a positive Real Number

$$z \sim N\left(\mu, \exp(0.5 \cdot 2 \log \sigma)\right)$$

THIS IS GIVING THE PREDICTION $q_\phi(z|x)$

How do we learn the weight of f

We need a Loss which should measure the DISTANCE between our prob. distribution of the input, w.r.t. to z.

We may use reverse KL Divergence or (Expected value of log likelihood)

$$D_{KL}(q_\phi \,||\, p_\theta) = \mathbb{E}_{q_\phi} \log \left[\frac{q_\phi(z|x)}{p_\theta(z|x)} \right]$$

(q_ϕ: prediction; p_θ: ground truth prob. distribution)

$$= \int_{\mathbb{R}} q_\phi(z|x) \log \left[\frac{q_\phi(z|x)}{p_\theta(z|x)} \right]$$

this is the GT but we do not have this

For this reason we use

ELBO Evidence Lower Bound Is the loss of the variational Autoencoder

Derivation of the loss starting from KL Divergence

$$D_{KL}(q_\phi \,||\, p_\theta) = \mathbb{E}_{q_\phi}\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)} \right]$$

Just rewrite the fraction as subtraction using log property

$$= \mathbb{E}_{q_\phi}[\log q_\phi(z|x)] - \mathbb{E}_{q_\phi}[\log p_\theta(z|x)]$$

By Bayes Rule $p(z|x) = \frac{p_\theta(x|z)\, p_\theta(z)}{p_\theta(x)} = \frac{p_\theta(z,x)}{p_\theta(x)}$

$$= \mathbb{E}_{q_\phi}[\log q_\phi(z|x)] - \mathbb{E}_{q_\phi}\left[\log \frac{p_\theta(z,x)}{p_\theta(x)}\right]$$

Some separation as Before

$$= \mathbb{E}_{q_\phi}[\log q_\phi(z|x)] - \mathbb{E}_{q_\phi}[\log p_\theta(z,x)] + \mathbb{E}_{q_\phi}[\log p_\theta(x)]$$

$$= \mathbb{E}_{q_\phi}[\log q_\phi(z|x)] - \mathbb{E}_{q_\phi}[\log p_\theta(z,x)] + \int_{\mathbb{R}} q_\phi(z|x) \cdot \log p_\theta(x)\, dz$$

Moved out because doesn't depend on z

$$= \mathbb{E}_{q_\phi}[\log q_\phi(z|x)] - \mathbb{E}_{q_\phi}[\log p_\theta(z,x)] + \log p_\theta(x) \int_{\mathbb{R}} q_\phi(z|x)\, dz$$

$\int_{\mathbb{R}} q_\phi(z|x)\, dz = 1$

$$= \mathbb{E}_{q_\phi}\left[\log q_\phi(z|x)\right] - \mathbb{E}_{q_\phi}\left[\log p_\theta(z,x)\right] + \log p_\theta(x)$$ Marginal Log likelihood Evidence

We cannot do nothing then we rewrite every thing like follows

Marginal log likelihood

$$\log p_\theta(x) = \mathbb{E}_{q_\phi}\left[\log q_\phi(z|x)\right] + \mathbb{E}_{q_\phi}\left[\log p_\theta(z,x)\right] + D_{KL}\left(q_\phi \| p_\theta\right)$$

if This is MAXIMIZED this has to be MINIMIZED to hold the equality

we know that this ≥ 0

Let's reason a Bit on this

Then we can say that at minimum we can have

$$\log p_\theta(x) \geq \mathbb{E}_{q_\phi}\left[\log q_\phi(z|x)\right] + \mathbb{E}_{q_\phi}\left[\log p_\theta(z,x)\right]$$ Joint prob of x

We call this ELBO

$$\text{ELBO} = -\mathbb{E}_{q_\phi}\left[\log q_\phi(z|x)\right] + \mathbb{E}_{q_\phi}\left[\log p_\theta(x|z)\right] + \mathbb{E}_{q_\phi}\left[\log p_\theta(z)\right]$$

$$= \mathbb{E}_{q_\phi}\left[\log p_\theta(x|z)\right] - \mathbb{E}_{q_\phi}\left[\log \frac{q_\phi(z|x)}{p_\theta(z)}\right]$$

Expected reconstruction error — KL Divergence between posterior and prior

Using Elbo we want to maximize it

$$\text{ELBO} = \mathbb{E}_{q_\phi}\left[\log p_\theta(x|z)\right] - \mathbb{E}_{q_\phi}\left[\log \frac{q_\phi(z|x)}{p_\theta(z)}\right]$$

We have this

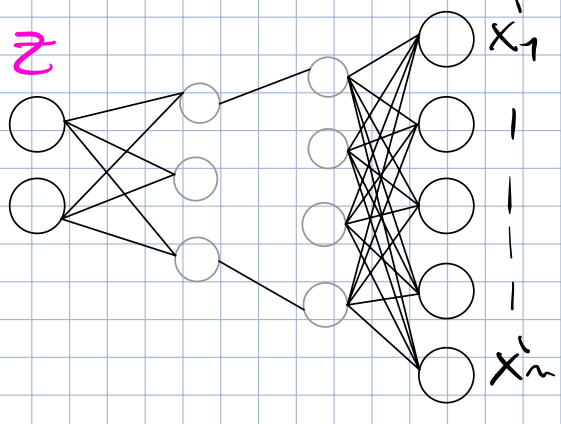
We have this

Here we have problemes

Again we can apply the same reason of before, we try to find the parameter for a function that allows to obtain X giving as input Z.

$z = \text{encoder}(x)$

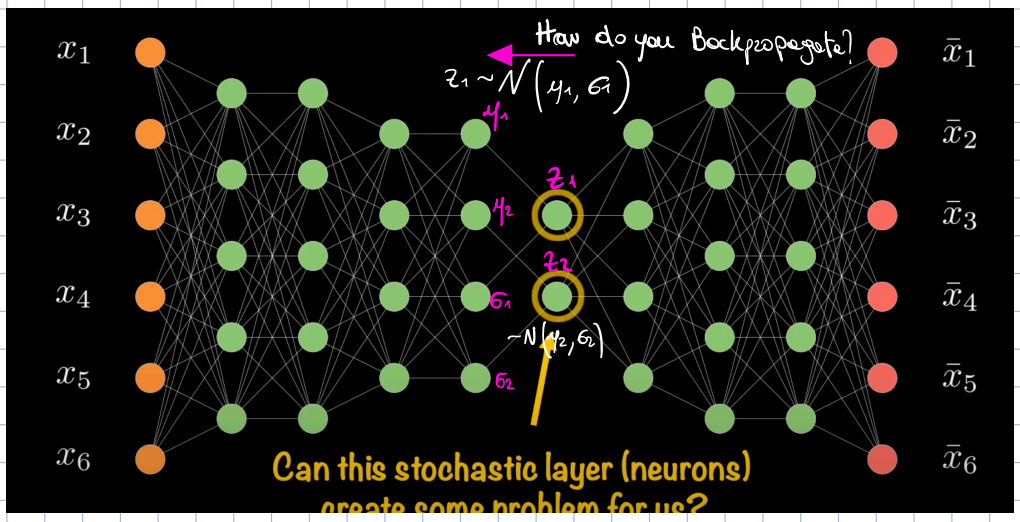
$x' = \text{decoder}(z)$



with x' we can now substitute this term just with

$$MSE(x, x')$$

REPARAMETRIZATION TRICK



Remember the Elbo Loss

$$\mathcal{L}_{\phi,\theta}(x) = \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p_\theta(z)}\right]$$

θ params of decoder

ϕ parameters of the encoder

this is the set that contains μ, σ for z

Let's rewrite some stuff

$$= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z) - \log \frac{q_\phi(z|x)}{p_\theta(z)}\right]$$

$$= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z) - \log q_\phi(z|x) + \log p_\theta(z)\right]$$

$$= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x,z) - \log q_\phi(z|x)\right]$$

And now we can take gradient WRT ϕ and θ, for θ ok, for ϕ we cannot because it's not possible to apply the Liebniz Rule

Change of variable to sample from a distrib.

$x \sim p_\theta(X)$ and we take $x = g(\varepsilon, \theta)$ where $\varepsilon \sim p(\varepsilon)$

Example $p_\theta(x) = \mathcal{N}(x; \mu, \sigma)$ where $\theta = \{\mu, \sigma\}$

Sample as $x = \mu + \sigma \cdot \varepsilon$ where $\varepsilon \sim p(\varepsilon)$ Base distribution

How do we sample from the Base Distribution?
How the Base Distrib. is caracterized?

$$p(\varepsilon) = \mathcal{N}(\varepsilon, 0, 1)$$

Now Just a Remainder of Lotus

$$\mathbb{E}_{p_\theta(x)}[f(x)] = \int f(x)\, p_\theta(x)\, dx$$

then because $x \sim p_\theta(x)$ and $x = g(\varepsilon, \theta)$ where $\varepsilon \sim p(\varepsilon)$ then

$$\nabla_\theta \mathbb{E}_{p_\theta(x)}[f(x)] = \nabla_\theta \int f(g(\varepsilon,\theta))\, p(\varepsilon)\, d\varepsilon \quad \} \text{ Lotus}$$

$$= \int \nabla_\theta f(g(\varepsilon,\theta))\, p(\varepsilon)\, d\varepsilon \quad \} \text{ Leibniz}$$

this is the obj of expectation of Continuous RV

$$= \mathbb{E}_{p(\varepsilon)}[\nabla_\theta f(g(\varepsilon,\theta))]$$

by stacos the Batch

$$\simeq \frac{1}{N} \sum_i^N \nabla_\theta f(g(\varepsilon^i,\theta)) \qquad \varepsilon^i \sim p(\varepsilon)$$

Now we can do the same on ELBO

$$\text{ELBO} = \mathcal{L}_{\phi,\theta}(x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x,z) - \log q_\phi(z|x)]$$

Set $\varepsilon \sim p(\varepsilon)$ sample from a base distr.
$z = g(\phi, x, \varepsilon)$ deterministic funct of ϕ (differentiable) by encoder
} Here is applied change of variable

Using Lotus

$$\mathcal{L} = \mathbb{E}_{\varepsilon \sim p(\varepsilon)}[\log p_\theta(x,z) - \log q_\phi(z|x)]$$

Thanks to Leibniz Rule we move on the gradient

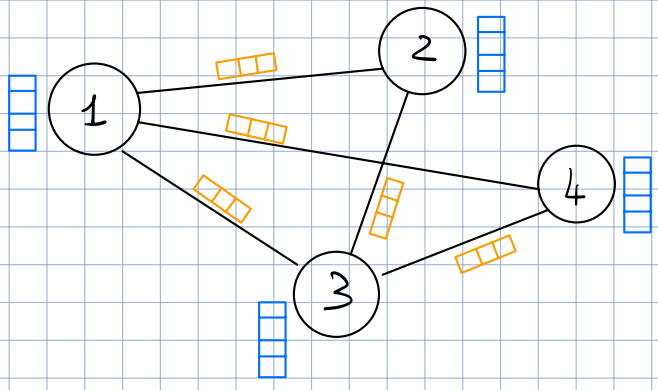
$$\nabla_\phi \mathcal{L}_{\theta,\phi}(x) = \nabla_\phi \mathbb{E}_{p(\varepsilon)}[\log p_\theta(x,z) - \log q_\phi(z|x)]$$

CAPITOLO 10

GRAPH NEURAL NETWORKS

- GNN Basic
- Tools
- Graph CNN
- Batching
- Aggregation
- Spectral Graph
- Graph laplacian
- Spectral Graph convolution
- Chebnet

GNN Basic

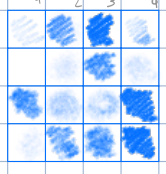


To describe a Graph G we need 3 matrices

- A Adiacency Matrix
  $|v| \times |v|$ $i,j = 1$ if node i is linked to node j

	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

- X Node Feature Matrix
  $d^0 \times |v|$ numb of nodes × d^0 numb of feature



- E Edge feature Matrix
  $d^e \times |E|$ dim of feature edge × Numb of Edges

PERMUTATION OF NODES → P Matrix that change the ordering of node

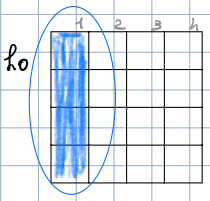
$$X' = PX$$
$$A' = P^T A P$$

GNN Layer

Node feature

$$h_0 = \text{Layer}(X)$$

Each column represent the hidden representation of a node



TASKS

Graph Classification

$$P_2(y = 1 \mid X, A) = \text{SIG}\left[\beta_k + W_k H_k \mathbb{1}/N\right]$$

- β_k: scalar
- H_k: last hidden layer
- $\mathbb{1}$: column vector 1
- $1 \times D$

P_2 Probability of graph to be classified as 1 given node feature Matrix And Adj matrix

this gives a single scalar

Node Classification

$$P_2(y^m = 1 \mid X, A) = \text{SIG}\left(\beta_k + W_k h_k^m\right)$$

Just consider the hidden state of the node to be classified (m)

Edge Classification

$$P_2(y^{mn} = 1 \mid X, A) = \text{SIG}\left[h^m \; h^{n\,T}\right]$$

Make use of the emb. of nodes

GRAPH CNN

Graph Convolutional Layer

$$H_1 = F(X, A, \phi_0)$$

$$H_i = F(H_{i-1}, A, \phi_{i-1})$$

(Suppose Layer i)

H_{i-1} depends on previous hidden state and previous set of ϕ

$$H_k = F(H_{k-1}, A, \phi_{k-1})$$

Then, A full network of GCL

$$H_1 = a\left[\beta_0 + W_0 X (A+I)\right]$$

$$H_i = a\left[\beta_{i-1} + W_i H_{i-1} (A+I)\right]$$

$$H_k = a\left[\beta_{k-1} + W_k H_{k-1} (A+I)\right]$$

GRAPH CLASSIF.

$$f(X, A, \phi) = a\left[\beta_k + W_k H_k \mathbb{1}/N\right]$$

AGGREGATION

NODE CLASS.

$$f[X, A, \phi] = \text{SIG}\left[\beta_k \mathbb{1}^T + W_k H_k\right]$$

$\forall$ node m, in Layer i we sum the aggregator $\text{AGG}[m, i] = \sum_{m \in N(e)} \tilde{h}_i^m$

We sum up all the hidden states of this Layer from the neighbours node of node m

How do we get $\tilde{h}_{i+1}$ (Hidden State of node m for layer $i+1$)

$$\tilde{h}_{i+1} = a\left[\beta_i + W_i \tilde{h}_i^m + W_i \text{AGG}[m, i]\right]$$

(Non linear act-funct.)

Proj. of current hidden State information — Aggregated informations projected

If we want to do it for all the nodes

$$H_{i+1} = a\left[\beta_i \mathbb{1}^T + W_i H_i + W_i H_k A\right]$$

($\mathbb{1}$: Column Vector; H_i: $|X| \cdot D$)

$$= a\left[\beta_i \mathbb{1}^T + W_i H_i (A+I)\right]$$

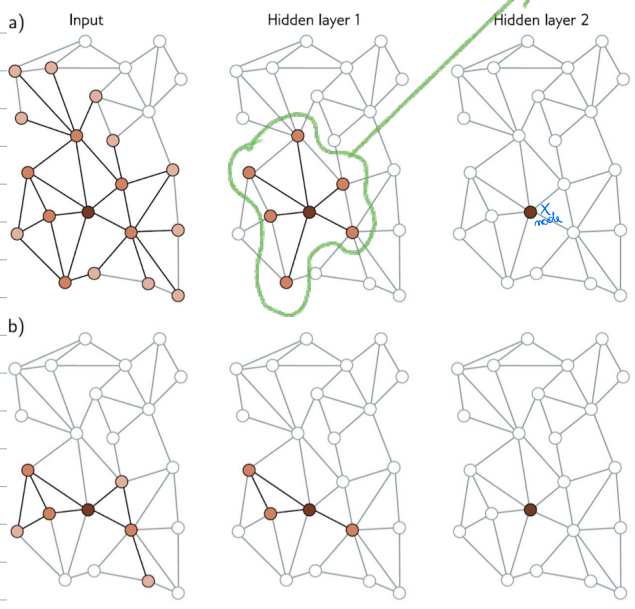
I: Identity

BATCHING

NODE SUBSET

Choose a random subset of nodes $\forall$ training step and then we select the neighbors from previus layers

One Layer News we are taking only the 1-hop Nodes



$$h_1^x = a\left[\beta_0 + w_0 x + w_1 \text{Agg}(x, 1)\right]$$

$\mathcal{O}2$ a random subset of $N(x)$

$$\sum_{m \in N(x)} h_m$$

$$h_2 = a\left[\beta_1 + W_0 h_1 + w_1 \text{Agg}(x, 2)\right]$$

Another Mechanism is the GRAPH PARTITIONING

AGGREGATION MECHANISM

Summation

$$H_{k+1} = a\left[\beta_k 1^T + W_k H_k (A + I)\right]$$

Diagonal Enhancement

$$H_{k+1} = a\left[\beta_k 1^T + W_k H_k \left(A + (1+\varepsilon_k) I\right)\right]$$

$$Agg(x, i) = \sum_{m \in N(x)} (1+\varepsilon_i)\, h_i^m$$

Each node is multiplied by a factor of $(1+\varepsilon_k)$ before contributing to the sum, ε_k is learnt and differ for each layer

Generalized Diagonal Enhancement

$$H_{k+1} = a\left[\beta_k 1^T + \underbrace{\Omega_k H_k A}_{Agg} + W_k H_k\right]$$

$$\Omega'_k \left[\begin{matrix} H_k A \\ H_k \end{matrix}\right] \qquad [\Omega_k \; W_k]$$

Different weights for the emb. of the current Node

Residual Connection

$$H_{k+1} = \left[a\left[\beta_k + W_k H_k A\right], H_k\right]$$

1. Aggregate, linear transf., activation
2. Conc. to the projection of current node

Mean Aggregation

$$Agg(x, i) = \frac{1}{|N(x)|} \sum_{m \in N(x)} h_m$$

$$H_{k+1} = a\left[\beta_k 1^T + W_k H_k \left(AD^{-1} + I\right)\right]$$

Smooth the contr. of aggregation, less informat. about the structure of the graph

KIPF Normaliz.

$$AGG(x, i) = \sum_{m \in N(x)} \frac{h_m}{\sqrt{|N(x)| \cdot |N(m)|}}$$

Smooth the contribution of the neigh. nodes that have a lot of neighbors. (They have less unique information)

$$H_{k+1} = a\left[\beta_k 1^T + W_k H_k \left(D^{-1/2} A D^{-1/2} + I\right)\right]$$

MAX Pooling

$$AGG(n, i) = \max_{m \in N(x)} h_m$$

Take the max embedding of the neighbours.

Attention in Aggregation

Get a projection of current node embeddings

$$H'_k = \beta_k 1^T + W_k H_k$$

Then consider similarity score $s_{xm} = a\left[\phi_k^T \left[\begin{matrix} h'_x \\ h'_m \end{matrix}\right]\right]$

S is a $N \times N$ Matrix of similarity Scores

$$H_{k+1} = a\left[H'_k \cdot Softmax(S, A + I)\right]$$

SPECTRAL GRAPH CONV. FRAMEWORK

A GRAPH CONVOLUTIONAL LAYER is composed of

- **PATCH FUNCTIONS**

Shape of the convolutional filter: selection of the node that have to interact with each other in each layer. Matrix $|N| \times |N|$, one $\forall$ layer i $\forall i$ $1 \dots k$

$$|N| \times |N| \rightarrow \left(f_1(W, H^\ell) \dots f_K(W, H^\ell)\right)$$

Matrix of node features at layer ℓ

k can be:
- Spatial: defining the number of neighbors
- Spectral: with the rank of matrix

- **CONVOLUTION FILTERS WEIGHTS**

$\forall$ layer we have k matrices $d_\ell \times d_{\ell+1}$ $(\Theta^\ell_1 \dots \Theta^\ell_k)$ and this are convolved with every patch H^ℓ

$$m_k^{\ell+1} = f_k(W, H^\ell) H^\ell \Theta_k^\ell \quad \text{for} \quad 1 \le k \le K$$

- **MERGING FUNCTIONS**

Combine the output of multiple convolution into one single representation

$$H^{\ell+1} = h(m_1^{\ell+1}, \dots, m_K^{\ell+1})$$

Method	Model	$g_k(.)$	$h(m_1, \dots, m_k)$
Spectrum-based: $\tilde{L} = U \Lambda U^\top$, $f_k(W, H) = g_k(U)$	SCNN	$g_k(U) = u_k u_k^\top$	$\sigma(\sum_k m_k)$
Spectrum-free:	ChebNet	$g_k(W, D) = T_k(\frac{2(I - D^{-1/2} W D^{-1/2})}{\lambda_{max}(I - D^{-1/2} W D^{-1/2})} - I)$	$\sigma(\sum_k m_k)$
$f_k(W, H) = g_k(W, D)$	GCN	$g_1(W, D) = (D + I)^{-1/2}(W + I)(D + I)^{-1/2}$	$\sigma(m_1)$
Spatial:	SAGE-mean	$g_1(W, D) = I, g_2(W, D) \sim \mathcal{U}_{\text{norm}}(D^{-1} W, q)$	$\sigma(m_1 + m_2)$
$f_k(W, H) = g_k(W, D)$	GGNN	$g_1(W, D) = I, g_2(W, D) = W$	GRU$(m_1, m2)$
Attention:	MoNet	$g_k(U^s) = \exp(-\frac{1}{2}(U^s - \mu_k)^\top \Sigma_k^{-1}(U^s - \mu_k))$	$\sigma(\sum_k m_k)$
$f_k(W, H) = \alpha(W \cdot g_k(H))$	GAT	$g_k(H) = \text{LeakyReLU}(HB^\top b_0 \oplus b_1^\top B H^\top)$	$\sigma([m_1 \| \dots \| m_k])$

GRAPH LAPLACIAN

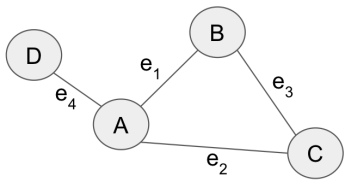
- Given a graph G:=(V,E)
  - V:={v_1,v_2,…,v_n} is the set of nodes/vertices
  - E:={e_1,e_2,…,e_m} is the set of edges
  - adjacency matrix $A \in \{0,1\}^{n \times n}$

$$A_{ij} := \begin{cases} 1, & \text{if there is an edge between } v_i \text{ and } v_j \\ 0, & \text{otherwise} \end{cases}$$

- and degree matrix $D \in \mathbb{Z}^{n \times n}$

$$D_{ij} := \begin{cases} \text{degree}(v_i), & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

Graph Laplacian = D − A



Functions map each point into Real number $f: V \to \mathbb{R}$

$f = [f(v_1) \dots f(v_m)]$

Define the notion of:
- point
- gradient
- functions
- divergencies

in the context of graphs

Point Each vertex $v \in V$ is a point

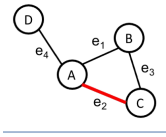
Gradient It is the easy difference of f for 2 nodes $e_k = (v_i, v_s) \in E$ edges

$g(e_k) = f(v_i) - f(v_s)$ $i < j$ and with ordered edges we can write $g = [g(e_1) \dots g(e_m)]$

How to do this in a Matrix Form? Incidence matrix K.

$$K_{ij} := \begin{cases} 1, & \text{if } i == k \text{ and } i > h \\ 1, & \text{if } i == h \text{ and } i > k \\ -1, & \text{if } i == k \text{ and } i < h \\ -1, & \text{if } i == h \text{ and } i < k \\ 0, & \text{otherwise} \end{cases}$$

Graph G



Incidence Matrix K

	A	B	C	D
e_1	1	−1	0	0
e_2	1	0	−1	0
e_3	0	1	−1	0
e_4	1	0	0	−1

Then we can compute

$$\underset{K}{\begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & -1 \end{bmatrix}} \underset{f}{\begin{bmatrix} f(A) \\ f(B) \\ f(C) \\ f(D) \end{bmatrix}} = \underset{g}{\begin{bmatrix} f(A) - f(B) \\ f(A) - f(C) \\ f(B) - f(C) \\ f(B) - f(D) \end{bmatrix}}$$

Divergence given node A Consider divergence $(A) = g(e_1) + g(e_2) + g(e_4)$

Or by matrix operation as

$$\underset{K^T}{\begin{bmatrix} 1 & 1 & 0 & 1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}} \underset{g}{\begin{bmatrix} g(e_1) \\ g(e_2) \\ g(e_3) \\ g(e_4) \end{bmatrix}} = \underset{d}{\begin{bmatrix} g(e_1) + g(e_2) + g(e_4) \\ -g(e_1) + g(e_3) \\ -g(e_2) - g(e_3) \\ -g(e_4) \end{bmatrix}}$$

The Laplacian is thus $L := K^T K$

$K^T K = D - A$

CAPITOLO 11

GEOMETRIC DEEP LEARNING

- Group & Actions
- Geom. dep. learning
- Graph NN, Equivariance
- Deep Sets
- Group Equivariant

Branch that aims to develop geometric invariant learning

Euclidean : function defined over a given number of coordinates (x, y, z)

In NON Euclidean : the function is not defined over the same number of coordinate $\forall$ input of the function
In Graph we have node A linked with 2 nodes (2 coordinates)
and node B linked with 3 coordinates.

We want invariance also in this type of Spaces

Goal of Geometric Deep Learning : build representation of object defined by a particular structure / geometry.

We define a Geometry as the set of possible transformation that applied over objects, preserve certain property

Euclidean Geometry → the set of transformations that preserve area, angles, distance

Given a certain type of data

→ ① Identify the invariant property (translation, rotation, simmetry ...) SYMMETRY PRIOR

→ ② Create a model which is robust to a given invariant (maintain a given invariant)
(choose or optimize)

→ Leverage the INDUCTIVE BIAS

Given the basic operation in a Deep Learning model : input is a vector, layer is a Matrix of weights

We can describe transformation (this matrices) as element of a Group
Representation $\rho : \rho_V : G \to \text{General Linear Group}(V)$
I need a tool that takes an element of this group and transform it into a Matrix

Invariance
Given a Group of transformation G
I have representation of the element in G $\rho(G)$
a function f is Invariant to G if

$$f(\rho(G)x) = f(x)$$

the outcome of f is the same both if x is not transformed or is transformed by an element of the group $f(\rho(G)x)$

Equivariance
f is Equivariance if

$$f(\rho(G)x) = \rho(G)f(x)$$

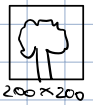
Transform input and get repres = transform the representation

Question: Can we emulate the receptive field mechanism of the convolution?

Hence, can we look at features of the input at a different SCALES (different levels of granularity) ?

The Local stability is the property of preserving a certain type of feature at different scales

Example

$f\left(\text{  }\right) = \text{Tree}$ (100x100 px) $f\left(\text{  }\right) = \text{Tree}$ (200x200)

The TREE Property is locally stable in f

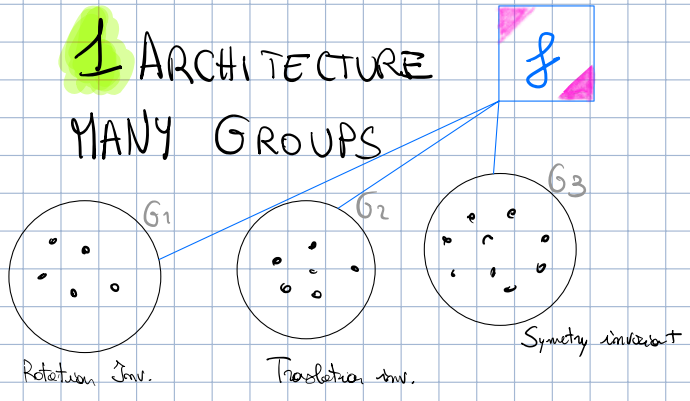
GEOMETRIC DEEP LEARNING

Input x → Equivariant layer $f(\varphi(G)\,x) = \varphi(G)\,f(x)$ → Local Pooling (is an aggregation function) → Equivariant layer → Single vector Representation

this is why you { take an Elent of the group of trasformation
ASSUME A CERTAIN GEOMETRY OF THE INPUT.
This is because then you need the group of certain trasformation adapt for certain geometry

What's the advantage ?

1 ARCHITECTURE MANY GROUPS



$$f(\varphi(G_i)\,x) = \varphi(G_i)\,f(x)$$

φ: A function that choose a certain element of a given group

Ideal characteristic of GNN

2 isomorfic graphs → Same representation
2 non isomorphic → Different represent.

To make a graph at as power as WL test
the aggregation funct should be

$$h_v^{(k)} = \text{MLP}^{(k)}\left(\left(1+\varepsilon^{(k)}\right)h_v^{(k-1)} + \sum_{i\in N(v)} h_i^{(k-1)}\right)$$

Self loop Weight

Multilayer Perc.

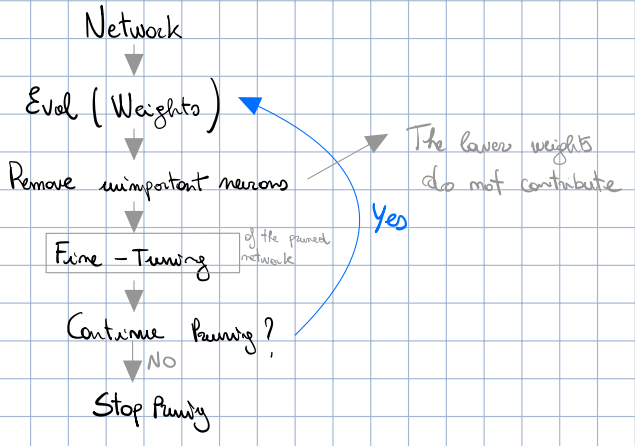
This makes it injective

CAPITOLO 12

- Pruning NN
- Lottery Ticket Hypotesis
- Weight Quantization

MODEL COMPRESSION

PRUNING



Given $f(x, \theta)$ init weight $\theta = \theta_0$

Train f and it gets loss ℓ at step J accuracy a

Now take $f(x, m \odot \theta)$ with init $\theta = m \odot \theta_0$

then you get ℓ_0 at J_0 with accuracy a_0

Lottery ticket: $\exists\, m \;/\; J_0 < J,\ a_0 \geq a,\ \|m\| < |\theta|$

The algorithm works as follows:

- Randomly initialize a neural network $f(x; \theta_0)$ (where $\theta_0 \sim D_\theta$).
- Train the network for j iterations, arriving at parameters θ_j.
- Prune p% of the parameters in θ_j, creating a mask m.
- Reset the remaining parameters to their values in θ_0, creating the winning ticket $f(x; m \odot \theta_0)$.

SAPIEN UNIVERSITÀ DI

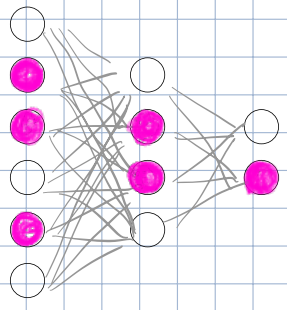
LOTTERY TICKET HYPOTESIS

Net (original) → X% of accuracy
Net (pruned) → Y% of accuracy with $Y < X$

The retraining idea
Smaller pruned network perform worse than the original network because of course they are less powerful.

The Lottery ticket hypotesis

A random initialized network contains a subnetwork (using the original weight initializ) such that if you just train this small network you can get the same performa of the big original network



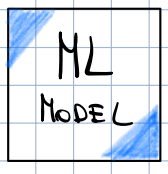
CAPITOLO 13

- Meta Learning Problem, Few Shot
- Supervised VS meta learning
- Metric - Based Meta Learning
- Model - Based Meta Learning
- Optimization - Based Meta Learning
- Learning to Learn
- MAML

META LEARNING

Design a ML model with human learning abilities:
learning new concept and tasks with few training examples.

Meta Learning in Supervised Learning



TASK 1 — Dataset $D_1 = \{(x, y)\}_{i=1}^{|D_1|}$
TASK 2 — D_2
⋮
TASK N — D_N

Task N+1
Task N+2
} The inference happens on an unseen task on unseen data

Learning will be

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\ \mathbb{E}_{D \sim p(D)}\left[\mathcal{L}_\theta(D)\right]$$

Find the best θ by optimize over a distribution of task

FEW - SHOT LEARNING K - SHOT N - CLASS

Meta learning supervised learning where each Dataset $Di = \langle S, B \rangle$

S is the subset of Di of (k) sample used for training

N classes tells the number of Dataset available

In terms of likelihood

$$\theta^* = \operatorname{argmax}_\theta\ \mathbb{E}_{(x,y)\in D}\left[P_\theta(y|x)\right]$$

with minibatches $\operatorname{argmax}_\theta\ \mathbb{E}_{B\in D}\left[\sum_{(x,y)\in B} P_\theta(y|x)\right]$

SUPERVISED VS META - LEARNING

$$\theta = \operatorname*{argmax}_\theta\ \mathbb{E}_{L\in\mathcal{L}}\left[\mathbb{E}_{S^L\in D,\ B^L\in D}\left[\sum_{(x,y)\in B^L} P_\theta(x, y, S^L)\right]\right]$$

L: a subset of the total labels across the whole dataset
S^L: Support set with samples with labels in L
B^L: Batch
Given a sample from batch use the support set S^L to compute the likelihood

LEARNER & META - LEARNER

LEARNER → f_θ : Model to operate for a given task

META - LEARNER → g_ϕ : Model that learns how to update the learner's params using the S^L

$$\theta' = g(\theta, S^L)$$

The final optimization would be

$$\mathbb{E}_{L\in\mathcal{L}}\left[\mathbb{E}_{S^L\in D,B^L\in D}\left[\sum_{(x,y)\in B^L} P_{g_\phi(\theta,S^L)}(y|x)\right]\right]$$

Given the problem, we can now look at the approaches
How $P_\theta(y|x)$ is modeled

METRIC - BASED	$\sum_{(x_i,y_i)\in S^L} k_\theta(x, x_i)\cdot y_i \quad (*)$
MODEL - BASED	$P_\theta(x, S^L)$
GRADIENT - DESCENT	$P_{g_\phi(\theta,S^L)}(y|x)$

k_θ: kernel function that actual similarity

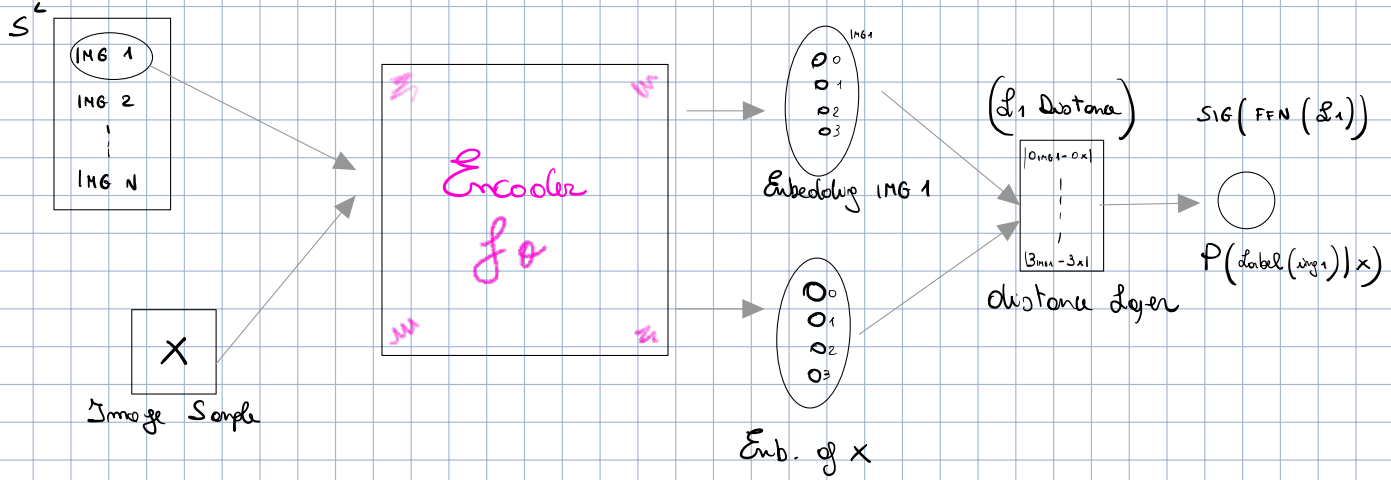
METRIC - BASED META-LEARNING

$$P_\theta(y|x) = \sum_{(x_i, y_i) \in S^L} k_\theta(x, x_i) \cdot y_i$$

The probability of y given a sample as the weighted sum of a kernel function the measure the similarity between x, x_i (element of S^L) as weight

Then to implement this we need
1. Learn the embedding of samples
2. Learn how to compute the similarity with the kernel's parameters

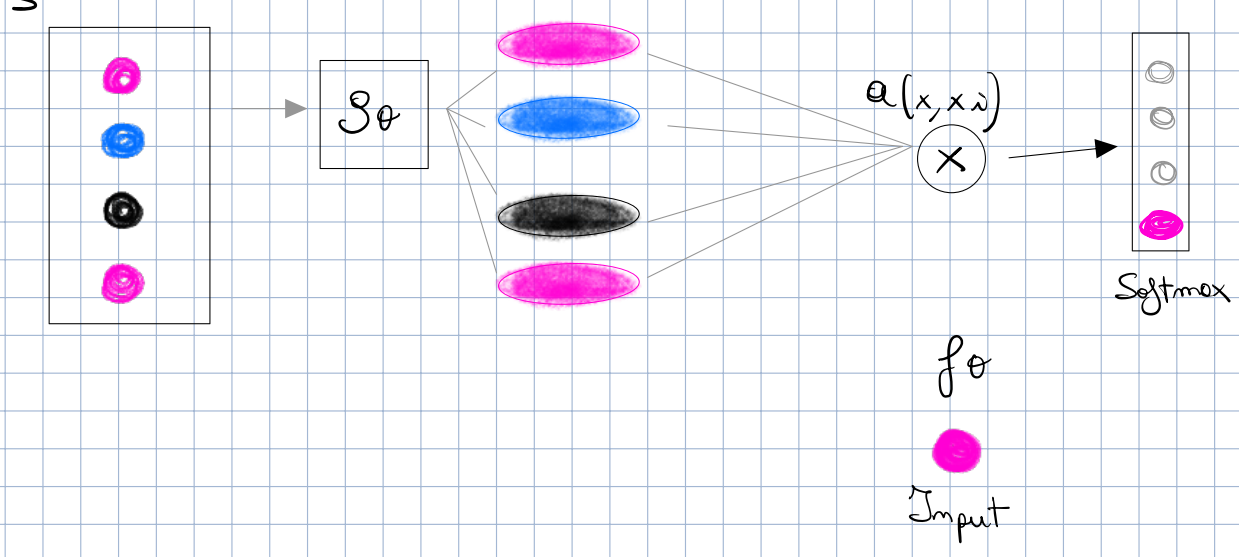
1 CONVOLUTIONAL SIAMESE NEURAL NETWORKS



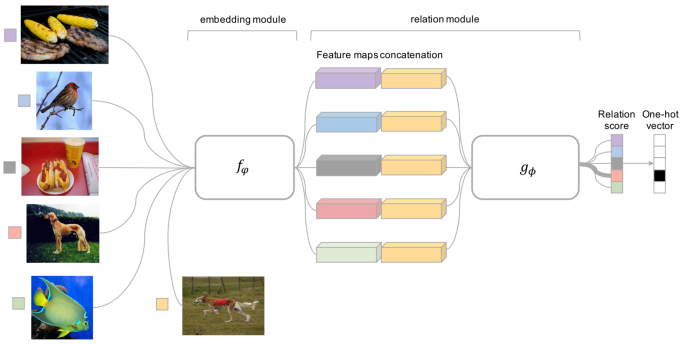
Prediction $\text{class}_{S^L}(x) = \text{class}\left(\underset{x_i \in S}{\text{argmax}}\ P(x, x_i)\right)$

2 MATCHING NETWORKS

Prediction $\text{class}_{S^L}(x) = P(y|x, S^L) = \sum_{i=1}^{k} a(x, x_i)\, y_i$ (k: num of classes)

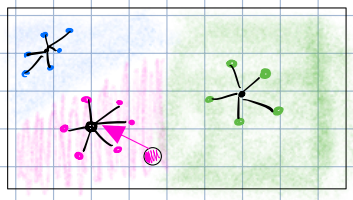


3 RELATION NETWORK



4 PROTOTYPICAL NETWORKS

Compute embeddings of class samples in S^k take the mean

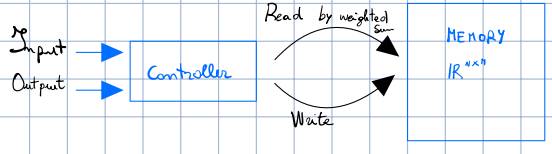


Centroid for class c $\quad v_c = \frac{1}{|S^c \in S^k|} \sum_{(x_i, y_i) \in S^c} f_\theta(x_i)$

MODEL BASED META-LEARNING

START FROM NEURAL TURING MACHINES

2 Components : CONTROLLER (NN) $\quad$ MEMORY $(N \times M)$ Matrix



Read by weighted Sum

$$r_t = \sum_{i=1}^{N} \underbrace{w_t(i)}_{\text{attention Score}} M_t(i)$$

the sum of the attention over the memory = 1

$$\sum_{i=1}^{N} w_t(i) = 1 \qquad \forall i \quad 0 \le w_t(i) \le 1$$

Write Like LSTM

① Erase some content from M according to an erase vector

$$\tilde{M}_t(i) = M_{t-1}(i) \left[1 - w_t(i) e_t \right]$$

② Add Some content by an add vector

$$M_t(i) = \tilde{M}_t(i) + w_t(i) a_t$$

OPTIMIZATION - BASED META LEARNING

Gradient - Descent needs → A lot of samples to learn
→ A lot of steps to converge

- How can a GD approach work with few examples ?

The Meta - Learner Approach

Learning Process → composed of → LEARNER M_θ (Network that execute the specific task) → Loss Function
composed of → META - LEARNER R_ϕ → Loss Function

LSTM - MODEL
Has the role of update θ parameters (learners params) using a small support set such that the learner lead to high performance

the meta learner introduce a function $g_t\left(\nabla_t(\theta_t), \phi\right)$ this function depends on the gradient of the learner and the parameter of the meta learner itself

Update the learner by GD using the g function to define how to actually update the weights

$$\theta_{t+1} = \theta_t + g_t\left(\nabla_t(\theta_t), \phi\right)$$

Recurrent neural Net

This means that the (best) params of the learner θ^* Depends on the parameters of the learner itself but also on the parameter of the meta learner ϕ

$$\theta^*(f_\theta, \phi)$$

How we define the goodness of the meta - learner ?

Considering its loss $\mathcal{L}(\phi) = \mathbb{E}_f\left[f\left(\theta^*(f, \phi)\right)\right]$

This means that the good meta learner is the one that give a low loss computed by the outcome that the learner function gives when makes use of the best parameters θ^* that depends on f and ϕ

g_t is an RNN parametrized by ϕ
in RNN we have

- h_t : HIDDEN STATES
- trajectory lenght : history T (horizon) of evolution of gradient of f

$$\mathcal{L}(\phi) = \mathbb{E}_f\left[\sum_{t=1}^{T} w_t f(\theta_t)\right]$$

w_t → arbitrary weight for time t

update $\theta_{t+1} = \theta_t + g_t$

and $\begin{bmatrix} g_t \\ h_{t+1} \end{bmatrix}$ = meta learner Model $\left(\nabla_t, h_t, \phi\right)$

the current gradient of f → $\nabla_\theta f(\theta_t)$

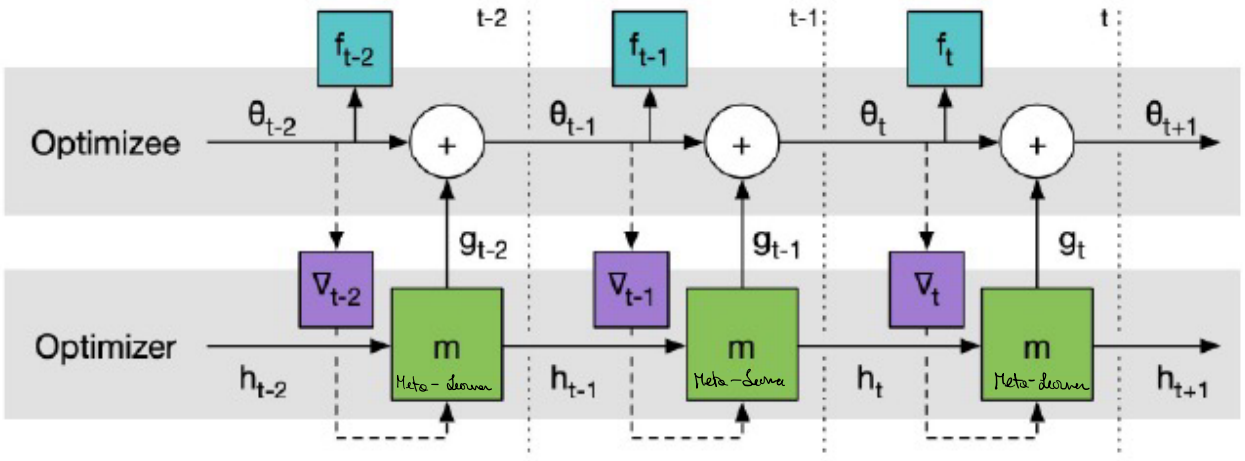


Figure 2: Computational graph used for computing the gradient of the optimizer.

MAML : Model Agnostic Meta-Learner

Really simple idea : the parameters of the learner θ are updated using a batch of training for a given task τ_i. With f and a task τ_i we can update θ_i' (batch 0)

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\tau_i}^{0}(f_\theta)$$

θ_i' : the new set of parameter of the learner for the given task i

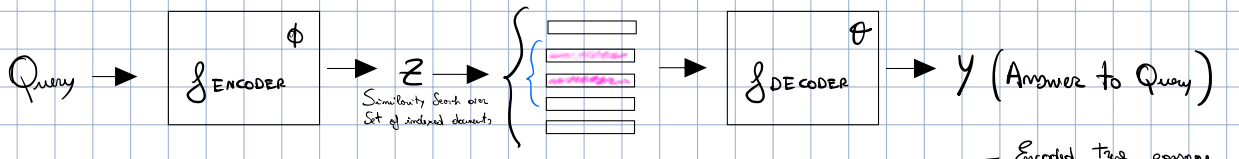
When all the θ_i' are computed for all the tasks, then we can update

$$\theta = \theta - \beta \nabla_\theta \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}(f_{\theta_i'})$$

CAPITOLO 14

Retrival Augmented Generation

- Make an LLM Acces external information



$$p_{RAG}(y|x) = \sum_{z \in Top\,k\ sim(\cdot,x)} p_\phi(z|x) \cdot p_\theta(y|x,z)$$

$$p_\phi(z|x) = \exp\left(d(z)^T \cdot f_\phi(x)\right)$$

$$\mathcal{L}_{RAG}\left(x, z^+, z_1^-, \dots z_J^-, \phi, \theta\right) = \frac{sim(x, z^+)}{sim(x, z^+) + \sum_{j=1}^{n} sim(x, z_j^-)}$$

(Encoded true passage)

DPR → Backprop $f_{ENCODER}$ (only for query), use a Frozen Generator
RAG → Backprop $f_{ENCODER}$ query, freeze doc encoder,
REALM → Doc Encoder and $f_{ENCODER}$ both trained

CAPITOLO 15

Deal with Noisy Label

SOURCE OF NOISE
- Human Error
- Ambiguity of the task
- Multiple correct Labels

The training procedure has as objective minimize the empirical risk (ERM) over the Dataset D

Minimize $R_D(g) = \mathbb{E}_D \mathbb{I}\left(y \neq \underset{i=\{1..C\}}{\operatorname{argmax}}\, g(x)_i\right)$

the best classifier will be

$\hat{g}^*_{m,\epsilon} \in \operatorname{argmin}_g \hat{R}_{D,\epsilon}(g) = \frac{1}{m}\sum_{i=1}^{m} \ell(g(x_i), y_i)$

ERM for noisy dataset ←

NOISE FORMALIZATION

$\tilde{D}: X \times \tilde{Y}$ $\quad \tilde{y} \in \{0,1\}^C$, $\quad \tilde{S} = \{(x_1, \tilde{y}_1) \dots (x_m, \tilde{y}_m)\} \sim \tilde{D}$

set of corrupted samples

with a noise Rate

$\eta(x) = P_2\left(\tilde{Y}=y \mid Y=y, X=x\right)$

$$\hat{R}_{D,\epsilon}(g) = \frac{1}{m}\sum_{i=1}^{m} \ell\left(g(x_i), \tilde{y}_i\right)$$

Quantify the noise with the NOISE TRANSITION MATRIX

observation = Noise transition Matrix × T(x) • Hidden truth

$$\begin{bmatrix} P_1(\tilde{Y}=1 \mid X=x) \\ \vdots \\ P_2(\tilde{Y}=C \mid X=x) \end{bmatrix} = \begin{bmatrix} P_2(\tilde{Y}=1 \mid Y=1, X=x) & \dots & P_2(\tilde{Y}=1 \mid Y=C, X=x) \\ \vdots & & \\ P_2(\tilde{Y}=C \mid Y=1, X=x) & \dots & P_2(\tilde{Y}=C \mid Y=C, X=x) \end{bmatrix} \cdot \begin{bmatrix} P(Y=1 \mid X=x) \\ \vdots \\ P(Y=C \mid X=x) \end{bmatrix}$$

$$T(x) = \begin{bmatrix} 0.8 & 0.1 & 0.1 \\ 0.2 & 0.7 & 0.1 \\ 0.1 & 0.2 & 0.7 \end{bmatrix}$$

APPROACHES TO DEAL WITH NOISE

- Estimate the Transition Matrix
- Robust Loss
- Sample Weighting / Sample Selection
- Ensemble
- Meta-Learning

Classifier trained on noisy dataset lead to the same error of the classifier trained on clean data

- GENERALIZED CROSS ENTROPY

$$L_q\left(g(x), e_j\right) = \left(1 - g_j(x)^q\right)$$

e_j: one hot of class; $g_j(x)$: j-th component of the output prob. distribution

- EARLY-LEARNING REGULARIZ. LOSS

$$\ell_{ELR} = \ell_{CE} + \frac{\lambda}{m}\sum \log\left(1 - \langle p_i, q_i \rangle\right)$$

m: steps in training; p_i: prediction target sample i

- Regularization

Mix-Up Data Augment

$x = \lambda x_1 + (1-\lambda) x_2$ overlap images and

$y = \lambda y_1 + (1-\lambda) y_2$ labels

$\lambda \cong 0.5$

- Ensemble

More than one network

- Early Learning + Memorization

Learning easy pattern (clean samples) ↓

The model starts to memorize the noise

Sample Selection

- Co-Teaching (2 Models w_f w_g)
  - Get Loss of samples for model w_f and model w_g
  - Select Sample with loss $< R(T)$ $\tilde{D}_f$ and $\tilde{D}_g$
  - Update $w_f = w_f - \alpha \nabla \ell(f, \tilde{D}_g)$

    $w_g = w_g - \alpha \nabla \ell(f, \tilde{D}_f)$
  - Update the threshold $R(T)$