Time series: TS x is a measure in time t $x = x_0, x_1, \dots, x_t, \dots, x_N$

Observation can be observable at irregular time intervals.

TS analysis assumes Weakly stationary data:

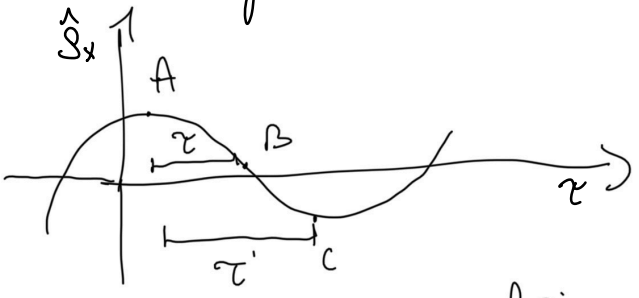
- $E[x_t] = \mu$ (doesn't change with time t)
- $\text{Cov}(x_{t+\tau}, x_t) = \gamma_\tau \quad \forall t$ (γ depends only on lag τ)
  ↳ lag between observations

Sample mean: $\hat{\mu} = \frac{1}{N} \sum_{t=1}^{N} x_t$

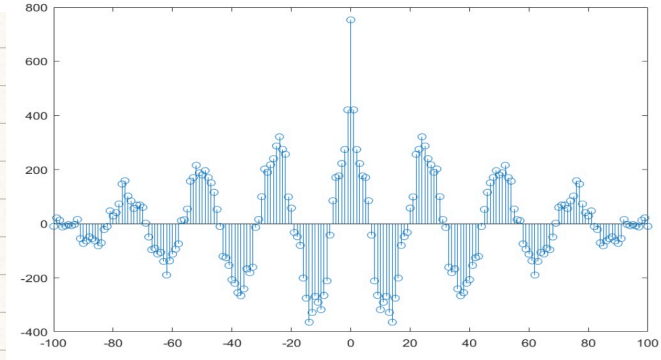
Autocovariance, for lag $-N \leq \tau \leq N$: $\hat{\gamma}_x(\tau) = \frac{1}{N} \sum_{t=1}^{N-|\tau|} (x_{t+|\tau|} - \hat{\mu})(x_t - \hat{\mu})$

Autocorrelation: $\hat{\rho}_x(\tau) = \frac{\hat{\gamma}_x(\tau)}{\hat{\gamma}_x(0)}$ ↳ signal without delay

Autocorrelation analysis can reveal repeating patterns such as periodic signals



τ' ha autocorrelazione massima perché il segnale si ripete ma combiato di segno



Autocorrelogram reveals a sine wave

Cross-Correlation: x^1, x^2 time series. Measure of similarity as a function of a time lag τ

$$\phi_{x^1x^2}(\tau) = \sum_{t=\max\{0,\tau\}}^{\min\{(T^1-1+\tau),(T^2-1)\}} x^1(t-\tau)\cdot x^2(t)$$

$\tau \in [-(T^1-1),\dots,0,\dots,(T^1-1)]$. The maximum $\phi_{x^1x^2}(\tau)$ w.r.t. τ identifies the displacement of x^1 vs x^2 (portamento)

CC normalized: $$\bar{\phi}_{x^1x^2}(\tau) = \frac{\phi_{x^1x^2}}{\sqrt{\sum_{t=0}^{T^1-1}(x^1(t))^2 \sum_{t=0}^{T^2-1}(x^2(t))^2}} \in [-1,+1]$$

- $\bar{\phi}_{x^1x^2}(\tau) = +1 \Rightarrow$ the two TS have the same shape if aligned at time τ
- $\bar{\phi}_{x^1x^2}(\tau) = -1 \Rightarrow$ // // // but opposite sign if aligned at time τ
- $\bar{\phi}_{x^1x^2}(\tau) = 0$ there are no linear relationship between the two TS

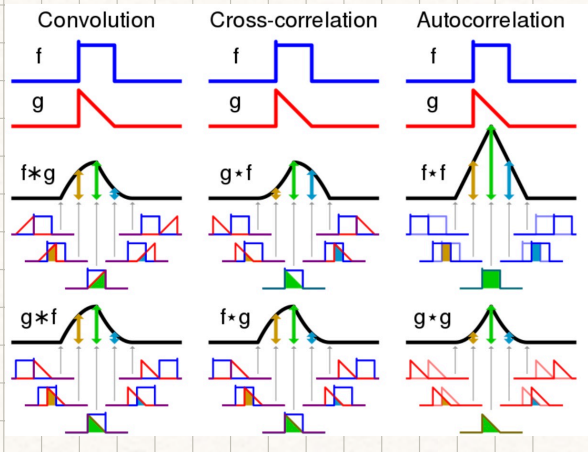
Convolution: $$(f * g)[n] = \sum_{t=-M}^{M} f(n-t)\, g(t)$$ $[-M,+M]$ support

x^1 τ x^2

Cons discrete →

Similar to CC but one of the signals is reversed

$$\frac{\partial (f*g)}{\partial x} = \frac{\partial f}{\partial x}\cdot g = \frac{\partial g}{\partial x}\cdot f$$



Autoregressive Process: a TS autoregressive process (AR) of order K is the linear system

$$x_t = \sum_{k=1}^{K} \alpha_k x_{t-k} + \varepsilon_t \qquad \alpha = \{\alpha_k\}_K$$

K → memoria, iperparametro

x_t regresses on itself. α_k ⇒ linear coefficients s.t. $|\alpha| < 1$

ε_t ⇒ sequence of i.i.d. values with $E[\varepsilon_t] = 0$ $Var[\varepsilon_t] = \sigma$!!!

ARMA (Autoregressive with Moving Average process)

$$x_t = \sum_{k=1}^{K} \alpha_k x_{t-k} + \sum_{q=1}^{Q} \beta_q \varepsilon_{t-q} + \varepsilon_t \longrightarrow \text{white noise}$$

reg. of TS — reg of the past errors (stochastically uncorrelated information)

denote new information at time t

Need to estimate: α_t and β_t, the order of K and Q.

I can use the set of autoregressive parameter $\alpha^i_1, \dots, \alpha^i_K$ fitted to a specific time series x^i to confront it to other TS

- TS clustering $d(x^1, x^2) = \| \alpha^1 - \alpha^2 \|^2_M$
- novelty / anomaly detection Test Error $(x_t, \hat{x}_t) \leq \xi$

  $\hat{x}_t$ AR predicted value

Spectral analysis: analyzing TS in the frequency domain

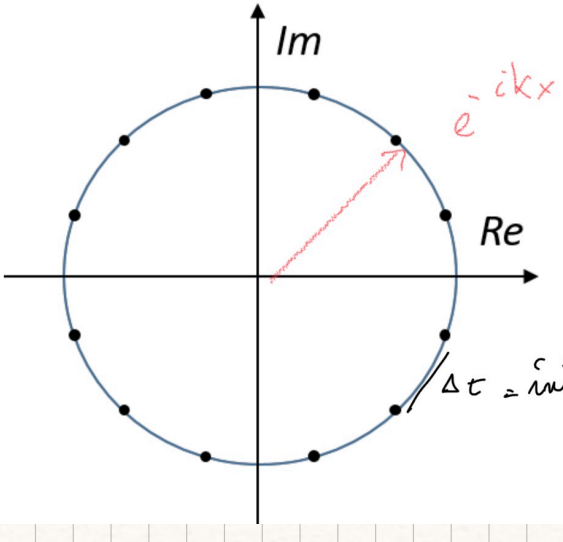
(linear combination of sinusoids and cosine)

Given an orthonormal system $\{e_1, e_2, \dots, e_n\}$ for E, we can represent any $f \in E$ by: $\sum_{1}^{\infty} \langle f, e_k \rangle e_k$

Fourier Series: $\frac{a_0}{2} + \sum_{k=1}^{\infty} [a_k \cos(kx) + b_k \sin(kx)]$

using $\cos(kx) - i \sin(kx) = e^{-ikx}$ $\quad i = \sqrt{-1}$

$$\sum_{k=-\infty}^{\infty} c_k e^{ikx}$$



Δt = intervallo di campionamento dei dati della time series

Given TS $x = x_0, x_1, \dots x_{N-1}$ $\quad$ length$(x) = N$ $\quad x_n \in \mathbb{R}$

using the exponential formulation, the orthonormal system is made of $\{e_0, e_1, \dots, e_{N-1}\}$, $e_k \in \mathbb{C}^N$. The n-th element of e_k is: $[e_k]_n = e^{\frac{-2\pi i n k}{2}}$

A basis e_k at frequency k has N elements sampled from the root of the unitary circle in imagery - real space.

Discreet Fourier Transform : given TS $x = x_0, x_1 \dots, x_{N-1}$

$$X_k = \sum_{n=1}^{N-1} x_n \, e^{\frac{-2\pi i n k}{N}} \qquad k = 1 \dots N$$

coefficient in the spectral domain

$$x_n = \frac{1}{N} \sum_{k=1}^{N-1} X_k \, e^{\frac{2\pi i n k}{N}}$$

data in the time domain. Back to the time domain

Amplitude : $A_k = |X_k| = \sqrt{Re^2(X_k) + Im^2(X_k)}$

Power : $P_k = \frac{|X_k|^2}{N}$

Spectral centroid :

$$\mu = \frac{\sum_{k=b_1}^{b_2} f_k \, s_k}{\sum_{k=b_1}^{b_2} s_k}$$

f_k → k-th frequency (Hz)

s_k → corresponding weight (A_k or P_k)

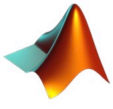
spectral-weighted average frequency (between b_1 and b_2 frequency bands)

- Spread - Standard deviation around the spectral centroid μ

$$\sigma = \sqrt{\frac{\sum_{k=b_1}^{b_2} (f_k - \mu)^2 s_k}{\sum_{k=b_1}^{b_2} s_k}}$$

- Kurtosis – (4^{th} order moment) Measures flatness or non-Gaussianity of the spectrum around the centroid μ

$$K = \frac{\sum_{k=b_1}^{b_2} (f_k - \mu)^4 s_k}{\sigma^4 \sum_{k=b_1}^{b_2} s_k}$$



k = pkurtosis(x)

Image description

Color histograms : count the number of pixel for a given colour, no information regarding shape and position. Invariant to image permutation.

Intensity vector w (grey scale) normalized to make invariant w.r.t. intensity change $d = \frac{w - \bar{w}}{\|w - \bar{w}\|}$

Scale Invariant Feature Transform (SIFT):

1 Center the image patch on a pixel (x, y) of image I

2 Represent image at scale σ, convolve it with Gaussian filter with std σ

like a magnifier

$$L_\sigma(x, y) = G(x, y, \sigma) * I(x, y)$$

$$G(x, y, \sigma) = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

σ high $\Rightarrow$ blur image

σ low $\rightarrow$ clean image

3 Compute the gradient of intensity (direction in changing luminority) in the patch

$$\frac{\partial L_\sigma}{\partial x} = G_x = [1 \; 0 \; -1] * L_\sigma(x, y)$$

$$\frac{\partial L_\sigma}{\partial y} = G_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} * L_\sigma(x,y)$$

intensity of the pixel

Magnitude $m_\sigma(x,y) = \sqrt{G_x^2 + G_y^2}$ Orientation $\theta_\sigma(x,y) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$

direction of the pixel

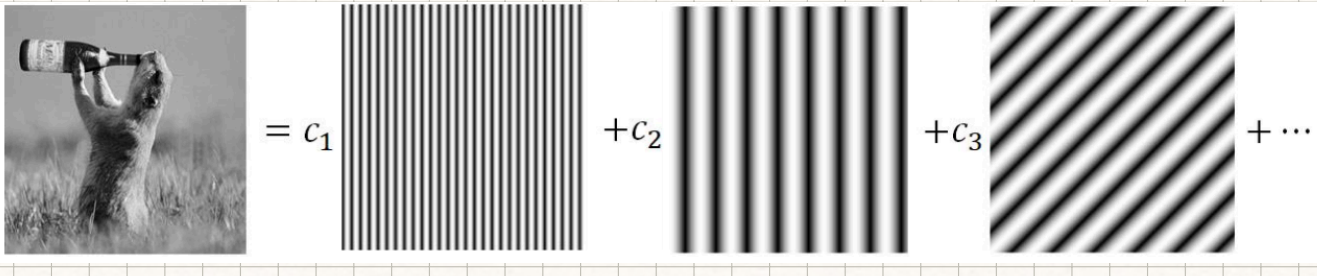
4 Create gradient histogram

- 16x16 neighbourhood is taken around the point, dividing it in 4 x 4 gradient window
  - Histogram of 4x4 samples per window on 8 different orientation
  - Gaussian weighting on center keypoint (?) (width $1,5\sigma$)
  - $4 \times 4 \times 8 = 128$ total descriptor size
- Normalize to unity for illuminance invariance
- Threshold gradient magnitude to 0,2 to avoid saturation
- Rotate all angles by main orientation (mean of the gradients) to obtain rotational invariance.
- Invariant to image scale and noise

Fourier coefficients of a 2D-DFT

$$H(K_x, K_y) = \sum_{x=1}^{N-1} \sum_{y=1}^{M-1} I(x,y)\, e^{-2\pi i \left(\frac{x K_x}{N} + \frac{y K_y}{M}\right)}$$

I can write my image as a sum of sine/cosine waves of varying frequency in x and y direction



Convolution theorem: $F(f * g) = F(f) \cdot F(g)$

$$I * g = F^{-1}(F(I) \cdot F(g))$$

F^{-1} inverse fourier transf.

Image, a function returning intensity value

a filter (gaussian)

Visual feature detector should has repeatability property: detect the same feature in different image portions and in different images.

- Image gradient (greylevel) $\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]$ direction of change of intensity

Edges are pixel region in which intensity gradient changes abruptly

$$G_x = \frac{\partial I}{\partial x} \approx I(x+1, y) - I(x-1, y)$$

$$G_y = \frac{\partial I}{\partial y} \approx I(x, y+1) - I(x, y-1)$$

The gradient of the image can be obtained convolving the image with some operator:

Sobel $\left\{ G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \right\}$

the dimensions of the matrix could be bigger $\Rightarrow$ more computational cost

Blob detection: blob are connected pixel regions with little gradient variability. We use Laplacian of Gaussian as a filter

$g_\sigma(x,y)$ has maximum response when centered on a circle of radius $\sqrt{2}\,\sigma$

$$LoG = \nabla^2 g_\sigma(x,y) = \frac{\partial^2 g_\sigma}{\partial x^2} + \frac{\partial^2 g_\sigma}{\partial y^2}$$

normalized multiplying by σ^2

1. Convolve image with a LoG filter at different scales, starting from a small σ, increasing each time a little bit. $\sigma = k\sigma_0$ by varying k. k power of 2
2. Find maxima of squared LoG response, in particular considers:
   - Find maxima on space-scale ⇒ takes the pixel with the highest response for a given scale.
   - Find maxima between scale. Takes the largest circle between circle including same pixel response.
   - threshold.

LoG filter can be approximated as a Difference of Gaussian

$$g_{k\sigma_0}(x,y) - g_{\sigma_0}(x,y) \approx (k-1)\,\sigma_0^2\,\nabla^2 g_{(k-1)\sigma_0}$$

Laplacian detector are invariant to scale thanks to maximization in space-scale. Not invariant to affine transformation

Maximally Stable Extremal Regions (MSER)

Blobs

1 Extract covariant regions (blobs) that are stable connected components of intensity sets of the image

2 The blobs are generated (locally) by binarizing the image over a large number of thresholds
- Invariance to affine transformation of image intensities
- Stability, on multiple threshold
- Multi-scale (connected components are identified by intensity stability not by scale)
- Sensitive to local lighting effects, shadows...

3 We can fit an ellipse enclosing the stable region

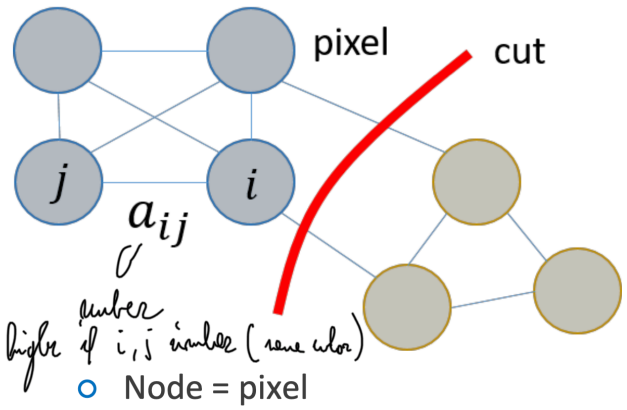
Generates frames from the image by thresholding it on all grayvels. Capture those regions that from a small seed of pixel grow to a stably connected region. Stability is assessed by looking at derivatives of region masks in time (most stable ⇒ minima of connected region variation)

Image Segmentation: the process of partitioning an image into a set of homogeneous pixel

Naive approaches: - Apply K means to pixels color and position.

Normalized Cuts (NCut): Breaking graph into pieces, obtaining a disconnected graph, by cutting edges of low affinity.

Each node is a pixel of the image.

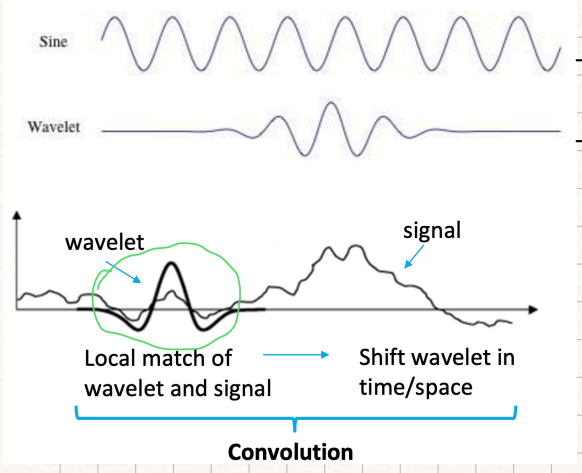


- Node = pixel
- a_{ij} = affinity between pixels (at a certain scale σ)

the normalized cut problem is NP-hard approximate solution as an eigenvalue problem, but it's intractable with big images. We can use superpixels: cluster the pixel with K-means and use the cluster as nodes

We can apply NCut to Image Segmentation problem => Superpixel trick -Group together similar pixel. the nodes of the graph represents different Superpixel. Now I can use Ncut, MRF, ...

Limitation of DFT: sometimes is good practice to slice the signal in time slots in time analysis and frequency slots in frequency analysis



→ basis function used in Fourier transform
→ basis function used in wavelet transform

1 scale and shift original signal
2 compare signal to a wavelet
3 compute a coefficient of similarity

Split the signal using an orthonormal basis generated by translation and dilatation of a mother wavelet

$$\sum_t x(t)\,\psi_{j,k}(t)$$

j, k regulate scaling and shifting of

the wavelet

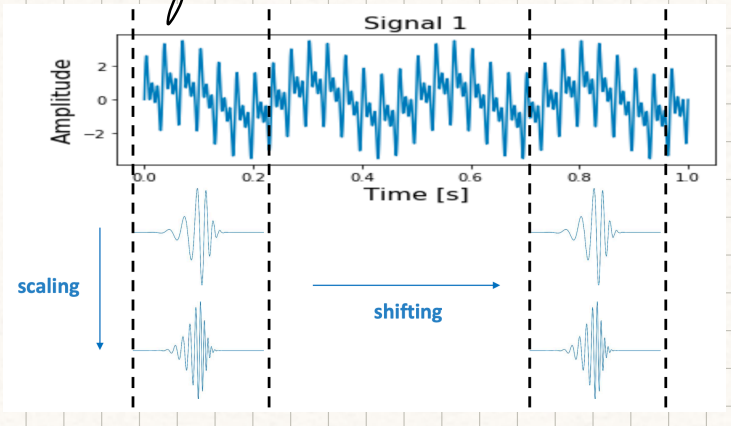
$$\psi_{j,k}(t) = 2^{\frac{k}{2}} \psi\left(\left(t - j2^k\right) / 2^k\right)$$

$k < 1$ compresses the signal ; $k > 1$ dilates the signal

mother wavelets

Scaling and dilation is akin to a sort of frequency: high scale mean stretched wavelet with slowly changing coarse feature and low frequency, while low scale compressed wavelet with rapidly changing details and high frequency

Shifting moves the wavelet in time/space on the signal



Compute coefficients of signal - wavelet match across all scales and shift

Discrete wavelet transform (DWT): uses a finite set of scales and shifts rather than "any possible value" as in the continuous wavelet transform

Generative learning: ML models that represent knowledge inferred from data under the form of probabilities.

- Probabilities can be sampled, therefore new data can be generated.
- Different type of tasks: supervised, unsupervised, weakly supervised.
- Possibility to incorporate prior knowledge on data and tasks.
- Understand how data are generated.

Random variables (RV): is a function describing the outcome of a random process by assigning unique values to all possible outcome of the experiment. A single RV is referring to a attribute of our data. $P(X = x)$ → possible outcome, ↳ RV

We have discrete / continuous RV

↳ $P(X = x) \in [0,1]$, $\sum_{x \in \mathcal{R}} P(X = x) = 1$

$p(t)$ describe the probability of a RV to take the value t

$$\int_{-\infty}^{+\infty} p(t)\,dt = 1$$

$$P(X \leq x) = \int_{-\infty}^{x} p(t)\,dt$$

A discrete random process can be described by a set of RVs $X_1, \dots, X_m$ then the joint probability is:

$$P(X_1 = x_1, \dots, X_m = x_m) = P(x_1 \wedge \dots \wedge x_m)$$

the joint conditional probability of $x_1, \dots, x_m$ given y:

$$P(x_1, \dots, x_m | y)$$

measures the effect of the realization of an event y on the occurence of $x_1, \dots, x_m$. For each value of y there is a distribution $P(x|y)$

Chain Rule: $P(x_1, \dots, x_i, \dots, x_n | y) = \prod_{i=1}^{n} P(x_i | x_1, \dots, x_{i-1}, y)$

Marginalization: marginalizzato rispetto a un sottoinsieme di elementi di X_2

complete theorem $P(X_1 = x_1) = \sum_{X_2} P(X_1 = x_1 / X_2 = x_2) P(X_2 = x_2)$

Bayes Rule: given hypotheris $h_i \in H$ and observation d

$$P(h_i | d) = \frac{P(d | h_i) \cdot P(h_i)}{P(d)} = \frac{P(d | h_i) \cdot P(h_i)}{\sum_j P(d | h_j) P(h_j)}$$

$P(h_i)$ = prior probability of h_i

$P(d | h_i)$ = is the conditional probability of observing d given that hypoteris h_i is true (likelihood)

$P(d)$ is the marginal probability of d

$P(h_i | d)$ is the porterir probability that hypotheris is true given the data and the previous belief about the hypotheris

Independence: X and Y are independent if knowledge about X does not change the uncertainty about Y and vice versa

$$X \perp Y = P(X, Y) = P(X | Y) P(Y) = P(Y | X) P(X) = P(X) P(Y)$$

Conditional Independence: X and Y are cond. ind. given Z if the realization of X and Y is an independente event of their conditional probability distribution given Z

$$X \perp Y | Z = P(X, Y | Z) = P(X | Y, Z) P(Y | Z) = P(Y | X, Z) P(X | Z) = P(X | Z) P(Y | Z)$$

three approaches to inference

$$\text{Bayesian} = P(X|d) = \sum_i \overbrace{P(X|h_i)}^{A} \overbrace{P(h_i|d)}^{B}$$ → weight all hypothesis by its posterior prob.

d = DATASET I.I.D.

MAP Infer X from $P(X|h_{MAP})$, where h_{MAP} is the (Maximum a Posteriori hypothesis given d. MAP maximize B

$$h_{MAP} = \arg\max_{h \in H} P(h|d) = \arg\max P(d|h)P(h)$$

MAP regularize w.r.t. MLE because it consider the model complexity ($P(h)$)

MLE, used usually to perform model selection. It maximize A.

Assuming uniform priors $P(h_i) = P(h_j)$. Infer X from $P(X|h_{MLE})$

$$h_{MLE} = \arg\max_{h \in H} P(d|h)$$

I prefer to use MAP in the case I have sparse data (few data for high dimensional input) or small data, otherwise I prefer MLE

MLE and MCP are point estimation of the Bayesian since they infer only on one possible hypothesis. MAP and Bayesian prediction become closer as more data gets available

$P(h)$ introduce regularization: complex hypotheses have lower prior probability. Hypothesis prior embodies trade-off between complexity and degree of fit.

MAP hypothesis $h_{MAP} = \max_h P(d|h)P(h) \equiv \min_h -\log_2 P(d|h) - \log_2 P(h)$

MAP implies choosing the hypothesis that provides maximum compression

number of bit required to specify h

Maximum likelihood Estimation: find the model θ that is most likely to have generated the data d

$$\theta_{MLE} = \arg\max_{\theta \in \Theta} P(d|\theta)$$

from a family of parametrized distribution $P(X|\theta)$.

Opt. prob. that consist in maximize $\mathcal{L}(\theta|x) = P(x|\theta)$, to be a function of θ.

$$\frac{\partial \mathcal{L}(\theta|x)}{\partial \theta} = 0$$

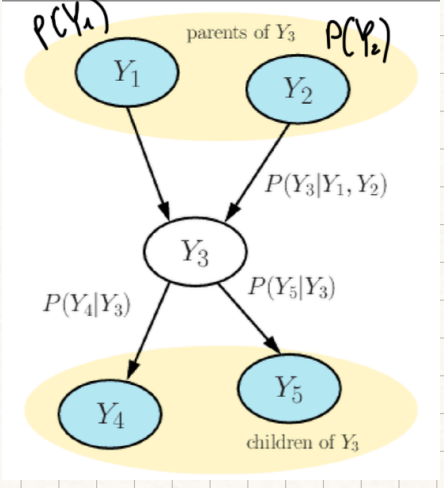
With both observed/hidden variables:

$$P(X|\theta) = \sum_z \mathcal{L}(\theta|X,z) = \sum_z P(X,z|\theta) = \sum_z P(z|X,\theta)\, P(X|\theta)$$

a z-step size iterative process

$$\theta^{(k+1)} = \arg\max_{\theta} \sum_z P(z=z|X,\theta^{(k)}) \log \mathcal{L}_c(\theta|X,z=z)$$

Bayesian Network (DAG)



DAG $G = (V, E)$

nodes $v \in V$ represent random variables

Shaded = observed; Empty = un-observed

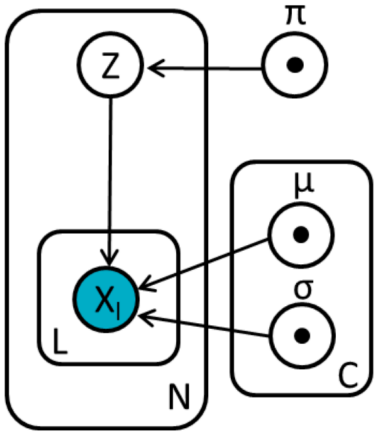
edges $e \in E$ describe the conditional independence

Conditional distribution table (CPT)

$$P(Y_1, \ldots, Y_n) = \prod_{i=1}^{N} P(Y_i | par(Y_i))$$

If the N nodes have a maximum of L children $\Rightarrow (K-1)^L \times N$ independent parameters

(K: distinct value of R.V.)



Gaussian Mixture Model

- Boxes denote replication for a number of times denoted by the letter in the corner
- Shaded nodes are observed variables
- Empty nodes denote un-observed latent variables
- Black seeds (optional) identify model parameters
  - $\pi \rightarrow$ multinomial prior distribution
  - $\mu \rightarrow$ means of the C Gaussians
  - $\sigma \rightarrow$ std of the C Gaussians

Local Markov Property: each RV is conditionally independent of all its non-descendants given a joint state of its parent

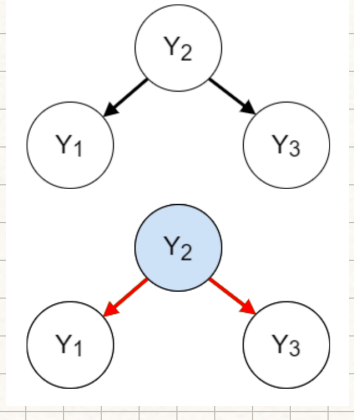
$$Y_v \perp Y_{V \setminus ch(v)} \mid Y_{pa(v)} \quad \text{for all } v \in V$$

Markov Blanket: $MB(A)$ of a node A is the minimal set of vertices that shield the node from the rest of the BN. The behaviour of a node can be determined and predicted from the knowledge of its Markov blanket. $P(A \mid MB(A), z) = P(A \mid MB(A)) \; \forall z \notin MB(A)$

$MB(A)$ contains: parent $pa(A)$, children $ch(A)$, children's parent $pa(ch(A))$

Tail-to-tail connections: $P(Y_1, Y_3 \mid Y_2) P(Y_2) =$

$= P(Y_1 \mid Y_2) P(Y_3 \mid Y_2) P(Y_2)$



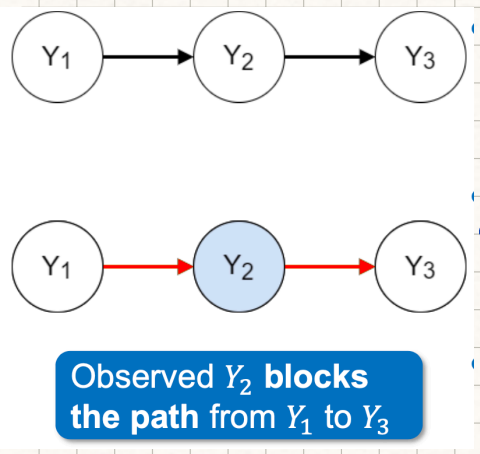
if Y_2 is unobserved, Y_1 and Y_3 are marginally dependent $\quad Y_1 \not\perp Y_3$

if Y_2 is observed, Y_1 and Y_3 are conditionally independent

$$Y_1 \perp Y_3 \mid Y_2$$

When Y_2 is observed is said to block the path from Y_1 to Y_3

Head to Tail:



$$P(Y_1, Y_2, Y_3) = P(Y_1) P(Y_2 \mid Y_1) P(Y_3 \mid Y_2) =$$

$$= P(Y_1 \mid Y_2) P(Y_3 \mid Y_2) P(Y_2)$$

use Bayes rule

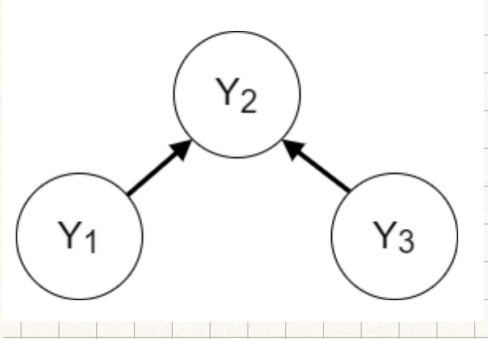
- if Y_2 is unobserved, Y_1 and Y_3 are marginally dependent

$$Y_1 \not\perp Y_3$$

- if Y_2 is observed, Y_1 and Y_3 are conditionally independent

$$Y_1 \perp Y_3 \mid Y_2$$

Head to Head:



$$P(Y_1, Y_2, Y_3) = P(Y_1) P(Y_3) P(Y_2 \mid Y_1, Y_3)$$

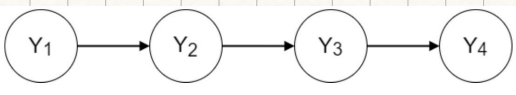
- if Y_2 is observed, Y_1 and Y_3 are conditionally dependent

$$Y_1 \not\perp Y_3 \mid Y_2$$

- if Y_2 is unobserved, Y_1 and Y_3 are marginally independent

$$Y_1 \perp Y_3$$

Derived conditional Independence Relationships



Local Markov Relationships

$Y_1 \perp Y_3 \mid Y_2$

$Y_4 \perp Y_2 \mid Y_3$

Derived Relationship

$Y_1 \perp Y_4 \mid Y_2$

D-separation: Let $r = Y_1 \leftrightarrow \dots \leftrightarrow Y_2$ be an undirected path between Y_1 and Y_2, then r is d-separated by Z (group of nodes) if there exist at least one node $Y_c \in Z$ for which path r is blocked.

D-separation holds if at least one of the following holds:

- r contains an head-to-tail structure $Y_i \to Y_c \to Y_j$ and $Y_c \in Z$
- " " " tail-to-tail " $Y_i \leftarrow Y_c \to Y_j$ and $Y_c \in Z$
- " " " head-to-head " $Y_i \to Y_c \leftarrow Y_j$ and neither Y_c nor its descendants are in Z

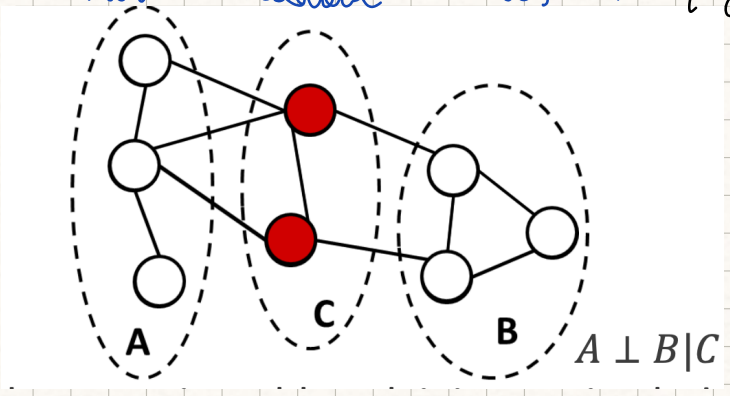
Nodes d-separation: Two nodes Y_i and Y_j in a BN G are said to be d-separated by $Z \subset V$ (denoted by $\text{Dsep}_G(Y_i, Y_j | Z)$) if and only if all undirected paths between Y_i and Y_j are d-separated by Z.

If Y_i and Y_j are d-separated by Z they are conditional independent

Markov Blanket: The Markov blanket $MB(Y)$ is the minimal set of nodes which d-separates a node Y from all other nodes (Y is conditionally independent of all other nodes in the BN)

$$MB(Y) = \{pa(Y), ch(Y), pa(ch(Y))\}$$

Markov Random Fields



$A \perp B | C$

$A, B \subset V$ are conditionally independent given $C \subset V \setminus \{A, B\}$ if all paths between nodes in A and B pass through at least one of the nodes in C

- Markov Blanket of a node include all and only its neighbors

Nodes that are not neighbors are conditionally independent given the remainder of the nodes

$$P(X_j, X_i | X_{V \setminus \{j,i\}}) = P(X_j | X_{V \setminus \{j,i\}}) \cdot P(X_i | X_{V \setminus \{j,i\}})$$

Clique: A subset of nodes C in graph G such that G contains an edge between all pair of nodes in C

Max clique: a clique that cannot include any further node from G without ceasing to be a clique.

Boltzmann distribution: A convinient and widely used strictly positive representation of the potential function is

$$\Psi(X_c) = \exp\{-\beta(X_c)\}$$

$\beta(X_c)$: energy function on c max clique

Structural learning problem: find the structure of the graph, defining edges between nodes and therfore causal relationship between nodes (assign direction to links)

Starting from a graph G_0 we can move in the search space of G_K. Then we return the highest scoring graph G^*.

Scoring function: AIC (minimize), BIC (maximize)

information theoretic = fit likelihood + model complex.

$$\log P(D|G) \approx \sum_i \sum_x \log \hat{P}(x|Pa(x)) + \log P(G)$$

Bayesian = likelihood + prior choice

Search strategy: find maximal scoring structures is NP complete.

Constrain search strategy: starting from a candidate structure modify iteratively by local operations (edge/node ins. del.?. Each operation has a cost → cost opt. problem (hill-climb, sim. annealing.)

Constrain search space: Known node order → start with Markov blanket. Search in the space of structure equivalence classes Search in the space of node orderings

Constraint-based Models: Tests of conditional independence $I(X_i, X_j|Z)$ determine edge presence (network skeleton).

statistical hypothesis testing: $G^2(X_i, X_j|\mathbf{Z}) = 2\sum_{x_i,x_j,\mathbf{z}} n_D(x_i, x_j, \mathbf{z}) \frac{n_D(x_i,x_j,\mathbf{z}) n_D(\mathbf{z})}{n_D(x_i,\mathbf{z}) n_D(x_j,\mathbf{z})}$

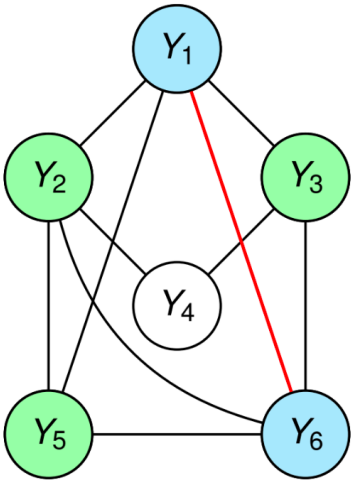
Use deterministic rules based on local Markov dependencies to determine

edge orientation (DAG)

Testing strategy: Level-wise → test $I(X_i, X_j | Z)$ are performed in order of increasing size of the conditioning set Z (starting empty)

↓

PC algorithm



undirected

Initialize a fully connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

for each edge $(Y_i, Y_j) \in \mathcal{V}$

- **if** $I(Y_i, Y_j)$ **then** prune (Y_i, Y_j) delete if independent

$K \leftarrow 1$

for each test of order $K = |Z|$

- **for each** edge $(Y_i, Y_j) \in \mathcal{V}$
  - $Z \leftarrow$ set of conditioning sets of K-th order for Y_i, Y_j
  - **if** $I(Y_i, Y_j | z)$ **for any** $z \in Z$ **then** prune (Y_i, Y_j) delete if cond. independent
- $K \leftarrow K + 1$

return $\mathcal{G}$

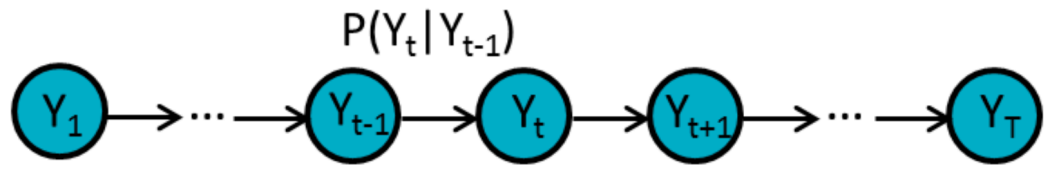
Node-wise: tests are performed on a single edge at the time, exhausting independence checks on all conditioning variables. ↑ IAPC alg

Nodes that enter Z are chosen in the neighborhood of X_i and X_j

Hybrid models: model selection of constrained structures. Multi-stage algorithm combining previous approaches: independence tests to find a good sub-optimal skeleton as starting point, then search and score refining skeleton. Max-Min Hill Climbing (MMHC): optimized constraint-based approach to reconstruct the skeleton using the candidate parents in the skeleton to run a search and score appro

Hidden Markov Models

Sequences :



a collection of observations Y_t, where t represent the position of the element according to a order (e.g. time)

- $Y_1, \dots, Y_N$ i.i.d., generally different sequences have different lenghts

Markov Chain: First order MC are directed graphical model for sequences s.t. element X_t only depends on its predecesor in the sequence.

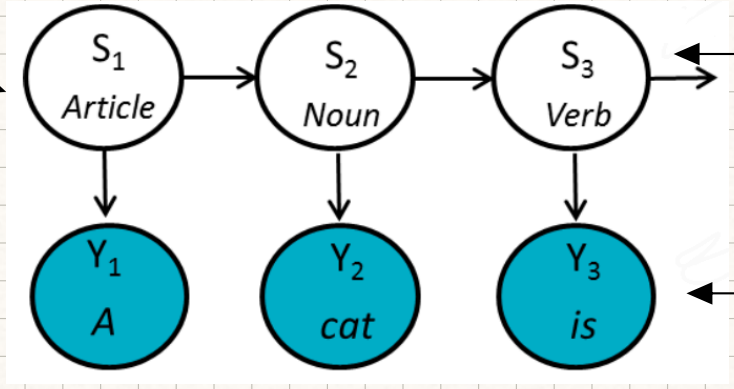
- Joint probability factorizes as

$$P(X) = P(X_1, \dots, X_T) = P(X_1) \prod_{t=2}^{T} P(X_t | X_{t-1})$$

$P(X_1)$: prior distribution

$P(X_t | X_{t-1})$: transition distribution

Hidden Markov Model (HMM): Stochastic proces where transition dynamics is disentangled from observations generated by the proces



mean observable variable

$P(S_t | S_{t-1})$ transition distribution

$P(Y_t | S_t)$ emission probability

State transition is an observed proces characterized by the hidden state variable S_t, which are often discrete with value in $\{1 \dots C\}$. Multinomial **state transition** and **prior probability**

a table $\leftarrow A_{ij} = P(S_t = i \mid S_{t-1} = j)$ $\quad \pi_i = P(S_1 = i)$

Observation are generated by the **emission distribution**

a table if y is discrete, possibly a gaussian if y is continuous $\leftarrow b_i(y_t) = P(Y_t = y_t \mid S_t = i)$

HMM Joint Probability Factorization: parametrized by $\theta = (\pi, A, B)$ and the finite number of hidden states C

$$P(Y_1 \dots Y_T) = P(Y = y) = \sum_S P(Y = y, S = s) =$$

$$= \sum_{s_1 \dots s_T} \left\{ P(S_1 = s_1) P(Y_1 = y_1 \mid S_1 = s_1) \prod_{t=2}^{T} P(S_t = s_t \mid S_{t-1} = s_{t-1}) \cdot \right.$$

$$\left. \cdot P(Y_t = y_t \mid S_t = s_t) \right\}$$ recall discrete time in Markov chain

Smoothing: given a model θ and an observed sequence y, determine the distribution of the hidden state at time t $P(S_t \mid Y = y, \theta)$

Learning: given a dataset of N observed sequences $D = \{y^1, \dots, y^N\}$ and the number of hidden states C, find the parameters

π, A, B that maximize the probability of model $\theta = \{\pi, A, B\}$ having generated the sequences in D

Optimal State Assignment: given a model θ and an observed sequence y, find an optimal state assignment $s = s_1^*, \ldots, s_T^*$ for the MMC

Forward-Backward Algorithm for smoothing

$$P(S_t = i \mid y) \underset{\text{proportional } / P(y)}{\simeq} P(S_t = i, y) = P(S_t = i, Y_{1:t}, Y_{t+1:T}) =$$

$$P(Y_{t+1:T} \mid S_t = i) P(S_t = i, Y_{1:t}) = \beta_t(i)\, \alpha_t(i)$$

$\beta_t(i)$: future mess. $\alpha_t(i)$: past mess.

α computed as part of forward recursion

$$\alpha_1(i) = \overbrace{P(y_1 \mid S_1 = j)}^{b_i(y_1)} \overbrace{P(S_1 = j)}^{\pi_i}$$

$$\alpha_t(i) = P(S_t = i, Y_{1:t}) =$$

$$= P(y_1 \mid S_1 = j) \sum_{j=1}^{C} \overbrace{P(S_t = i \mid S_{t-1} = j)}^{A_{ij}} \alpha_{t-1}(j)$$

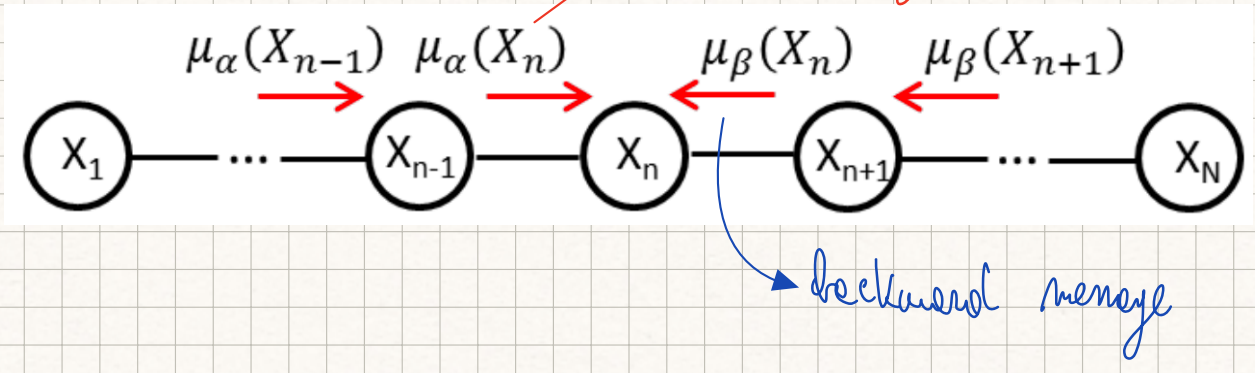
β-term computed as part of backward recursion

$$\beta_T(j) = 1$$

$$\beta_t(j) = P(y_{t+1:T} \mid s_t = j) =$$

$$= \sum_{i=1}^{C} \underbrace{P(s_{t+1} \mid s_t)}_{A_{ij}} \underbrace{P(y_{t+1} \mid s_{t+1})}_{b_i(y_{t+1})} \cdot \beta_{t+1}(i)$$

Sum-Product Message Passing



$$\underbrace{\mu_\alpha(X_n)}_{\alpha_t(i)} = \sum_{\underbrace{X_{n-1}}_{\sum_{j=1}^{C}}} \underbrace{\psi(X_{n-1}, X_n)}_{b_i(y_t) A_{ij}} \underbrace{\mu_\alpha(X_{n-1})}_{\alpha_{t-1}(j)}$$

$$\underbrace{\mu_\beta(X_n)}_{\beta_t(j)} = \sum_{\underbrace{X_{n+1}}_{\sum_{i=1}^{C}}} \underbrace{\psi(X_n, X_{n+1})}_{b_i(y_{t+1}) A_{ij}} \underbrace{\mu_\beta(X_{n+1})}_{\beta_{t+1}(i)}$$

prior

emission

Learning in HMM: learning parameter $\theta = (\pi, A, B)$ by maximum likelihood

π: transition

i.i.d. dataset

$$\mathcal{L}(\theta) = \log \prod_{n=1}^{N} P(Y^n|\theta) = \log \prod_{n=1}^{N} \left\{ \sum_{S_1^n \dots S_{T_n}^n} P(S_1^n) P(Y_1^n | S_1^n) \prod_{t=2}^{T_n} P(S_t^n | S_{t-1}^n) P(Y_t^n | S_t^n) \right\}$$

Expectation maximization algorithm: add **indicator variables** z

$$z_{ti}^n = \begin{cases} 1 & \text{if } n\text{-th chain is in state } i \text{ at time } t \\ 0 & \text{otherwise} \end{cases} = P(S_t = i | y^n)$$

1 only for 1 combination, 0 for the others

$$\mathcal{L}_c(\theta) = \log P(X, z|\theta) = \log \prod_{n=1}^{N} \left\{ \prod_{i=1}^{C} [P(S_1 = i) P(Y_1^n | S_1 = i)]^{z_{1i}^n} \right.$$

$$\left. \prod_{t=2}^{T_n} \prod_{i,j=1}^{C} P(S_t = i | S_{t-1} = j)^{z_{ti}^n z_{(t-1)j}^n} P(Y_t^n | S_t = i)^{z_{ti}^n} \right\} =$$

$$= \sum_{n=1}^{N} \left\{ \sum_{i=1}^{C} z_{1i}^n \log \pi_i + \sum_{t=2}^{T_n} \sum_{i,j=1}^{C} z_{ti}^n z_{(t-1)j}^n \log A_{ij} + \sum_{t=1}^{T_n} \sum_{i=1}^{C} z_{ti}^n \log b_i(y_t^n) \right\}$$

Expectation - maximization algorithm: maximize complete likelihood $\mathcal{L}_c(\theta)$

E-Step: given the current estimate of the model parameters $\theta^{(t)}$, compute $Q^{(t+1)}(\theta | \theta^{(t)}) = E_{z|X,\theta^{(t)}} [\log P(X, z|\theta)]$

variable ← | → current

M-Step: Find the new estimate of the model parameters

$$\theta^{(t+1)} = \arg\max_{\theta} Q^{(t+1)}(\theta|\theta^{(t)})$$

Iterate E-steps until $|\mathcal{L}_c(\theta)^{k+1} - \mathcal{L}_c(\theta)^{k} - 1| < \varepsilon$

Details of E-Step: Compute the expectation of the complete log-likelihood w.r.t. indicator variable z_{ti} assuming $\theta^t = (\pi^t, A^t, B^t)$ fixed at time t

$$Q^{(t+1)}(\theta|\theta^{(t)}) = E_{Z|X,\theta^{(t)}}[\log P(X, Z|\theta)]$$

given $E_z[z] = \sum_z z \cdot P(Z=z)$, to compute the conditional expectation $Q^{(t+1)}(\theta|\theta^{(t)})$ for the complete HMM log-likelihood we need to estimate

$$E_{Z|Y,\theta^{(k)}}[z_{ti}] = P(S_t = i|Y)$$

$$E_{Z|Y,\theta^{(k)}}[z_{ti} z_{(t-1)j}] = P(S_t = i, S_{t-1} = j|Y)$$

I can use forward-backward alg.

$$P(S_t = i|Y) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{c} \alpha_t(j)\beta_t(j)}$$

$$P(S_t = i, S_{t-1} = j|Y) = \frac{\alpha_{t-1}(j) A_{ij} b_i(y_t) \beta_t(i)}{\sum_{m,l=1} \alpha_{t-1}(m) A_{lm} b_l(y_t) \beta_t(l)}$$

Details of M-step: solve the optimization problem using the information computed at the E-step.

$$\theta^{(t+1)} = \arg\max_{\theta} Q^{(t+1)}(\theta|\theta^{(t)})$$

$$\frac{\partial Q^{(t+1)}(\theta|\theta^{(t)})}{\partial \theta}, \qquad \theta = (\pi, A, B)$$

transition distribution: $$A_{ij} = \frac{\sum_{n=1}^{N} \sum_{t=2}^{T^n} P(S_t = i, S_{t-1} = j | \tilde{Y})}{\sum_{n=1}^{N} \sum_{t=2}^{T^n} P(S_{t-1} = j | \tilde{Y})}$$

prior distribution: $$\pi_i = \frac{\sum_{n=1}^{N} P(S_1 = i | \tilde{Y})}{N}$$

emission distribution: $$B_{ki} = \frac{\sum_{n=1}^{N} \sum_{t=1}^{T_n} P(S_t = i | \tilde{Y}) \mathcal{I}(Y_t = k)}{\sum_{n=1}^{N} \sum_{t=1}^{T_n} P(S_t = i | Y^n)}$$

($\mathcal{I}$: indicator function for emission symbol)

Decoding problem: find the optimal state assignment $s = s_1^*, \ldots, s_T^*$ for Y sequence and trained HMM. Two options

1) Identify single s_t that maximize posterior $s_t^* = \arg\max_{i=1\ldots C} P(S_t = i | Y)$
2) Find most likely joint hidden state assignment $s^* = \arg\max_{s} P(Y, S = s)$

Viterbi algorithm: used for solving the optimal hidden state assignment $s = s_1^*, \dots, s_T^*$ given a sequence y and a trained HMM

- find the most likely joint distribution of hidden state assignment

$$s^* = \arg\max_s P(Y, S = s)$$

An example of max-product message passing algorithm:

$$\max_{s_1 \dots s_T} \prod_{t=1}^{T} P(y_t | s_t) P(s_t | s_{t-1}) =$$

$$= \max_{s_1 \dots s_{T-1}} \left(\prod_{t=1}^{T-1} P(y_t | s_t) P(s_t | s_{t-1}) \cdot \max_{s_T} P(Y_T | s_T) \cdot P(s_T | s_{T-1}) \right)$$

$\varepsilon_T(s_{T-1})$

the message $\varepsilon_T(s_{T-1})$ carries information from the end of the chain to the penultimate timestep. We can continue in this manner defining the recursion:

$$\varepsilon_{t-1}(s_{t-1}) = \max_{s_t} P(Y_t | s_t = s_t) P(s_t = s_t | s_{t-1} = s_{t-1}) \varepsilon_t(s_t)$$

$$\varepsilon_T(s_T) = 1 \qquad 2 \le t \le T$$

Root optimal state: $s_1^* = \arg\max_s P(Y_t | s_1 = s) P(s_1 = 1) \varepsilon_1(s_1)$

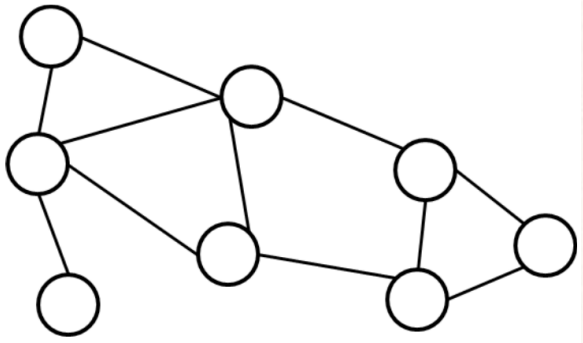
Recursive forward optimal state:

$$s_t^* = \arg\max_s P(Y_t | s_t = s) P(s_t = s | s_{t-1} = s_{t-1}^*) \varepsilon_t(s_t)$$

MARKOV RANDOM FIELD

Markov Random Fields (MFRs)

- Undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ (a.k.a. Markov Networks) UG
- Nodes $v \in \mathcal{V}$ represent random variables X_v
  - Shaded ⇒ observed
  - Empty ⇒ un-observed
- Edges $e \in \mathcal{E}$ describe bi-directional dependencies between variables (constraints)



Graph often coherent with data structure

Likelihood Factorization: $X = X_1, \dots, X_n$ RV associated to N nodes in the undirected graph

$$P(X) = \frac{1}{Z} \prod_C \psi_C(X_C)$$

- X_C → RV of the node in the maximal clique C
- $\psi_C(X_C)$ → potential function for clique C, is not a probability
- Z → partition function ensuring normalization $Z = \sum_{X \text{ over all states}} \prod_C \psi_C(X_C)$

expresses which configuration of the local variables are preferred

If the potential functions are positive, the Hammersley-Clifford theorem guarantees that the distribution can be represented by clique factorization.

Boltzmann distribution: strictly positive representation of the potential function

$$\psi_c(x_c) = \exp\{-E(x_c)\}$$

↳ energy function

factorized as: $\psi_c(x_c) = \exp\left(\sum_{k=1}^{K} \theta_{ck} f_{ck}(x_c)\right)$

K → number of feature functions

product of exponential

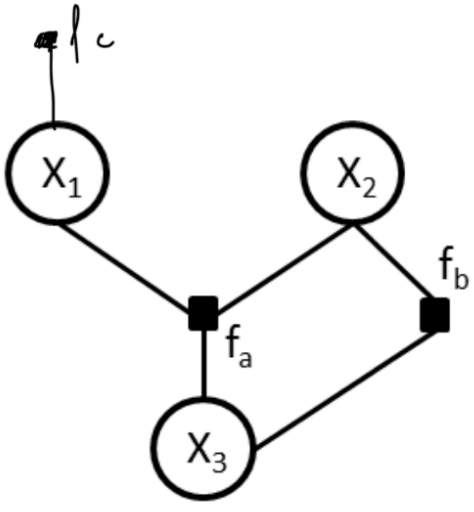
x_c → clique of RV

θ_{ck} → parameter $\in \mathbb{R}$

f_{ck} → feature function to compute the potential of clique c

We need factor graphs to represent factorization of potentials into feature functions

$f_c(x_1) \simeq P(x_1)$



- RV are circular nodes
- factors f_{ck} are denoted by square nodes
- edges connect a factor to RV

$$\psi(x_1, x_2, x_3) = f_a(x_1, x_2, x_3) f_b(x_2, x_3)$$

In ML we can assume that input data are observable RV.

- x_k - observable input in the factor k
- y_k - hidden (or partially observable) RV

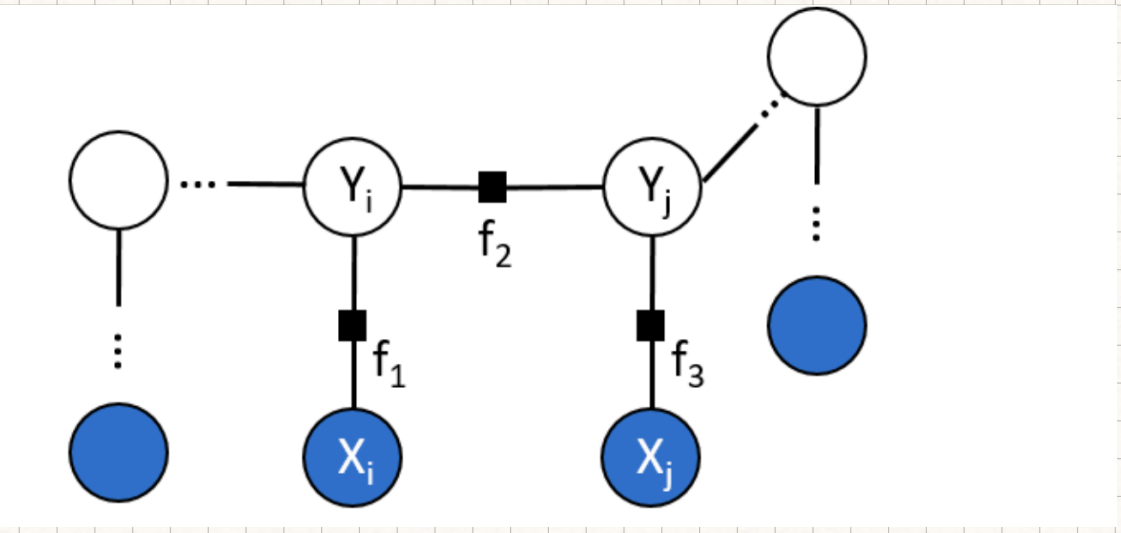
- $f_k(x_k, y_k)$ - factor feature function

Conditional distribution: $P(Y|X) = \frac{1}{Z(x)} \prod_k \exp\{\theta_k f_k(x_k, y_k)\}$

↓ input always available

$$Z(x) = \sum_y \prod_k \exp\{\theta_k f_k(x_k, y_k = y_k)\}$$

CONDITIONAL RANDOM FIELDS



$$P(Y|X,\theta) = \frac{1}{Z(x)} \exp(\theta_1 f_1(X_i, Y_i) + \theta_2 f_2(Y_i, Y_j) + \theta_3 f_3(X_i, Y_j) + \dots)$$

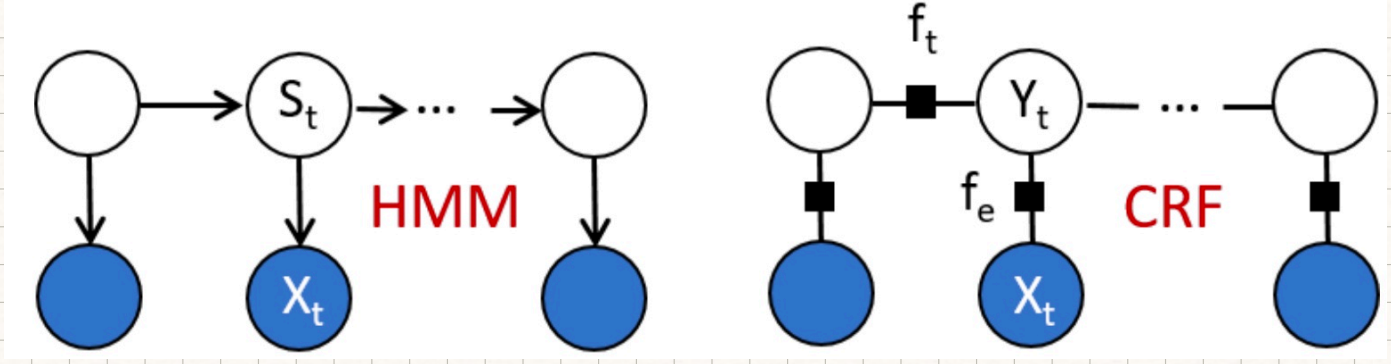
Feature function represents couplings or constraints between random variables, such as linear functions

If we know Y^m and X^m for some samples, we can fit θ in $P(Y|X,\theta)$

↓

We can use it to perform a prediction on a new X' without observable Y'. The model corresponds to the logistic regression / classifier

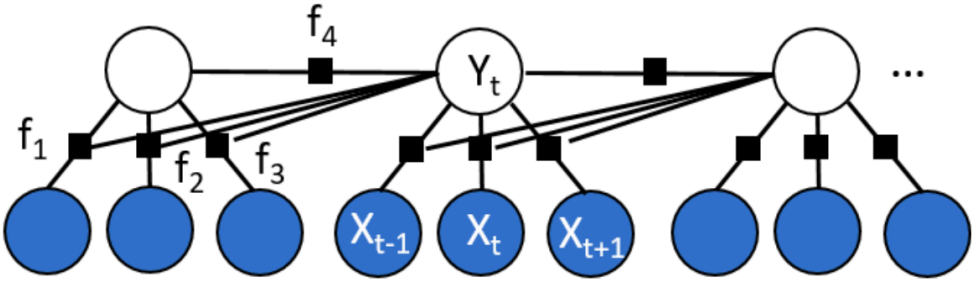
The undirected and discriminative equivalent of a HMM



$f_t(Y_{t-1}, Y_t)$ similar to $P(S_t|S_{t-1})$, $f_e(X_t, Y_t)$ similar to $P(X_t|Y_t)$

But I can place as many feature functions f_t I want between the same variables, while I can't place more transition prob. in HMM

Modeling explicitly input influence on transition



Linear Conditional Random Fields likelihood:

$$P(Y|X,\theta) = \frac{1}{Z(x)} \prod_t \prod_k \exp\left(\theta_k f_k(Y_t, Y_{t-1}, X_t)\right)$$

k parameters

We can use indicator variables in fk definition to include or disregard the influence of specific RV; $I_{Y_t=i}$, $I_{X_i=0}$

Smoothing problem in LCRF: $P(Y_t, Y_{t-1}|X)$, solved with forward-backward

$$P(Y_t, Y_{t-1}|X) \propto \underbrace{\alpha_{t-1}(Y_{t-1})}_{\text{forward msg}} \underbrace{\psi_t(Y_t, Y_{t-1}, X_t)}_{\text{clique weighting}} \underbrace{\beta_t(Y_t)}_{\text{backward msg.}}$$

clique weighting: $\psi_t(Y_t, Y_{t-1}, X_t) = \exp\left\{\theta_e f_e(X_t, Y_t) + \theta_t f_t(Y_{t-1}, Y_t)\right\}$

forward message: $\alpha_t(i) = \sum_j \psi_t(i, j, X_t)\, \alpha_{t-1}(j)$

backward message: $\beta_t(j) = \sum_i \psi_{t+1}(i, j, X_{t+1})\, \beta_{t+1}(i)$

Max product inference can be computed as in the Viterbi alg. (optimal hidden state assignment). the computationally expensive part is the computation of exponential summation in $Z(X)$ term.

Learning LCRF: maximum (conditional) log-likelihood

$$\max_\theta \mathcal{L}(\theta) = \max_\theta \sum_{n=1}^{N} \log\Big(P(y^n \mid x^n, \theta)\, P(\theta)\Big) =$$

$$= \max_\theta \log \prod_n \frac{1}{Z(x^n)} \prod_t \prod_k \exp\left(\theta_k f_k(y_t^n, y_{t-1}^n, x_t^n)\right) =$$

$$= \sum_n \sum_t \sum_k \theta_k f_k(y_t^n, y_{t-1}^n, x_t^n) - \sum_n \log Z(x^n) =$$

(n: dataset, t: time, k: feature function)

$$\log P(\theta) = \log \exp\left(-\frac{(\theta-\sigma)^T(\theta-\sigma)}{2\sigma^2}\right)$$

$$= \sum_n \sum_t \sum_k \theta_k f_k(y_t^n, y_{t-1}^n, x_t^n) - \sum_n \log Z(x^n) - \sum_k \frac{\theta_k^2}{2\sigma^2}$$

Penalization term

maximized by $\frac{\partial \mathcal{L}(\theta)}{\partial \theta_k} = 0$ iff: (A) = (B)

$$= \sum_{n,t} f_k(y_t^n, y_{t-1}^n, x_t^n) - \sum_{n,t} \sum_{y,y'} f_k(y, y', x_t^n)\, P(y, y' \mid x^n) - \frac{\theta_k}{\sigma^2}$$

(A) (B) (C)

(A): is $E[f_k]$ under the empirical distribution ↓ doesn't count

(B): is $E[f_k]$ under model distribution

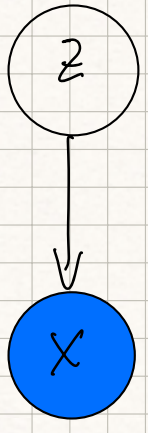
We learn θ parameters by SGD: $\theta^m = \theta^{m-1} - v_m \nabla L_m(\theta^{m-1})$

where $\nabla L_{mk}(\theta) =$

$$= \sum_t f_k(Y_t, Y_{t-1}, X_t^m) - \sum_t \sum_{y,y'} f_k(y, y', X_t^m) P(y, y' | X^m) - \frac{\theta_k}{N\sigma^2}$$

estimated by sum-product inference

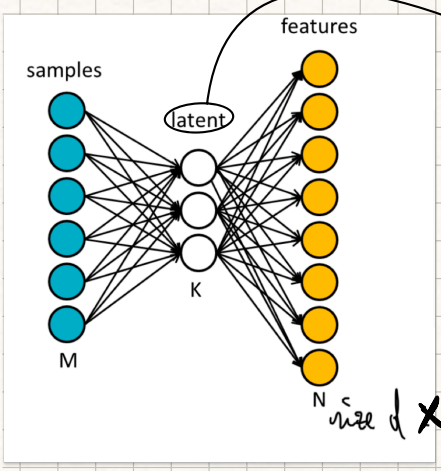
Latent Variable Models:



Latent variables: unobserved RV that define a hidden generative process of observed data. Practically a RV that define others RV.

Latent variable models likelihood:

$$P(X) = \int_z \prod_{i=1}^{N} P(x_i|z) P(z) \, dz$$



Latent space in which high-dimensional data can be represented.

Latent variables conditional and marginal distribution are more tractable than the joint distribution $P(X)$ $K << N$

Kullback-Leibler (KL) Divergence: an information metric on how close two distribution $p(z)$ and $q(z)$ are

$$KL(q \| p) = E_q\left[\log \frac{q(z)}{p(z|x)}\right] = \langle \log q(z) \rangle_q - \langle \log p(z|x) \rangle_q$$

KL is ≥ 0 because is composed by logarithm of probabilities. It is equal to 0 iff the two distributions are the same.

- q high and p high $\Rightarrow$ ☺
- q high and p low $\Rightarrow$ ☹
- q low $\Rightarrow$ don't care because Expectation is calculated on q

Jensen inequality: $\log f(\mathbb{E}[x]) \geq \mathbb{E}[\log f(x)]$

log-likelihood for a model with single hidden variable z and θ parameters:

$$\log p(x|\theta) = \log \int_z p(x,z|\theta)\, dz = \log \int_z \frac{q(z|\phi)}{q(z|\phi)}\, p(x,z|\theta)\, dz$$

$\hookrightarrow \neq 0$

we can write: $\log p(x|\theta) = \log E_q\left[\frac{p(x,z)}{q(z)}\right] \geq E_q\left[\log \frac{p(x,z)}{q(z)}\right] =$ (Jensen)

$$= E_q[\log p(x,z)] - E_q[\log q(z)] = \mathcal{L}(x,\theta,\phi)$$

pushing to overfitting the model ← Expectation of joint distribution — entropy → pushing to regularization

let's see how good is $\mathcal{L}(x, \theta, \phi)$ as a lower bound of $\log P(X|\theta)$

$$\log P(X|\theta) - \int_z Q(z) \log \frac{P(X,z)}{Q(z)} dz$$

introducing $Q(z)$ by marginalization $\left(\int_z Q(z) = 1\right)$

$$\int_z Q(z) \log P(x)\, dz - \int_z Q(z) \log \frac{P(x,z)}{Q(z)} dz =$$

$$= \int_z Q(z) \log \frac{P(x)Q(z)}{P(x,z)} dz = E_Q\left[\log \frac{Q(z)}{P(z|x)}\right] =$$

uso proprietà del logaritmo

$$= KL(Q(z|\phi) \,||\, P(z|x,\theta))$$

- We can assume the existence of a probability $Q(z|\phi)$ which allows to bound the likelihood $P(X|\theta)$ from below using $\mathcal{L}(x, \theta, \phi)$ → called evidence lower bound (ELBO)
- Optimal bound is $KL(Q(z|\phi) \,||\, P(z|x,\theta)) = 0$, that is $Q(z|\phi) = P(z|x,\theta)$
- Minimizing KL is equivalent to maximize the ELBO ⇒ change a sampling problem with an optimization problem

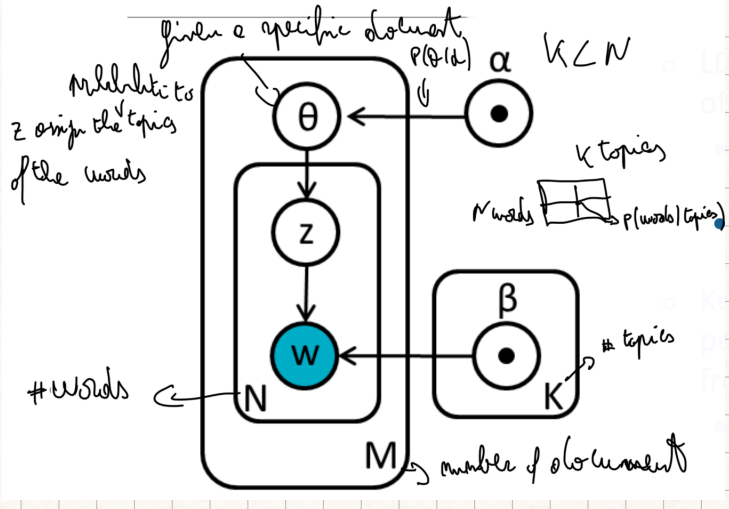
Expectation maximization learning: maximum likelihood learning with hidden variables can be approached by maximization of the ELBO:

$$\max_{\theta,\phi} \sum_{n=1}^{N} \mathcal{L}(x_n, \theta, \phi)$$

where θ are the model parameters and ϕ serve in $Q(z|\phi)$

- If $P(z|x,\theta)$ is tractable $\Rightarrow$ use it as $Q(z|\phi)$ (optimal ELBO)
- Otherwise, choose $Q(z|\phi)$ as a tractable family of distributions:
- find ϕ that minimize $KL(Q(z|\phi) \| P(z|x,\theta))$ & maximize $\mathcal{L}(x,\theta,\phi)$

Latent Dirichlet Allocation (LDA): models a document as a mixture of topics z.



- Assign one topic z to each item w with probability $P(w|z,\beta)$
- Pick one topic for the whole doc. with probability $P(z|\theta)$

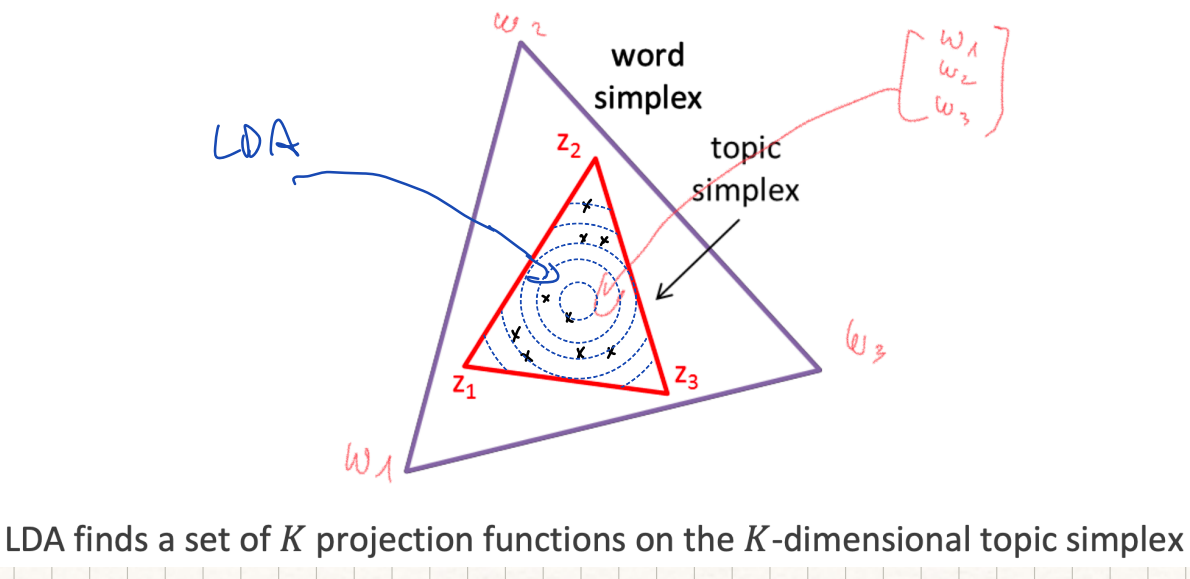
Each document has its personal topic proportion θ sampled from a distribution. θ defines a multinomial distribution but it is a random variable as well.

- $P(w|z,\beta)$ multinomial item-topic distribution
- $P(z|\theta)$ multinomial topic distribution with document-specific parameter θ

- $P(\theta|\alpha)$ Dirichlet distribution with hyperparameter α. Returns K vectors that sums to 1

Dirichlet distribution: $P(\theta|\alpha) = \frac{\Gamma(\sum_{k=1}^{K} \alpha_k)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1}$ for simplicity, $\alpha_k = \alpha \; \forall k$

α_k is a prior count of the k-th topic. It controls the mean shape and sparsity of multinomial parameters θ

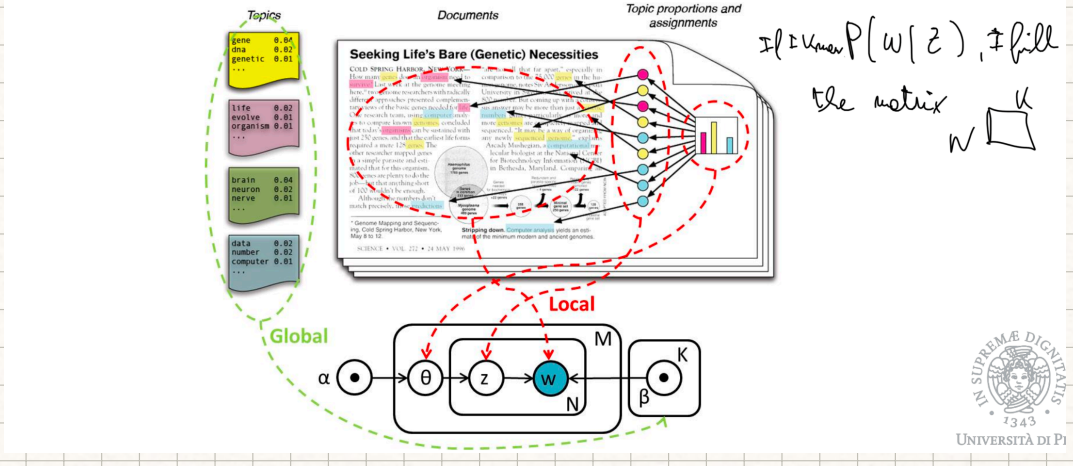


LDA finds a set of K projection functions on the K-dimensional topic simplex

α controls the degree of sparsity of the plate within the triangle

High α => uniform distribution. Small α => impulse, in our case $\alpha = 1$

LDA and Text Analysis



If I know $P(w|z)$, I fill the matrix $N \times K$

LDA generative process: for each of the M documents:

1 Choose $\theta \sim \text{Dirichlet}(\alpha)$

2 For each of the N items:
- choose a topic $z \sim \text{multinomial}(\theta)$
- pick an item w_j with multinomial probability $P(w_j | z, \beta)$

multinomial topic-item parameter matrix $[\beta]_{K \times V}$

$$\beta_{kj} = P(w_j = 1 | z_k = 1) \quad \text{or} \quad P(w_j = 1 | z = k)$$

$$P(\theta, z, w | \alpha, \beta) = P(\theta | \alpha) \prod_{j=1}^{N} P(z_j | \theta) P(w_j | z_j, \beta)$$

Dirichlet — multinomial

Learning in LDA: marginal distribution (likelihood) of a document $d = w$

$$P(w | \alpha, \beta) = \int_\theta \sum_z P(\theta, z, w | \alpha, \beta) \, d\theta = \int_\theta P(\theta | \alpha) \prod_{j=1}^{N} \sum_{z_j=1}^{K} P(z_j | \theta) P(w_j | z_j, \beta) \, d\theta$$

Given $\{w_1, \dots, w_M\}$, find (α, β) maximizing: $\alpha(\alpha, \beta) = \log \prod_{i=1}^{M} P(w_i | \alpha, \beta)$

Key problem is inferring latent (unobserved) variables posterior:

$$P(A|B,\theta) = \frac{P(A \cap B \cap \theta)}{P(B \cap \theta)} \qquad P(\theta, z | w, \alpha, \beta) = \frac{P(\theta, z, w | \alpha, \beta)}{P(w | \alpha, \beta)}$$

Learning with hidden variables corresponds to expectation-maximization

optimal ELBO is achieved when $Q(z)$ is equal to the latent

variable posterior $p(\theta, z | w, \alpha, \beta)$, but it is not tractable due to the couplings between β and θ in the summation over topics in the denominator:

$$p(w|\alpha,\beta) = \frac{\Gamma(\sum_{k=1}^{K}\alpha_k)}{\prod_{k=1}^{K}\Gamma(\alpha_k)} \int \prod_{k=1}^{K} \theta_k^{\alpha_k - 1} \left(\prod_{j=1}^{N} \sum_{k=1}^{K} \prod_{v=1}^{V} (\theta_k \beta_{kv})^{w_j^v} \right) d\theta$$

Variational inference: maximize the variational bound without using the optimal posterior solution. Write $q(z|\phi)$ function that is sufficiently similar to the posterior but tractable. $q(z|\phi)$ should be such that β and θ are no longer coupled. Fit ϕ parameter so that $q(z|\phi)$ is close to $p(w|\alpha,\beta)$ according to KL divergence.

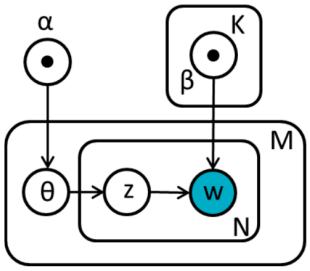
Sampling approach: construct a Markov chain on the hidden variables whose limiting distribution is the posterior

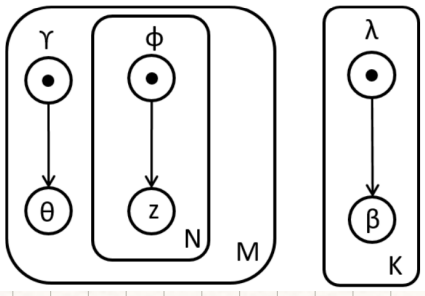
Assume that our distribution $q(z|\phi)$ is factorizable

$$q(z|\phi) = q(z_1, \dots, z_K | \phi) = \prod_{i=1}^{K} q(z_i | \phi_i)$$

z_i is independent and have their own parameters ϕ_i. Don't contain the true posterior because hidden variables are dependent.

VARIATIONAL LDA DISTRIBUTION:





given $\Phi = \{\gamma, \phi, \lambda\}$ as variational approximation parameters

$$q(\theta, z, \beta | \Phi) = q(\theta|\gamma) \prod_{n=1}^{N} q(z_n | \phi_n) \prod_{k=1}^{K} q(\beta_k | \lambda_k)$$

the model parameter $\Psi = \{\alpha, \beta\}$ of a simple distribution

$$p(\theta, z, w | \alpha, \beta) = p(\theta, z, w | \Psi)$$

find the ϕ, ψ that maximize the ELBO

$$\mathcal{L}(w, \Phi, \Psi) = E_q[\log p(\theta, z, w | \Psi)] - E_q[\log q(\theta, z, \Psi | \Phi)]$$

by alternate maximization:

while (little likelihood improvement):

fix Ψ: update variational parameters Φ^* (E-STEP)

fix $\Phi = \Phi^*$: update model parameter Ψ^* (M-STEP)

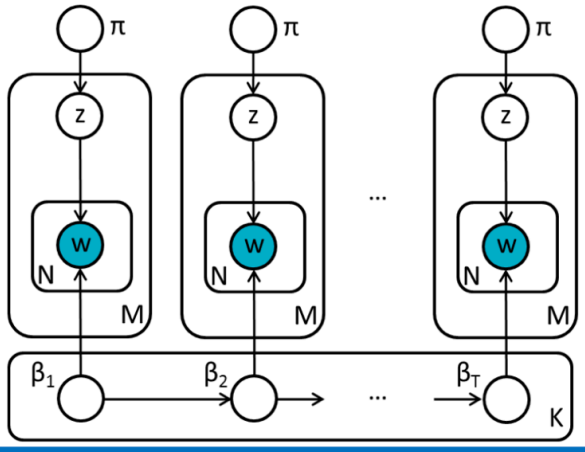
Unlike EM, variational EM doesn't garantee to reach local maxima of $\mathcal{L}$

LDA Applications: model for collections of high-dimension vectors whose attributes are multinomial distributions.

DYNAMICAL TOPIC MODELS

LDA assumes that the document order does not count

- What if we want to track topic evolution over time?
- Tracking how language changes over time
- Videos are image documents over time



Blei and Lafferty. Dynamic topic models, ICML 2006

Sampling: drawing a set of realization $X = \{x_1, \dots, x_L\}$ of a random variable x with distribution $p(x)$.

Useful in the case in which the $p(x)$ is **intractable** in computing

$$E_{p(x)}[f(x)] \approx \frac{1}{L} \sum_{l=1}^{L} f(x^l) = \hat{f}_x \rightarrow \text{estimatore}$$

- Boltzmann machine
- $P(\theta, z | w, \alpha, \beta)$ LDA
- posterior when non-conjugate priors are used.

In Bayesian models, parameters are RV, so we can learn the model parameters by sampling their posterior. In LDA we can learn the model parameter by sampling: $\theta, z, \beta \sim P(\theta, z, \beta | w, \alpha)$.

We can sample from the posterior to classify new instances as well:

$$\theta^*, z^* \sim P(\theta, z | w^*, \alpha, \beta)$$

the empirical distribution converge iff: $\lim_{L \to \infty} \frac{1}{L} \sum_{l=1}^{L} \mathbb{I}[x^l = i] = P(x = i)$

- the sampling approximation $\hat{f}_x$ of the expectation can be an **unbiased estimator** and can have **low variance** $\rightarrow$ need few sample to estimate the distribution

$E[\hat{\theta}] = \theta$

Let $\tilde{p}(x)$ the distribution over all possible realizations of the sampling set X, then $\hat{f}_x$ is an unbiased estimator if:

$$E_{\tilde{p}(x)}[\hat{f}_x] = E_{p(x)}[f(x)], \quad \text{true provided that } \tilde{p}(x^l) = p(x)$$

Variance of $\hat{f}(x)$ is: $E_{\tilde{p}(x)}\left[\left(\hat{f}_x - E_{\tilde{p}(x)}[\hat{f}_x]\right)^2\right]$

if we assume: $\tilde{p}(x^e) \simeq p(x^e)$ same marginals (valid sampler)

low approx. variance $\tilde{p}(x^e, x^{e'}) \simeq \tilde{p}(x^e)\,\tilde{p}(x^{e'})$ samples independence

we obtain: $E_{\tilde{p}x}\left[\left(\hat{f}_x - E_{\tilde{p}(x)}[\hat{f}_x]\right)^2\right] \simeq \frac{1}{L} \mathrm{Var}_{p(x)}[f(x)]$

We can use a small number of samples to accurately estimate the expectation

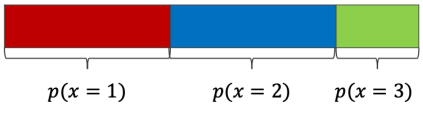
probability to obtain a sample set X ← $P(X) \neq p(x)$ → probability of the (multivariate) random variable

$p(x)$: distribution to sample. Doesn't depend on the sample

$\tilde{p}(x)$: distribution of the samples. Depends on the sampling procedure

Univariate sampling: Just need a random number generator that compute a value uniformely at random in $[0, 1]$

$$p(x) = \begin{cases} 0.4 & x = 1 \\ 0.4 & x = 2 \\ 0.2 & x = 3 \end{cases}$$



R	x
0.19	1
0.24	1
0.47	2
0.88	3
0.73	2
0.63	2
0.52	2
0.96	3

Multivariate sampling: build on univariate distribution $p(S)$, where S is a discrete variable with C^m states. We can use the univariate approach, but $O(C^m)$ states are computationally impossible.

We can rewrite the joint distribution with the chain rule:

$$p(s_1, \dots, s_m) = p(s_1) \cdot p(s_2 | s_1) \cdot p(s_3 | s_2, s_1) \cdots p(s_m | s_{m-1}, \dots, s_1)$$

we can sample the variables in the following order:

1 sample $\tilde{s}_1 \sim p(s_1)$

2 // $\tilde{s}_2 \sim p(s_2 | \tilde{s}_1)$ — univariate!

3 // $\tilde{s}_3 \sim p(s_3 | \tilde{s}_1, \tilde{s}_2)$

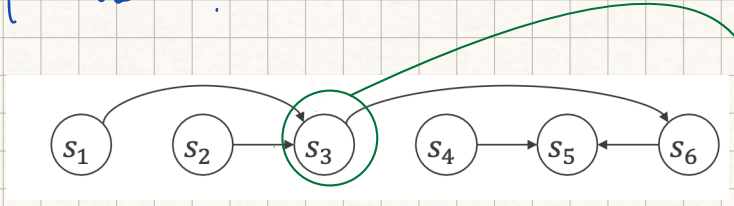
⋮

4 // $\tilde{s}_m \sim p(s_m | \tilde{s}_1, \dots, \tilde{s}_{m-1})$

ANCESTRAL SAMPLING

computing the distribution $p(s_i | s_{j<i})$ easily becomes exponential w.r.t. number of states!

If I know the Belief network,



I know the sampling order $\{s_1, s_2, s_4\} \to \{s_3\} \to \{s_6\} \to \{s_5\}$

AS performs exact sampling since each sample x^l is drawn from $p(x)$

V-shape: if I observe S_3, S_1 and S_2 become conditional dependent.

AS guarantee that the samples are independent, thus AS has low variance

Sampling with evidence: let's suppose that a subset of variables S_e are visible: $S = S_e \cup S_{\setminus e}$; $P(S_{\setminus e} | S_e) = \frac{P(S_{\setminus e}, S_e)}{P(S_e)}$

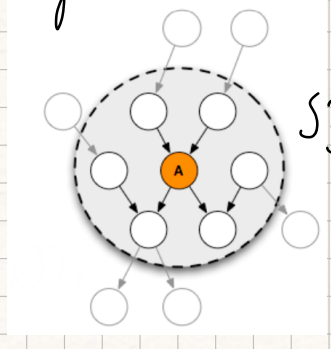
If we run AS we account the fact that $\Rightarrow$ V-shape: if I observe S_3, S_1 and S_2 become conditional dependent $\Rightarrow$ No more independent

We can run AS on the old structure and then discard any samples which do not match the evidence $\Rightarrow$ discard a lot of samples

Gibbs sampling: start from a sample $x_1 = \{s_1^1, \dots, s_n^1\}$ and update only one variable at a time $\Rightarrow$ samples are very much dependent

$\Rightarrow$ high variance

Definition: during the $l+1$ iteration, we select a variable s_j; we sample its value according to:



$$s_j^{l+1} \sim P(s_j | s_{\setminus j}) = \frac{1}{Z} P(s_j | pa(s_j)) \prod_{k \in ch(j)} P(s_k | pa(s_k))$$

where $s_{\setminus j}$ are clamped to $\{s_1^l, \dots, s_{j-1}^l, s_{j+1}^l, \dots, s_n^l\}$

Depends only on the Markov Blanket of s_j

Dealing with evidence is easy! We just do not select a variable

LDA Gibbs Sampling: start from an initial guess $\{z_{ij}^0, \theta^0, \beta^0\}$

1 $z_{ij}^{l+1} \sim p(z_{ij} | w, z_{ij}^{l-1}, \theta^l, \beta^l, \alpha)$

2 $\theta_i^{l+1} \sim p(\theta_i | w, z^{l+1}, \beta^l, \alpha)$

3 $\beta^{l+1} \sim p(\beta | w, z^{l+1}, \theta^{l+1}, \alpha)$

} learning part

The convergence criteria is based on $p(z, w, \theta | \beta, \alpha)$. the procedure terminates when the likelihood stop increasing

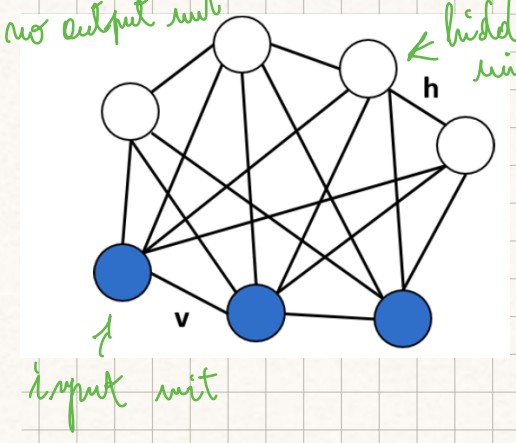
The Gibbs sampling draws a new sample x^l from $q(x^l | x^{l-1})$, we are not sampling from $p(x)$! If we compute the limit to $L \to \infty$, the series $\{x_1, \dots, x_L\}$ converges to sample taken from $p(x)$

Markov Chain Monte Carlo: let $q(x^{l+1} | x^l)$ the MC state-transition distribution, we must ensure that:

- MC is irreducible → it is possible to reach any state from anywhere
- MC is aperiodic → at each time-step, we can be anywhere

Hence, the MC has a stationary distribution. there are different algorithm which converge to $p(\cdot)$.

Boltzmann Machines: an example of Markov random field



- visible RV $v \in \{0,1\}$ · latent RV $h \in \{0,1\}$
- $s = [v, h]$ → general unit of the BM.
- linear energy function:

$$E(s) = -\frac{1}{2}\sum_{ij} M_{ij} s_i s_j - \sum_j b_j s_j = -\frac{1}{2} s^T M s - b^T s$$

("weight" M, "bias" b)

with symmetric and no self-recurrent connectivity

- Model parameters $\theta = \{M, b\}$ encode the interaction between the variables

We can see Boltzmann Machines as a type of recurrent neural network

- The state of a unit at a given timestep is sampled from a given probability distribution
- The network learns a probability distribution $P(V)$ from the training pattern. Network activity is a sample from posterior probability given inputs (visible data).
- BM have spiking point neuron with binary output s_j. At each time interval $(t+1 \equiv t+\Delta t)$, the neuron can emit a spike signal with probability $p_j^{(t)}$

$$s_j^{(t)} = \begin{cases} 1, & p_j^{(t)} \\ 0, & 1 - p_j^{(t)} \end{cases}$$

$p_j^{(t)} \simeq \sigma(x_j^{(t)})$ function of local potential x_j

Network of N neurons with binary activations s_j

- Weight matrix $M = [M_{ij}]_{i,j} \in \{1, \dots, N\}$
- bias vector $b = [b_j]_j \in \{1, \dots, N\}$

local neuron potential x_j defined as usual

net for neuron j at time $t+1$: $x_j^{(t+1)} = \sum_{i=1}^{N} M_{ij} s_i^{(t)} + b_j$

a chosen neuron is active with probability :

$$p_j^{(t+1)} = P(s_j^{(t+1)} = 1 \mid s^t) = \sigma(x_j^{(t+1)}) = \frac{1}{1 + e^{-x_j^{(t+1)}}}$$

let's suppose that we update each neurons in parallel (every Δt)

State transition : $P(s^{(t+1)} \mid s^{(t)}) = \prod_{j=1}^{N} P(s_j^{(t+1)} \mid s^t) = T(s^{(t+1)} \mid s^{(t)})$

yielding a Markov process for state update

$$P(s^{(t+1)} = s') = \sum_{s} T(s' \mid s) P(s^{(t)} = s)$$

Glauber dynamics : one neuron at random is chosen for update at each step. Spiking neuron output s has a stationary distribution

for the network at equilibrium state when its connectivity is symmetric. Given F_j as state flip operator for j-th RV $s_j^{t+1} = F_j\, s_j^{(t)}$

$$T(s^{(t+1)} | s^{(t)}) = \frac{1}{n} P(s_j^{(t+1)} | s^t)$$

Undirected connectivity enforces detailed balance condition

$$P(s)\, T(s'|s) = P(s')\, T(s|s')$$

ensure reversible transition guaranteeing existence of equilibrium (Boltzmann - Gibbs) distribution

Joint probability of BM: $P_\infty(s) = \frac{e^{-E(s)}}{Z}$ → energy function

Z → $Z = \sum_s e^{-E(s)}$ partition function ensure $\sum_x P(x) = 1$

Learning of a BM: Boltzmann machines can be trained so that the equilibrium distribution $P_\infty(s)$ tends towards any arbitrary distribution across binary vectors given samples from that distribution. BM becomes a universal approximator of probability mass function over discrete variables.

For the sake of simplicity, bias b is absorbed into weight matrix W and we consider only visible RV $s = v$. Then we maximize the log likelihood

$$\mathcal{L}(M) = \frac{1}{L} \sum_{l=1}^{L} \log P(v^l | M)$$

L: # dataset; M: parameters

Gradient approach: first, consider the gradient for a single pattern

$$\frac{\partial P(v|M)}{\partial M_{ij}} = -\langle v_i v_j \rangle + v_i v_j$$

free expectation: $\sum_v P(v) v_i v_j$

then the log-likelihood

$$\frac{\partial \mathcal{L}}{\partial M_{ij}} = -\langle v_i v_j \rangle + \langle v_i v_j \rangle_c$$

clamped expectation: $\frac{1}{L} \sum_{l=1}^{P} v_i^l v_j^l$

Hebbian learning: $\frac{\partial \mathcal{L}}{\partial M_{ij}} = \langle v_i v_j \rangle_c - \langle v_i v_j \rangle$

$\langle v_i v_j \rangle_c$: Wake; $\langle v_i v_j \rangle$: Dream

wake part is the standard Hebb rule applied to the empirical distribution of data that the machines see coming in from the outside world.

dream part is an anti-hebbian term concerning correlation between units when generated by the internal dynamics of the machine

Details: $P(v^l | M) = \frac{\exp(-E(v^l))}{Z = \sum_v \exp(-E(v))}$

$$\log P(v^l|H) = \underbrace{-E(v^l)}_{(1)} \underbrace{- \log Z}_{(2)}$$

$$+v^T H v \Rightarrow \sum_{p,q} H_{pq} v_q v_p$$

$$\frac{\partial \log f(x)}{\partial x} = \frac{f'(x)}{f(x)}$$

$$\frac{\partial \log P(v^l|H)}{\partial H_{ij}} = \underbrace{v_i^l v_j^l}_{(a)} - \sum_v \underbrace{\frac{\exp^{-E}}{Z}}_{(b)} \cdot v_i v_j = v_i^l v_j^l - \sum_v P(v|H) v_i v_j$$

$P(v|H)$

$$E_{v \sim P(v|H)}[v_i v_j]$$

$$\langle v_i^l v_j^l \rangle_H$$

$$= v_i^l v_j^l - \langle v_i v_j \rangle$$

$$\frac{1}{L} \sum_l \left(v_i^l v_j^l \right) - \langle v_i v_j \rangle_H$$

$$E_{v \sim Data}[v_i v_j]$$

$$\langle v_i v_j \rangle_0$$

$$v_i, v_j \in \{0, 1\}$$

$$\Delta H_{ij} = \frac{\partial \sum \log P}{\partial H_{ij}} = \langle v_i v_j \rangle_0 - \langle v_i v_j \rangle_H = 0$$

a matching of two expectation

correlation of the variable w.r.t. data & model should match

$$H'_{ij} = H_{ij} + \lambda \Delta H_{ij}$$ Hebbian learning

clamped expectation $\langle \cdot, \cdot \rangle_c$ is the expectation under the data

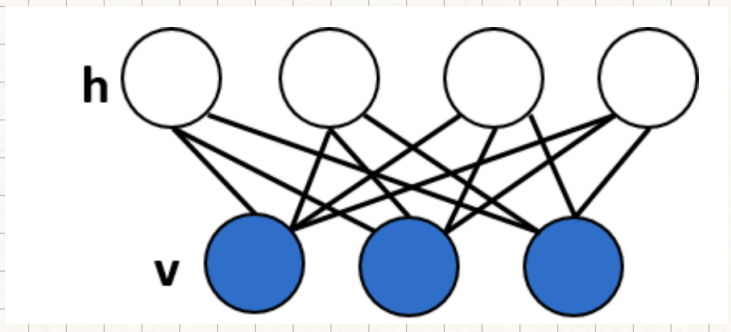
Learning with hidden variable: this time $s = [v, h]$

$$\frac{\partial P(v|\theta)}{\partial W_{ij}} = \sum_h s_i s_j P(h|v) - \sum_s s_i s_j P(s) = \langle s_i s_j \rangle_c - \langle s_i s_j \rangle$$

expectation is intractable due to the partition function Z.

Restricted Boltzmann machines

connection only between hidden and visible variable.



Energy function: $E(v, h) = -v^T W h - b^T v - c^T h$

bipartite graph

hidden units are conditionally independent given visible units and viceversa

$$P(h_j|v) = \sigma\left(\sum_i W_{ij} v_i + c_j\right)$$
$$P(v_i|h) = \sigma\left(\sum_j W_{ij} h_j + b_i\right)$$

} update in batch

Training RBM:

$$\frac{\partial L}{\partial W_{ij}} = \langle v_i h_j \rangle_c - \langle v_i h_j \rangle_M$$

a Gibbs sampling approach:

Wake part : • clamp data on v

• sample $v_i h_j$ for all pairs of connected units

• repeat for all elements of dataset

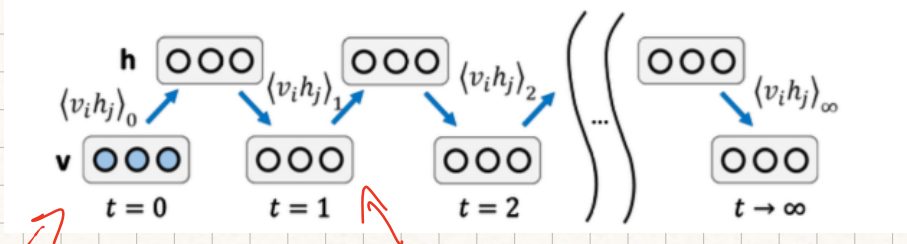
Dream part : • don't clamp units

• let network reach equilibrium

• sample $v_i h_j$ for all pairs of connected units

• repeat many times to get a good estimate

It's difficult to obtain an unbiased sample of the second term



plugging-in the data

reconstruction

1 starting with a training vector on the visible units

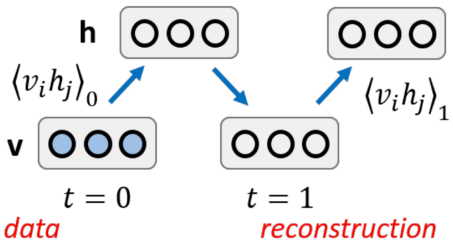
2 alternate between updating all the hidden units in parallel and updating all the visible units in parallel

3 iterate

$$\frac{\partial L}{\partial w_{ij}} = \underbrace{\langle v_i h_j \rangle_0}_{\text{data}} - \underbrace{\langle v_i h_j \rangle_\infty}_{\text{model}}$$

Gibbs sampling is slow to converge (high variance)

Contrastive divergence learning: approximation of the gradient of the log-likelihood, not following the gradient closely!

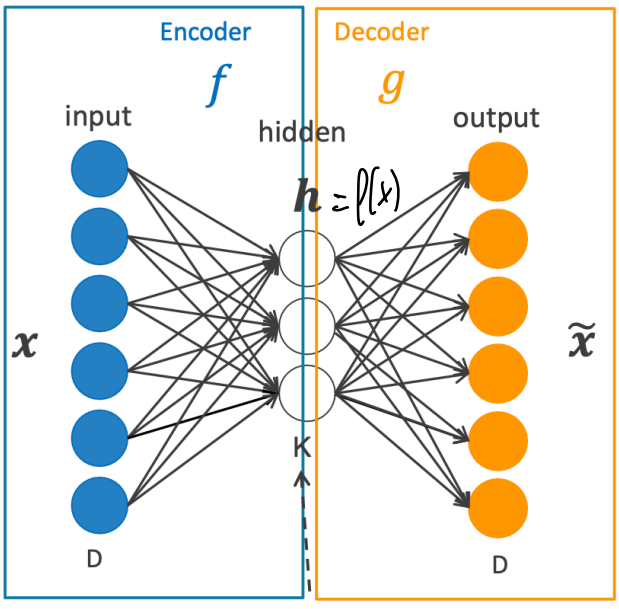


1. Clamp a training vector $\boldsymbol{v}^l$ on visible units
2. Update all hidden units in parallel
3. Update all the visible units in parallel to get a *reconstruction*
4. Update the hidden units again

$$\underbrace{\langle v_i h_j \rangle_0}_{data} - \underbrace{\langle v_i h_j \rangle_1}_{reconstruction}$$

MANCANO LO CNN

Autoencoder:



Latent space projection

train a model to reconstruct the input data. $K < D$ or h can be sparsely active

$L(x, \tilde{x}) = L(x, g(f(x)))$ loss minimization

In general we train nonlinear AE (Relu, sigmoid), with $K \geq D$, that can learn non trivial identity.

Sparse Autoencoder: we add a penalty term to h (vector of neurons that encode the input) in order to make them sparsely active

$$J_{SAE}(\theta) = \sum_{x \in S} \left(L(x, \tilde{x}) + \lambda \Omega(h) \right)$$

$$\Omega(h) = \Omega(f(x)) = \sum_j |h_j(x)|$$

↳ penalize activation function, not weight of the AE

training with regularization corresponds to MAP inference

$$\max \log P(x, h) = \max \left(\log P(x|h) + \log P(h) \right)$$

likelihood

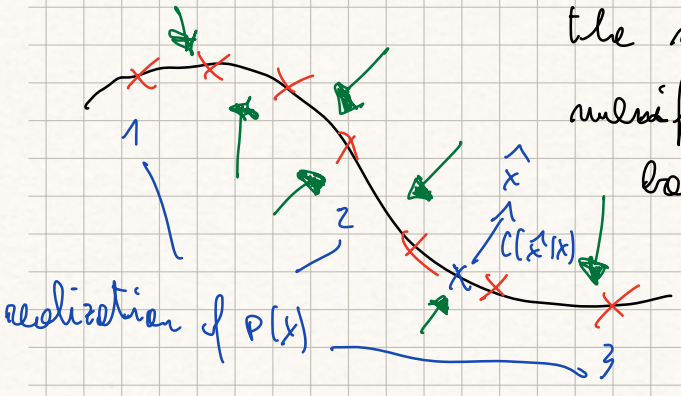
$$P(h) = \frac{\lambda}{2} \exp\left(-\frac{\lambda}{2} |h|_1\right)$$

↓ PRIOR

$$\Omega(h) = \lambda |h|_1$$

Denoising Autoencoder (DAE): train to minimize $L(x, g(f(\tilde{x})))$ where $\tilde{x} = x + \varepsilon$, $\varepsilon \sim N(0, \sigma)$. The keypoint is that we want to learn a representation that is robust to different noise. We can have a probabilistic interpretation of learning the denoising distribution $P(x|\tilde{x})$ by minimizing $-\log P_d(x | h = f(\tilde{x}))$

Manifold learning: we can imagine that data belongs to a lower-dimensional space called manifold.



the noise push the data away from the manifold. the DAE reconstruct $\hat{x}$ by pushing back the data to the manifold, but the reconstructed $\hat{x}$ could not be the same as the original x.

DAE learn a vectorial field (green arrow) that approximate the gradient of the unknown data generation distribution

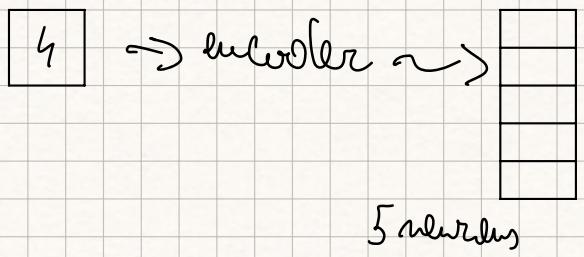
$C(\hat{x}|x) = N(\vec{x}|x, \sigma^2)$

$$g(h) - x \simeq \frac{\partial \log P(x)}{\partial x}$$

& iterate the reconstruction of $\hat{x}$ many time in order to

move closer the data to the manifold ⇒ increase likelihood

Manifold assumption: assume data lies on a lower dimensional non-linear manifold, since variables in data are typically dependent



I use the latent space to capture only the meaningfull direction of variations in the original data (size, orientation, shape...)

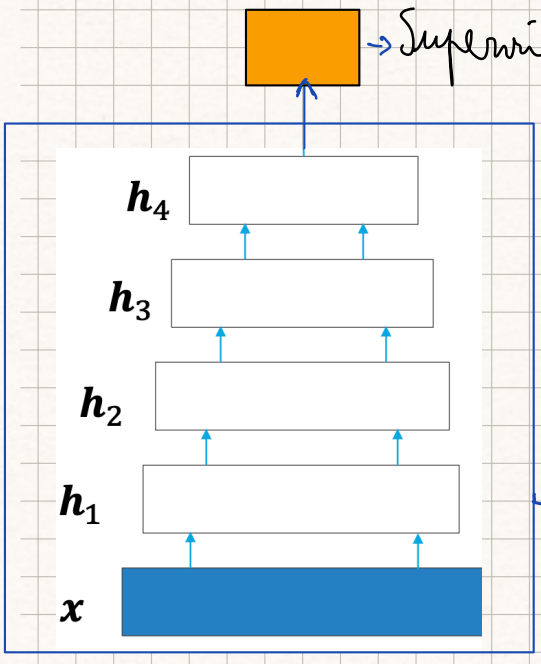
DAE are capable to represent changes in the manifold direction in order to reconstruct training examples

Contractive Autoencoders: robust to infinitesimal variations of inputs

$$J_{CAE}(\theta) = \sum_{x \in S} \left(L(x, \hat{x}) + \lambda \Omega(h) \right)$$

$$\Omega(h) = \Omega(f(x)) = \left\| \frac{\partial f(x)}{\partial x} \right\|_{Frobenius}$$

high when $f(x)$ changes a lot, but not x

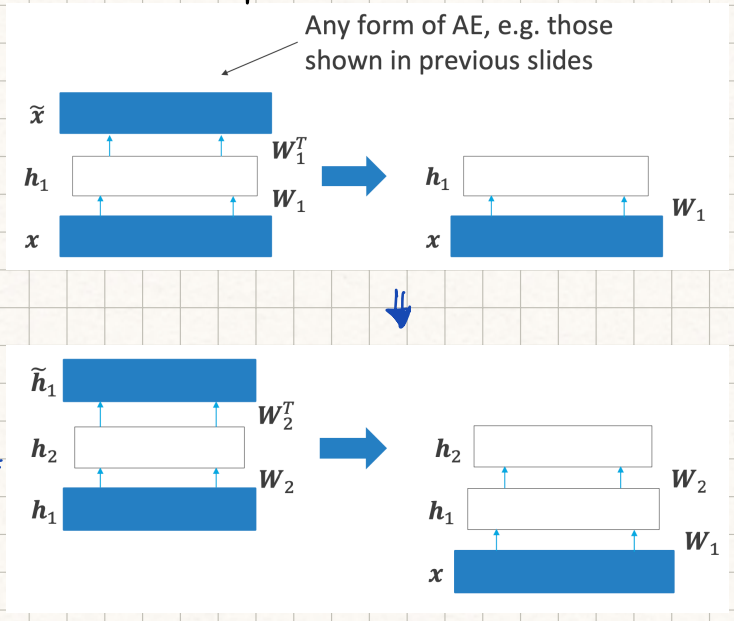


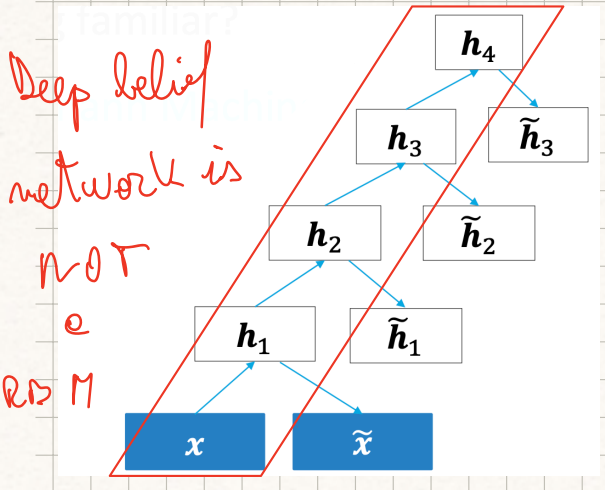
→ Supervised learning

We can stack several autoencoders such that we reduce the input dimension. Useful for: Data visualization, exploration, indexing, ...

→ Unsupervised learning: incremental unsupervised construction of Deep AE

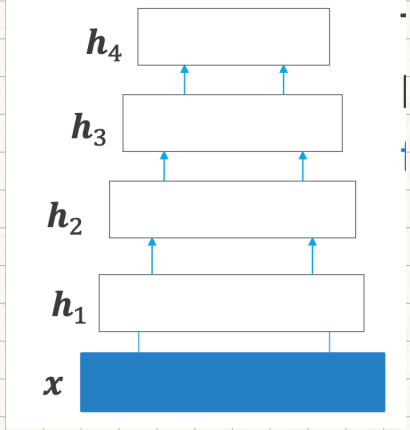
resemble a RBM, we can use it to perform layerwise pretraining and learn matrices W_i





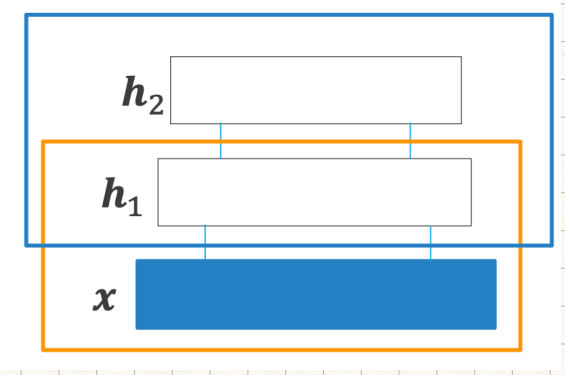
Deep belief network is NOT a RBM

Deep Boltzmann Machine: DBN but without arrow directions



$$P(h_j^1 \mid x, h^2) = \sigma\left(\sum_i w_{ij}^1 x_i + \sum_m w_{jm}^2 h_m^2\right)$$

$$P(x_i \mid h^1) = \sigma\left(\sum_j w_{ij}^1 h_j^1\right)$$



1) Pretraining the first layer entails fitting this model: $P(x|\theta) = \sum_{h^1} P(h^1|W^1)\, P(x|h^1, W^1)$

$\downarrow$

$P(h^1|W^1) = \sum_{x} P(h^1, x|W^1)$

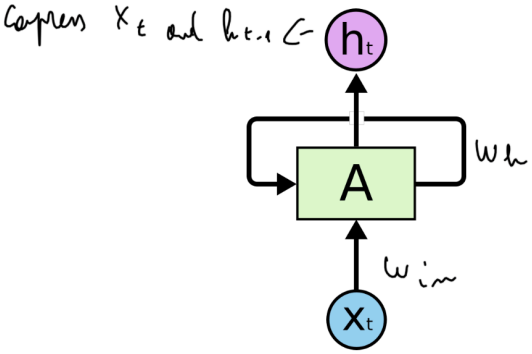
2) Pretraining the second layer changes h_1 prior by: $P(h^1|W^2) = \sum_{h_2} P(h^1, h^2|W^2)$

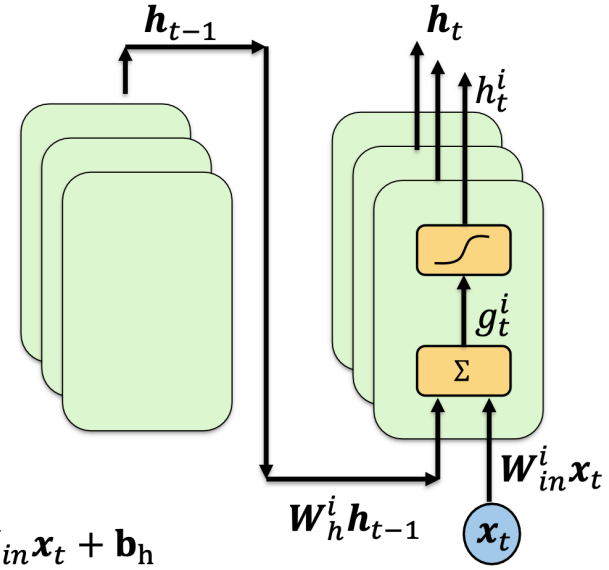
When putting things together we need to average between $P(h^1|W^1)$ and $P(h^1|W^2)$

When training with more than two RBM apply trick to first and last layers and halve weights of intermediate RBM

RNN

$$y_t = f(W_{out} h_t + b_{out})$$

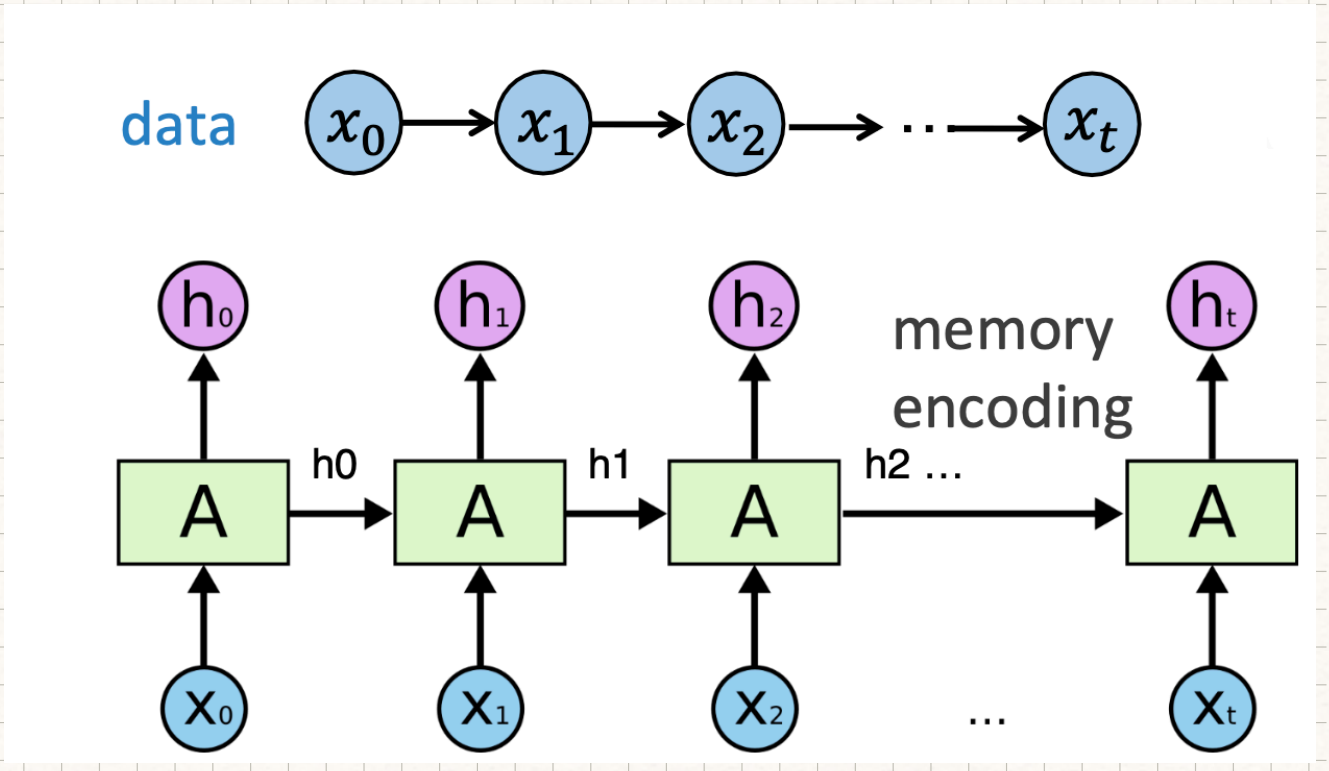




$$h_t = tanh(g_t)$$

$$g_t(h_{t-1}, x_t) = W_h h_{t-1} + W_{in} x_t + b_h$$

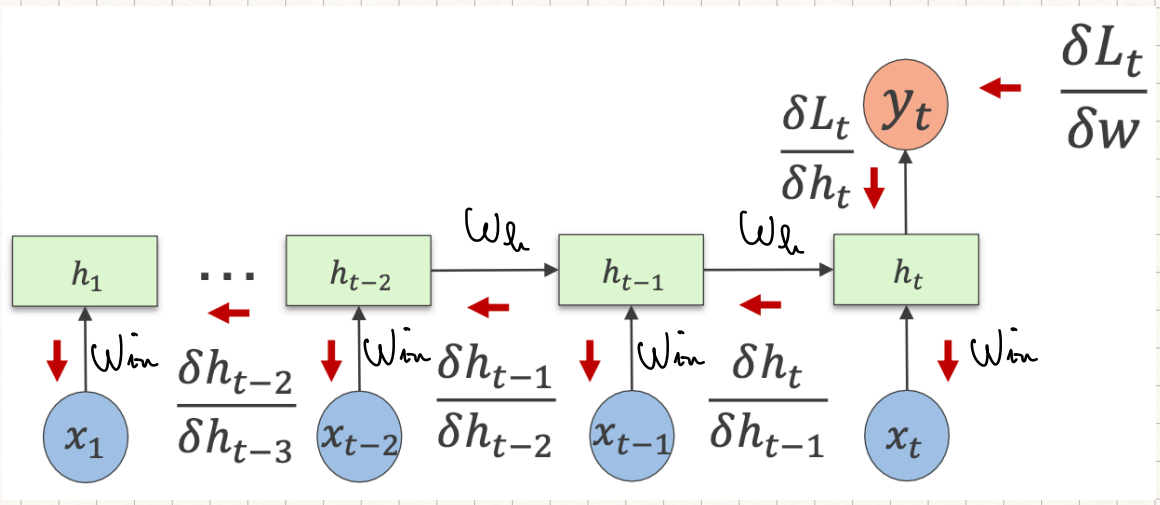
Unfolding the sequence



learning long term dependencies is difficult due to gradient vanishing / exploding => instability and oscillations

⇓

no learning



$$\frac{\partial L_t}{\partial w} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial w}$$

→ Jacobian matrix

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \times \frac{\partial h_{t-1}}{\partial h_{t-2}} \times \frac{\partial h_{t-2}}{\partial h_{t-3}} \times \dots \times \frac{\partial h_{k+1}}{\partial h_k}$$

$$\frac{\partial L_t}{\partial w} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial h_t} \left(\prod_{\ell=k}^{t-1} \frac{\partial h_{\ell+1}}{\partial h_\ell} \right) \frac{\partial h_k}{\partial w}$$

given $h_\ell = \tanh(W_{in} x_\ell + W_{h\ell} h_{\ell-1})$ $\quad \partial \tanh = \operatorname{sech}^2 = 1 - \tanh^2$

$$\frac{\partial h_{\ell+1}}{\partial h_\ell} = D_{\ell+1} W_{h\ell}^T \qquad D_{\ell+1} = \operatorname{diag}\left(1 - \tanh^2(W_{in} x_\ell + W_{h\ell} h_{\ell-1})\right)$$

$$\frac{\partial L_t}{\partial h_k} = \frac{\partial L_t}{\partial h_t} \cdot \left(\prod_{\ell=k}^{t-1} \frac{\partial h_{\ell+1}}{\partial h_\ell} \right) = \frac{\partial L_t}{\partial h_t} \prod_{\ell=k}^{t-1} D_{\ell+1} W_{h\ell}^T$$

Bounding the gradient $\left\| \frac{\partial L_t}{\partial h_k} \right\| =$

$$= \left\| \frac{\partial L_t}{\partial h_t} \prod_{k=1}^{t-1} D_{k+1} W_{hh}^T \right\| \leq \left\| \frac{\partial L_t}{\partial h_t} \right\| \prod_{k=1}^{t-1} \left\| D_{k+1} W_{hh}^T \right\| \leq$$

$$\leq \left\| \frac{\partial L_t}{\partial h_t} \right\| \prod_{k=1}^{t-1} \sigma(D_{k+1}) \sigma(W_{hh}^T)$$

$\sigma < 1$ vanishing

$\sigma > 1$ exploding

spectral radius $= \max\{|\lambda_1|, \dots, |\lambda_n|\}$

$\sigma = 1$ good solution

Gradient clipping (for gradient exploding)

$g = \frac{\partial L_t}{\partial W}$ if $\|g\| \geq \theta_0$ then $g = \frac{\theta_0}{\|g\|} \cdot g$

Vanilla RNN recap:

1. expressive and general model
2. gradient problem
3. the recurrence limits the parallelization opportunities

2) Sigmoid and tanh are always $\sigma < 1$. Identity function

$\sigma = 1$ for $W^T W = I$ orthogonal

$W^H W = I$

$W = I$

Orthogonal matrix + linear activation

$$h_{t+1} = W_h^T h_t + W_{in} x_{t+1}$$

$$\frac{\partial h_{t+1}}{\partial h_t} = W_h \qquad \left\| \frac{\partial h_{t+1}}{\partial h_t} \right\| = \|W_h\| = 1$$

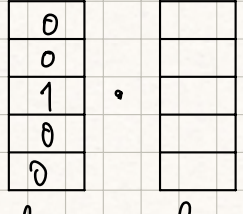
(CEC) Constant error propagation: · identity activation function
· identity weight matrix

$$h_t = h_{t-1} + \hat{c}(x_t)$$

Has the desired spectral properties but does not work in practice as it quickly saturates memory (e.g. with replicated/non-useful inputs and states). We want to be able to "control the forgetting".

Forget gate: add forget gate to CEC
↳ soft reset units

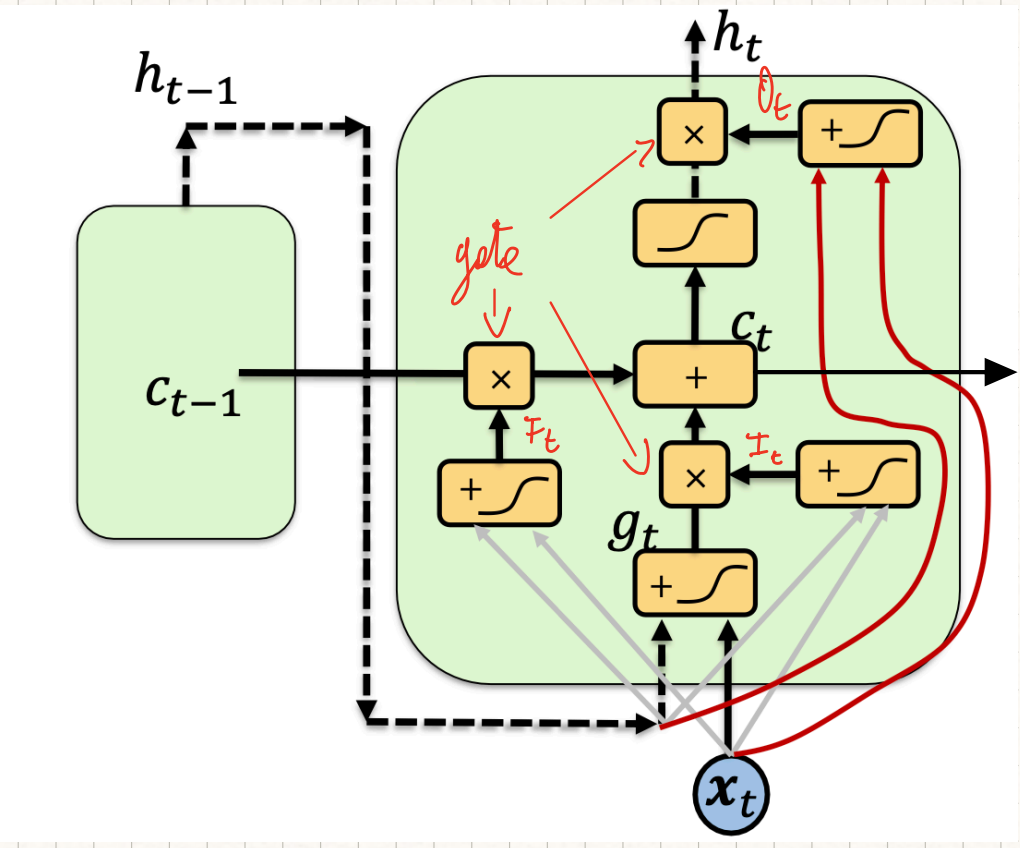
$$f_t = \sigma(W_{fh} h_{t-1} + W_{fx} x_t + b_f) \longrightarrow$$



$$h_t = f_t \odot h_{t-1} + \hat{c}(x_t)$$

· Avoid saturation, completely forgets the past, no guarantees about constant propagation

LSTM cell



- internal memory c_t
- input gate: controls how inputs contribute to the internal state

$$I_t(x_t, h_{t-1}) = \sigma \begin{cases} 1 \\ 0 \end{cases}$$

↑ logistic sigmoid

- forget gate: controls how past internal state c_{t-1} contributes to c_t

$$F_t(x_t, h_{t-1}) = \sigma \begin{cases} 1 \\ 0 \end{cases}$$ → logistic sigmoid

- output gate: controls what part of the internal state is propagated out of the cell

$$O_t(x_t, h_{t-1}) = \sigma \begin{cases} 1 \\ 0 \end{cases}$$ → logistic sig.

Equations:

1) Compute activation of input and forget gates

$$I_t = \sigma(W_{Ih} h_{t-1} + W_{Iin} x_t + b_I)$$

$$F_t = \sigma(W_{Fh} h_{t-1} + W_{Fin} x_t + b_F)$$

2) Compute input potential and internal state

$$g_t = \tanh(W_{ch} h_{t-1} + W_{in} x_t + b_h)$$

$$C_t = F_t \odot C_{t-1} + I_t \odot g_t$$

3) Compute output gate and output state

$$O_t = \sigma(W_{oh} h_{t-1} + W_{oin} X_t + b_o)$$

$$h_t = O_t \odot \tanh(C_t)$$

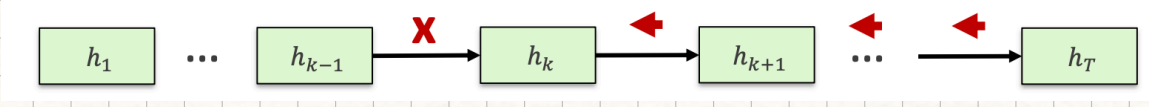
LSTM layers extract information at increasing levels of abstraction

Dropout: disconnect some units during the training process. I create a set of binary mask, drawn from a binomial dist., that I will apply to the units output. It is regulated by hyperparameter, the higher the more units will be disconnected. It prevents the LSTM units to coadapt. Models with many cells reproduce a counter effect (it's like having more LSTM working together). Need to adapt prediction phase.

Useful for obtaining confidence interval for a given input. I can apply different mask in the prediction phase to the models, given the same input.

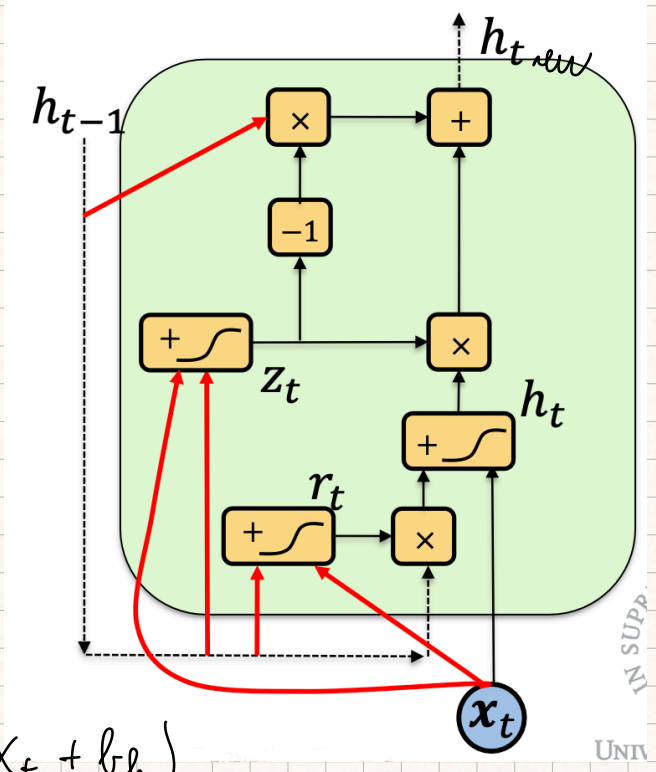
Minibatch LSTM and Truncated Backprop

If we have a huge dataset, we could use the minibatch approach. Another way to deal with huge sequences is the truncated gradient propagation which blocks the propagation after a number of iterations.



(GRU) Gated recurrent unit

Reset gate acts directly on output. It controls how much of the previous state we might still want to remember. r_t controls it



$$h_t = \tanh\left(W_{hh}\left(r_t \odot h_{t-1}\right) + W_{hin} x_t + b_h\right)$$

Update gate allows us to control how much of the new state h_t is just a copy of the old state h_{t-1}.

$$h_{t_{new}} = (1 - z_t) \odot h_{t-1} + z_t \odot h_t$$

Reset and **update** gates when coupled act as **input** and **forget** gates.

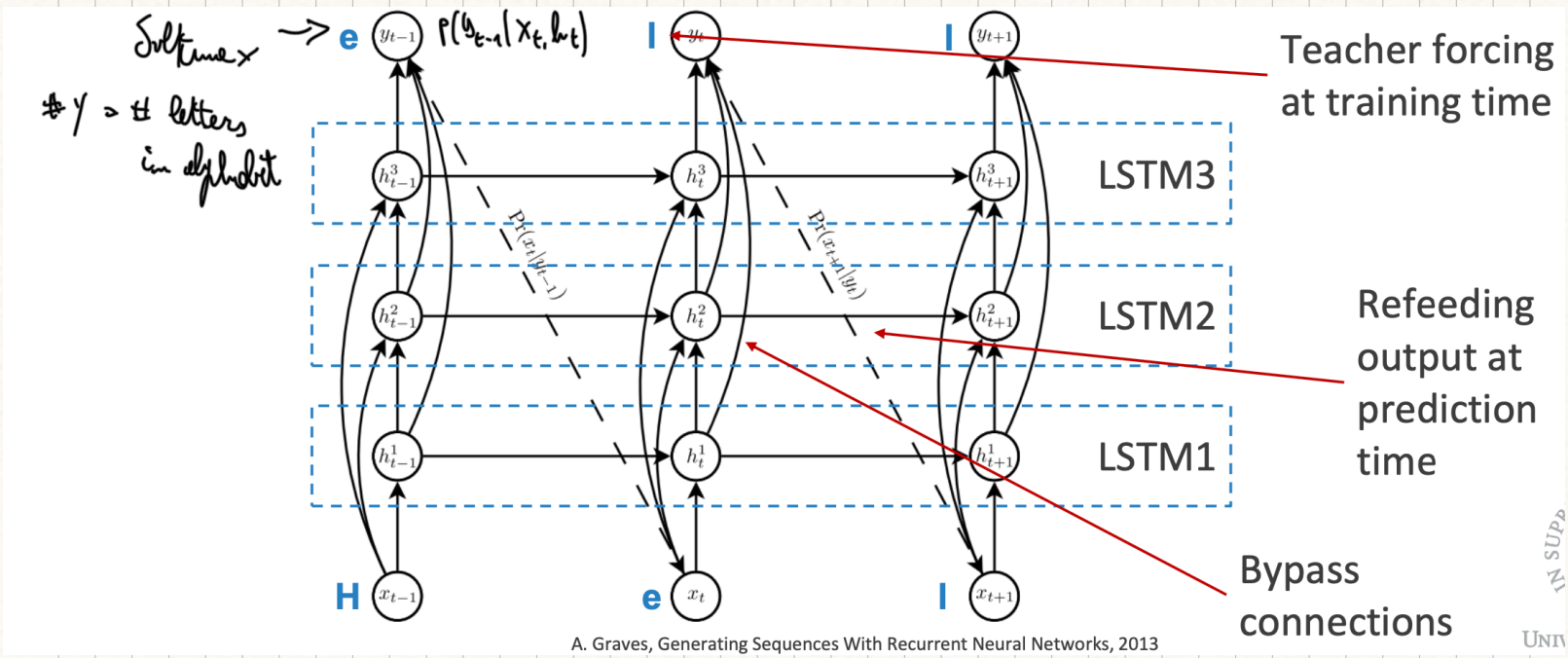
$$z_t = \sigma\left(W_{zh}\, h_{t-1} + W_{zin}\, x_t + b_z\right)$$

$$r_t = \sigma\left(W_{rh}\, h_{t-1} + W_{rin}\, x_t + b_r\right)$$

The main difference with LSTM is that a single gating unit simultaneously controls the forgetting factor and the decision to update the output unit

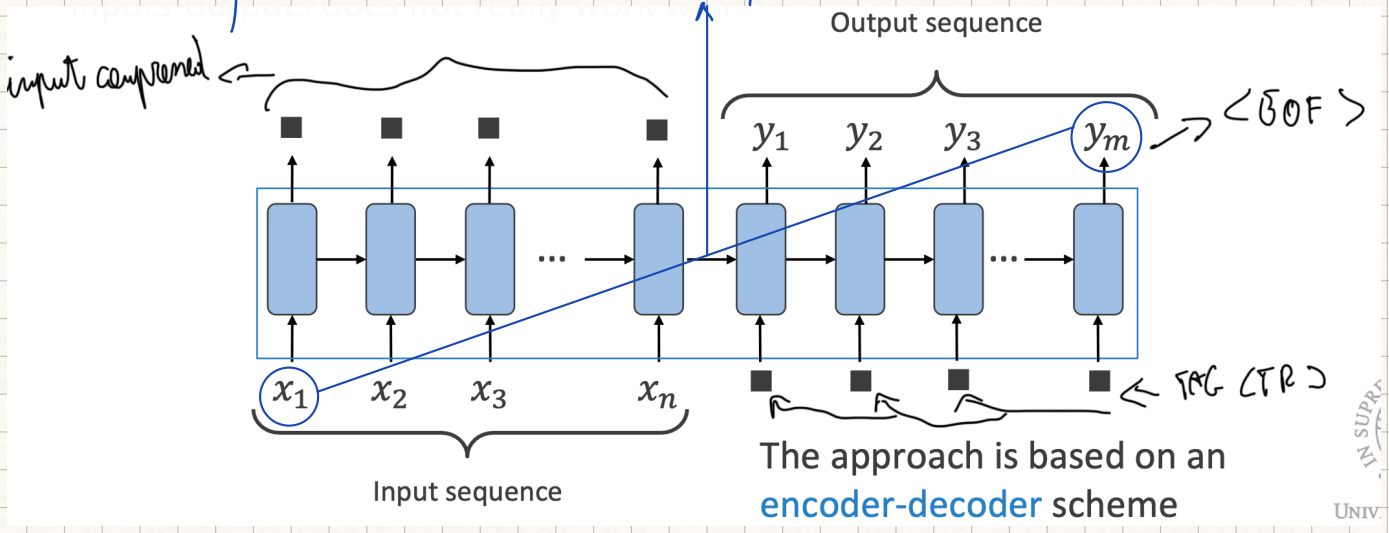
LSTM language model

teacher forcing: used at the early phase of training (like simulated annealing). During the training you will use the predicted output (probably a wrong prediction) in order to gives the model the capability to recover from error.

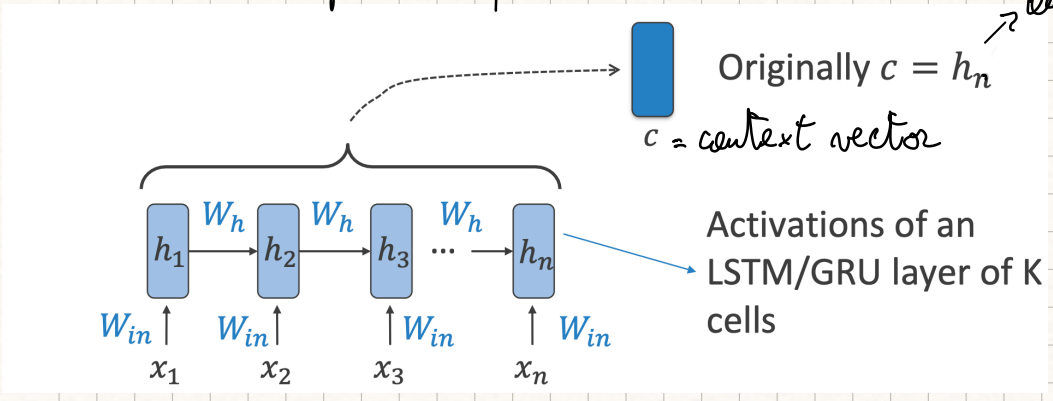


A. Graves, Generating Sequences With Recurrent Neural Networks, 2013

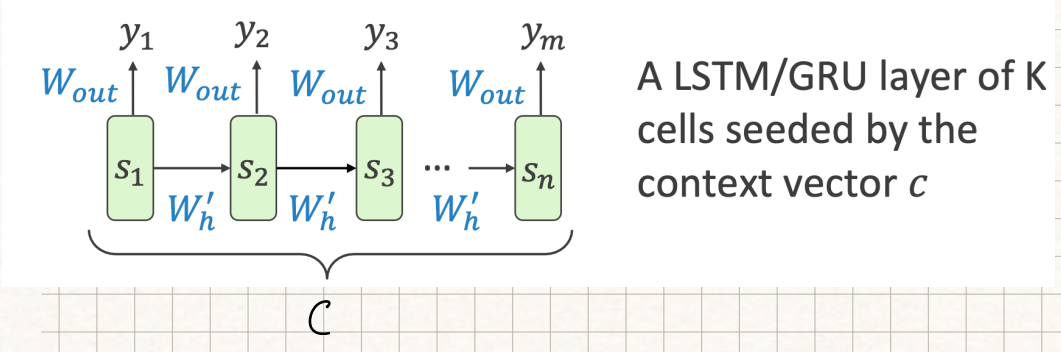
Sequence to sequence task — Very distance



Encoder: produce a compressed and fixed length representation c of all the input sequence $x_1, \dots, x_n$

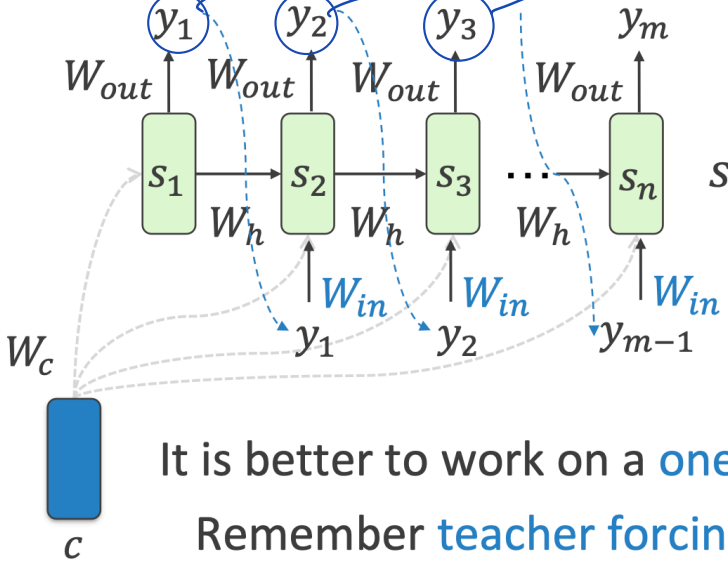


Decoder: starting from the context vector c, it generates the output sequences



How to pass to the decoder?

Teacher forcing



$$s_i = f(c, s_{i-1}, y_{i-1})$$

It is better to work on a one-step-ahead scheme

Remember teacher forcing (only) at training time

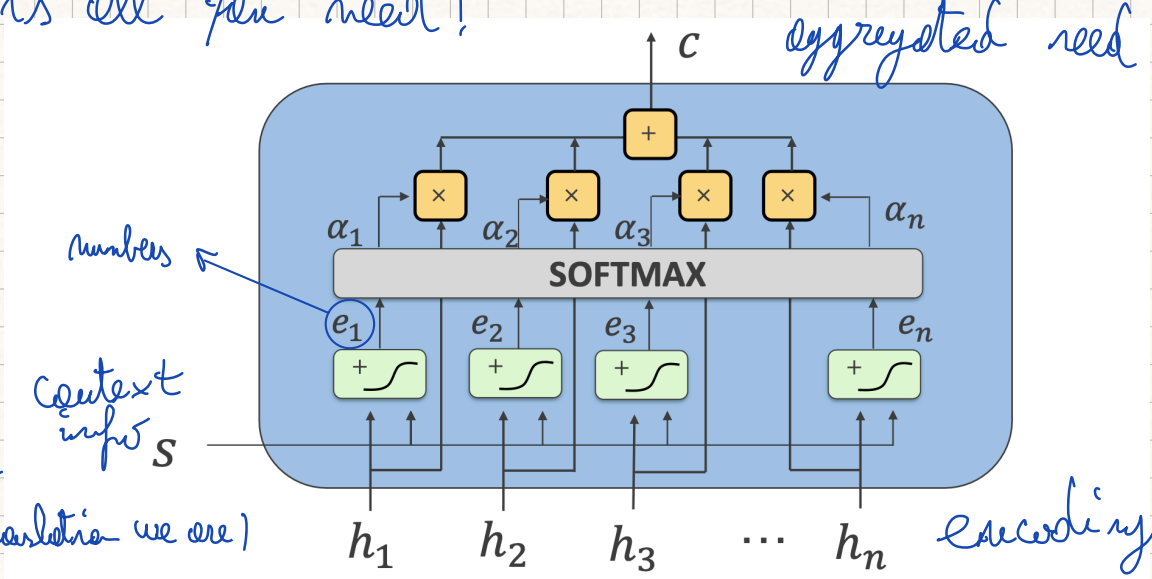
c is contextual information kept throughout output generation

Encoder - Decoder can be trained end-to-end or independently

Reversing the input sequence during the encoding phase could lead to better results. In fact, the last token encoded will be likely the first to be translated.

Encoder-decoder scheme assumes the hidden activation of the last input element summarizes sufficient information to generate the output (Bias toward the most recent part). We need attention

Attention is all you need!



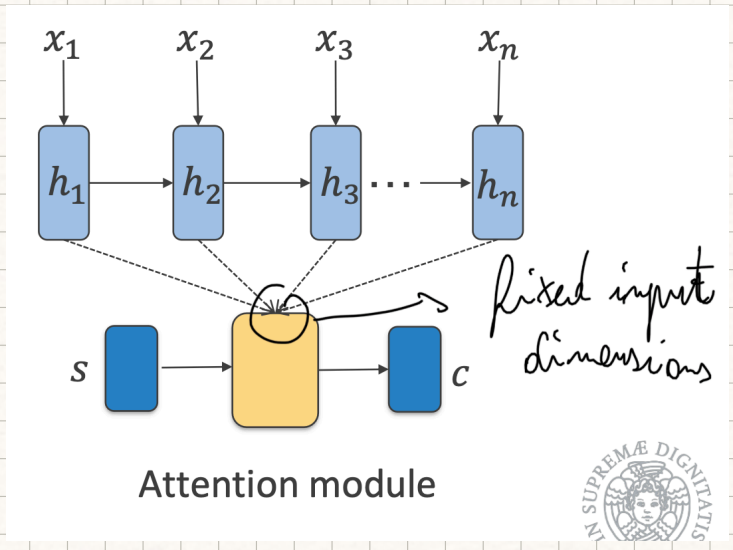
RELEVANCE

$e_i = a(s, h_i)$ units of a NN (tanh layer) fusing each encoding with current context s

SOFTMAX: differentiable max vector operator $\alpha_i = \frac{\exp(e_i)}{\sum_j \exp(e_j)}$

$\sum_i \alpha_i = 1$ convex combination

VOTING: aggregated read by (soft) attention voting $c = \sum_i \alpha_i h_i$

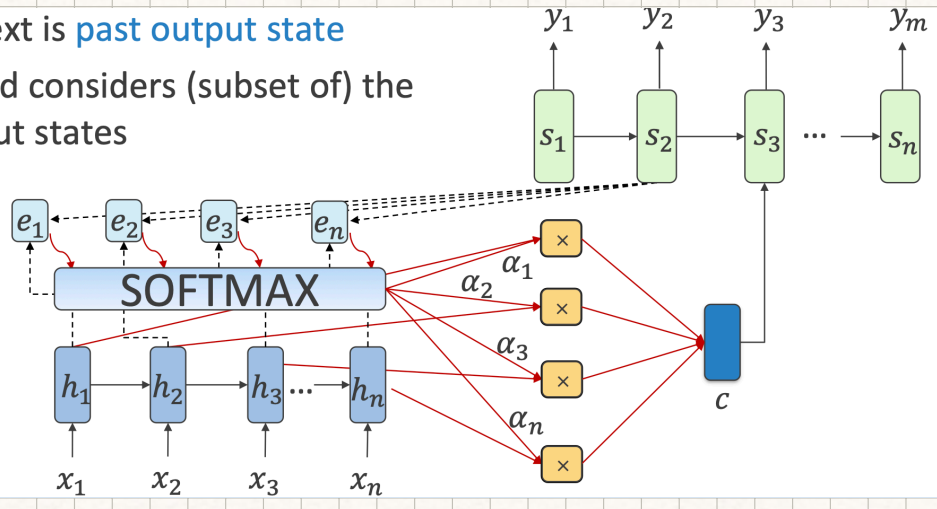


the attention module has a fixed input dimension

ATTENTION IN SEQ 2 SEQ

Context is past output state

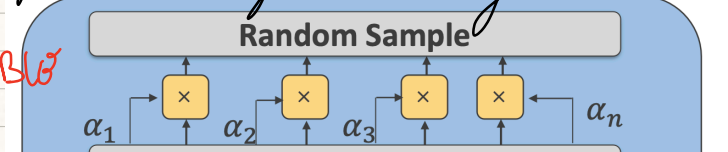
Seed considers (subset of) the input states



context s is the past output state

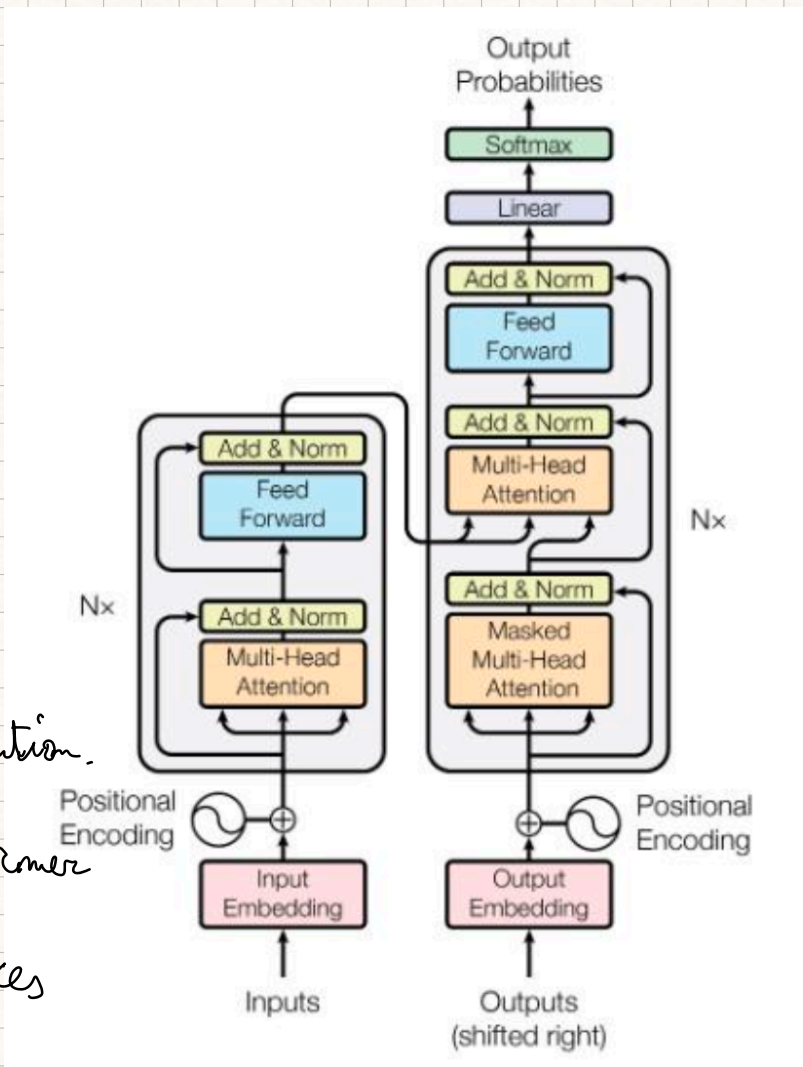
Hard attention: sample a single encoding h_i using probability α_i

NOT DIFFERENTIABLE



TRANSFORMERS:

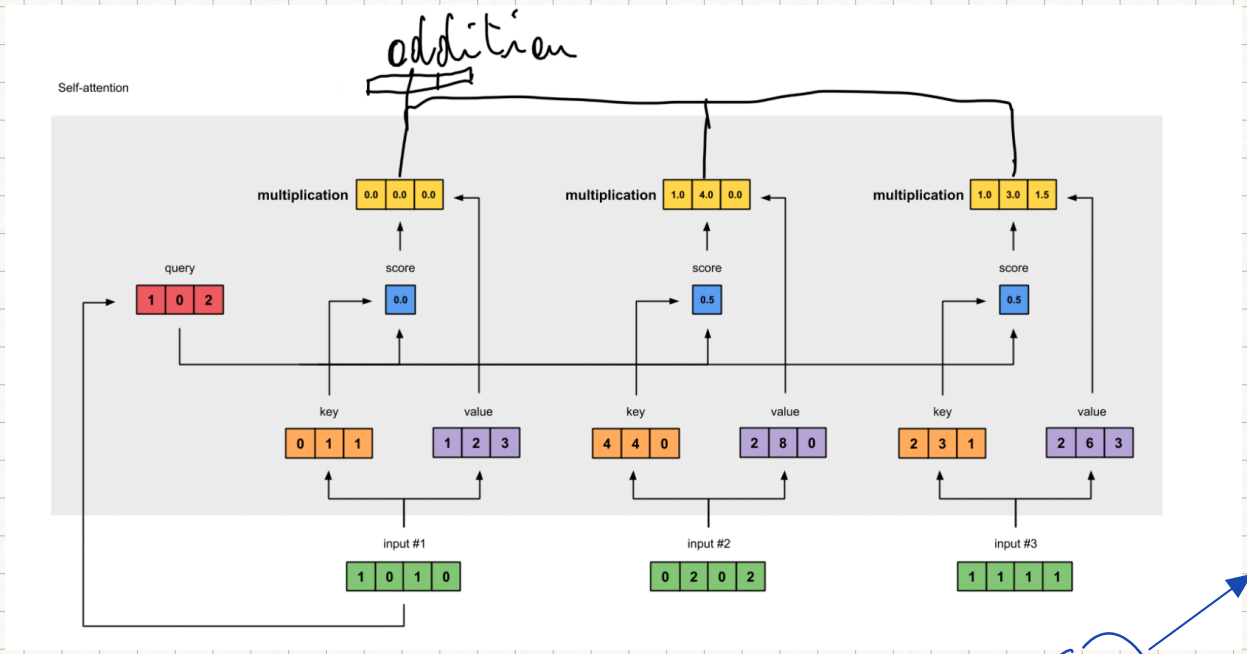
- encoder-decoder architecture
- 6 layer encoder & decoder
- NO RNN, pure attention model + NN



Self-attention: the building blocks are dot-product self-attention. For each attention units, transformer model learns three weight matrices: query weights W_q, key weights W_k and value weights W_v. For each token i, the input word embedding x_i is multiplied with each of the three matrices to produce: Query vector $q_i = x_i W_q$, key vector $k_i = x_i W_k$, value vector $v_i = x_i W_v$

Attention weights a_{ij} from token i to token j is the dot product between q_i and k_j, then they are divided by the $\sqrt{d_k}$ (dimensions of the key vectors), which stabilizes gradients during training, and passes through softmax which normalize the weights. The output of the attention unit for token i is the weighted

sum of the value vectors of all tokens, weighted by a_{ij} (attention from token i to token j, $\forall j$).



$$SA(Q_i, K, V) = \sum_j \text{softmax}_j\left(\frac{Q_i \cdot K^T}{\sqrt{d_K}}\right) \cdot V_j$$

matrix of all key vectors

Q_i → i-th query vector element

j → j-element of softmax vector

the fact that $W_Q \neq W_K$ allows attention to be non symmetric

Multi-head attention: one set of matrices (W_Q, W_K, W_V) is called attention head, we have multiple attention head in each transformer layer. Many transformer attention heads encode relevance relations that are meaningful to humans. The computations for each attention head can be performed in parallel. The outputs for the attention layer are concatenated to pass into the FFNN.

$$\text{MultiHead}(\text{Attention}(Q, K, V) = \text{Concat}(\text{Attention}(QW_i^Q, KW_i^K, VW_i^V))W^O$$

W^O: final projection matrix for the multi-head attention

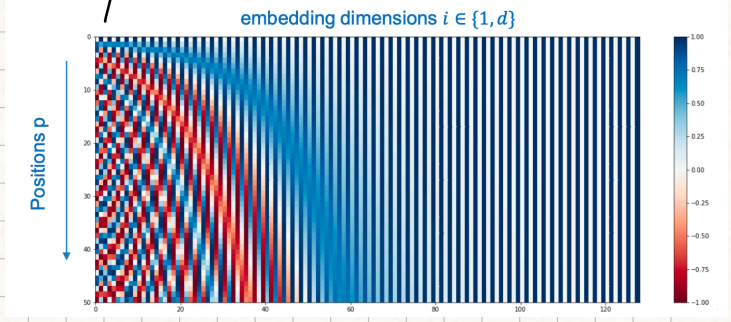
Masked attention: It may be necessary to cut out attention links between some word-pairs. In the decoder we would like to cut connection between token at position t and $t+1$. To use add a mask matrix M before softmax, such that there are negative infinity at entries where the attention link must be cut, and zero at other places.

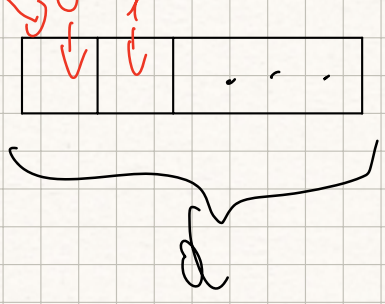
Positional encoding: the positional information is necessary for the transformer to make use of the order of the sequence. A positional encoding is a fixed-size vector representation that encapsulates the relative positions of tokens within a target sequence: it provides the transformer with information about where the words are in the input sequence

$$PE(p, 2i) = \sin(p/10000^{2i/d})$$

$$PE(p, 2i+1) = \cos(p/10000^{2i/d})$$

frequency





$\forall i \in \{0, 1, \dots, d/2 - 1\}$, $d > 0$ integer

dimension of the output word embedding space

PE is a matrix [seq-len, d]

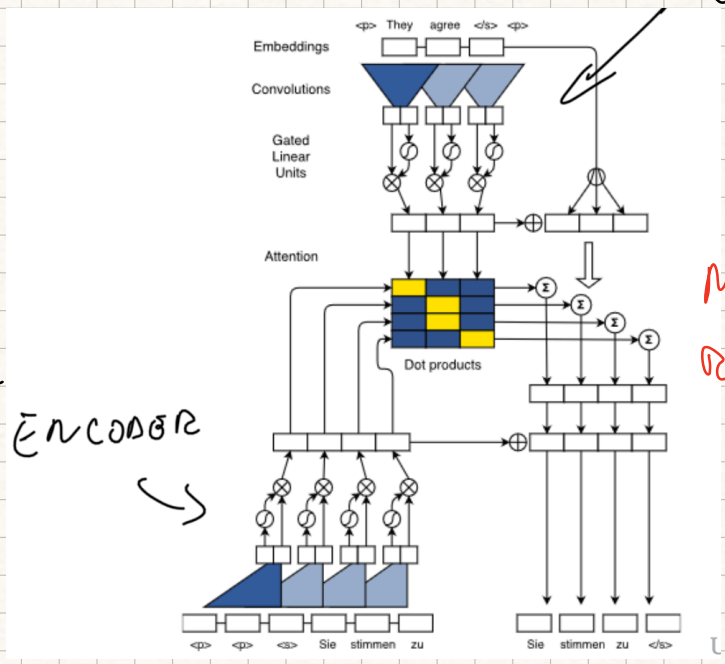
P is the position of the token in the seq.

MEMORY NETWORK 10.12 DL

NEURAL MACHINE TRANSLATION 12.4.5 DL

Convolutional Seq2Seq

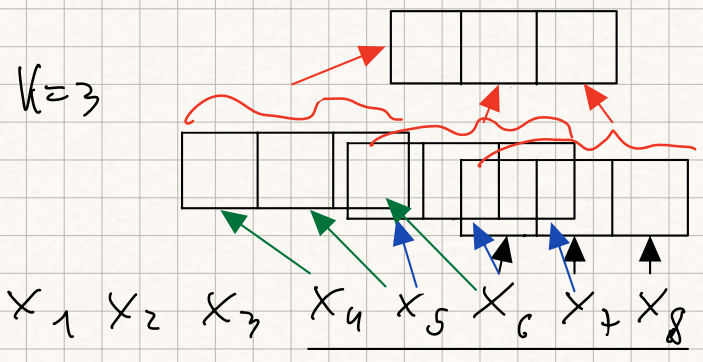
DECODER



NO RECURRENCE

ENCODER

Using convolution means that we can parallelize on GPU. We must choose the dim of the kernel (memory). We can shrink the distance between long-range dependencies stacking multiple layers of convolution.



extend the window from 3 to 6

DEPTH OF DIFFERENT ARCHITECTURES

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

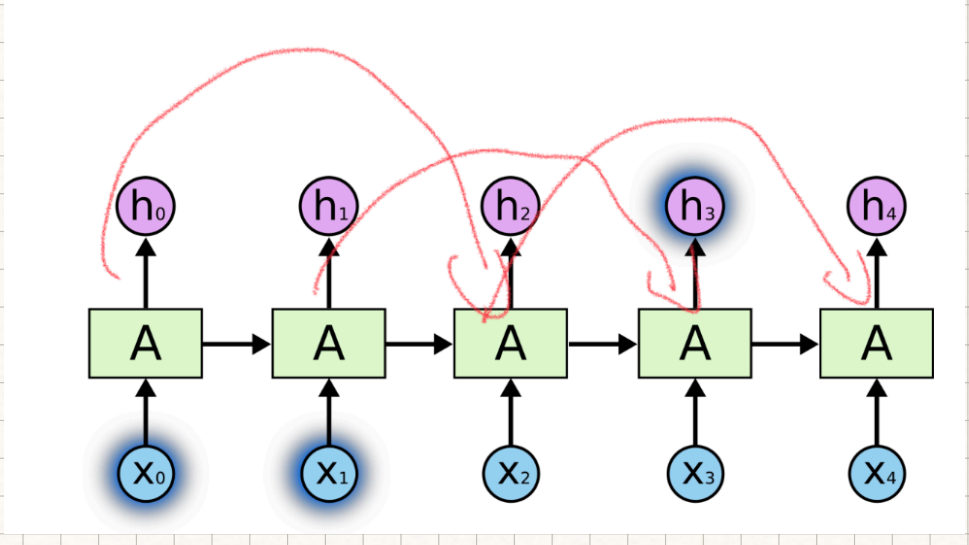
→ backpropagate through n layers

TRANSFORMERS

Vanishing gradients depend on the depth of the network

Hierarchical RNN

Add static skip connections to the model (static skip) or learn when to skip (adaptive skips)

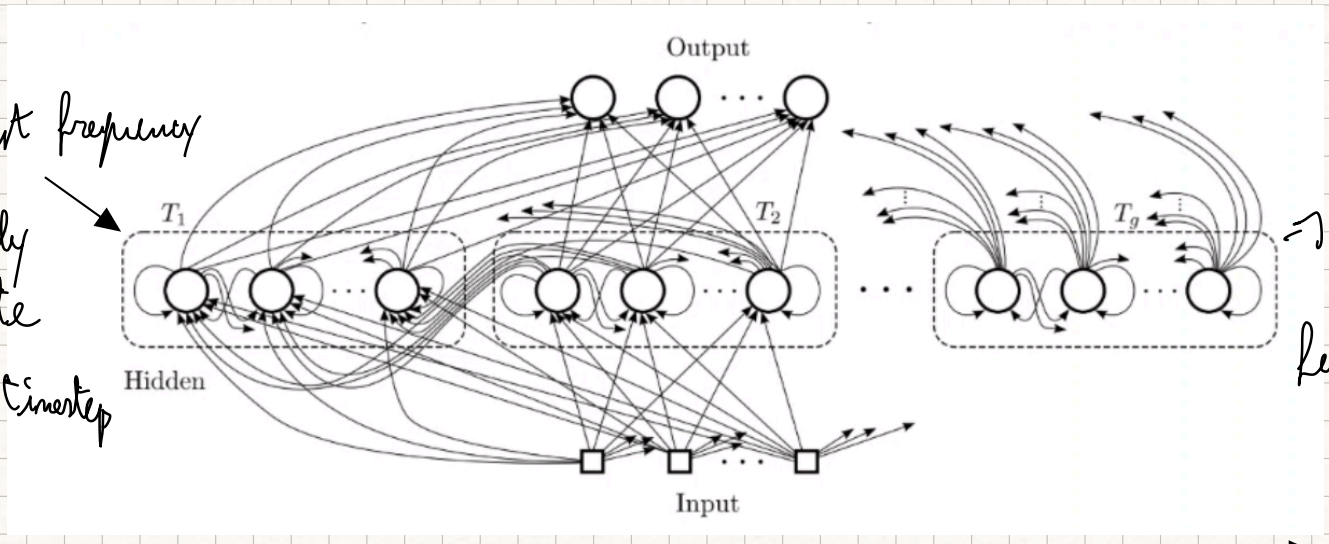


Zoneout - (Regularization): random choice to update a node. At each, given a binary mask (no differentiable), we force some units to keep the same value. Avoiding the update improves gradient propagation.

$$T = d_t \odot \tilde{T} + (1 - d_t) \odot 1$$

↳ normal update

Clockwork RNN:



biggest frequency

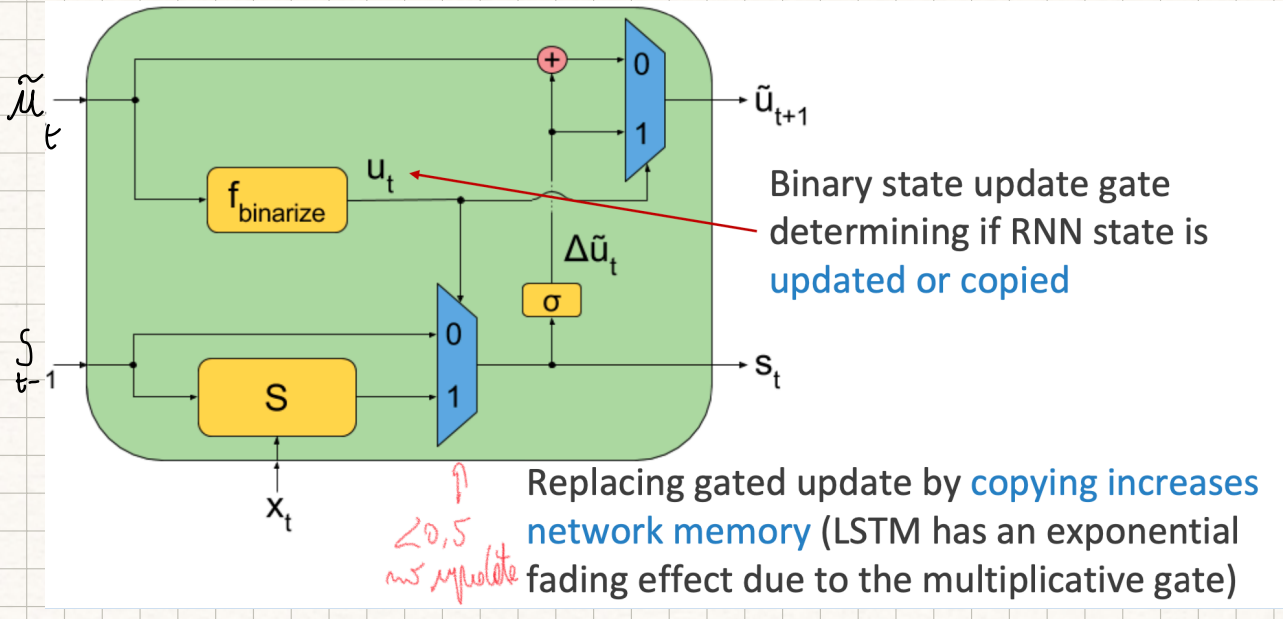
basically update every timestep

smallest frequency, less update

$$T_{g-1} \geq T_g$$

decreasing order of frequency. Each module is updated at its own frequency.

Skip RNN



$< 0,5$ no update

$s_t = f(s_{t-1})$ $\quad s_t = s_{t-1}$ no update

Replacing gated update by copying increases the network memory (LSTM has an exponential fading effect due to the multiplicative gate).

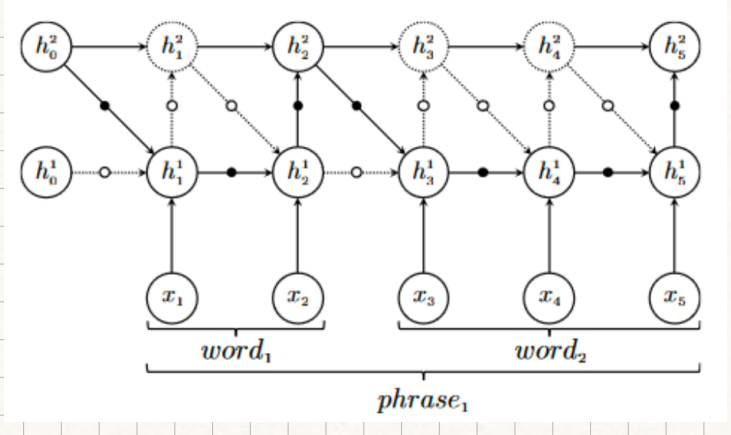
Hierarchical Multiscale RNN (HM-RNN): exploits latent hierarchical structure of sequences. If explicit boundaries are available, we can have different layers for each level of abstraction, combining representations from the lower layer to obtain representation for the higher layer.

OPERATIONS:

UPDATE: state update (LSTM cells) according to boundary detector

COPY: copies cell and hidden states from previous timestep to the current

FLUSH: sends memory to next layer and re-initialize the current layer's state



Recap → Recurrence: update at each timestep, linear scan of the sequence, path length = n

Convolution: update at each timestep but look at the last K timesteps, path length = key x (n)

Attention: update the entire sequence in parallel, path length=1

Zoneout: randomly choose unit update

CW-RNN: blocks of units with different update frequencies, static

Skip-RNN: adaptive gates learns to skip entire update, save comp.

HM-RNN: each layer models more abstract features by learning the boundaries, adaptive

Neural reasoning: Reasoning capabilities are not always straightforward to be accomplished by a simple NN, but NN are good at resolving problems for which we don't have full information (N-GPU without the link cost). We add a memory from which the NN can write and read information.

Memory network: includes a set of memory cells that can be accessed to read and write arbitrary content without explicit supervision about which actions to undertake, and allow end-to-end training without a supervision signal, via the use of a content based soft attention mechanism.

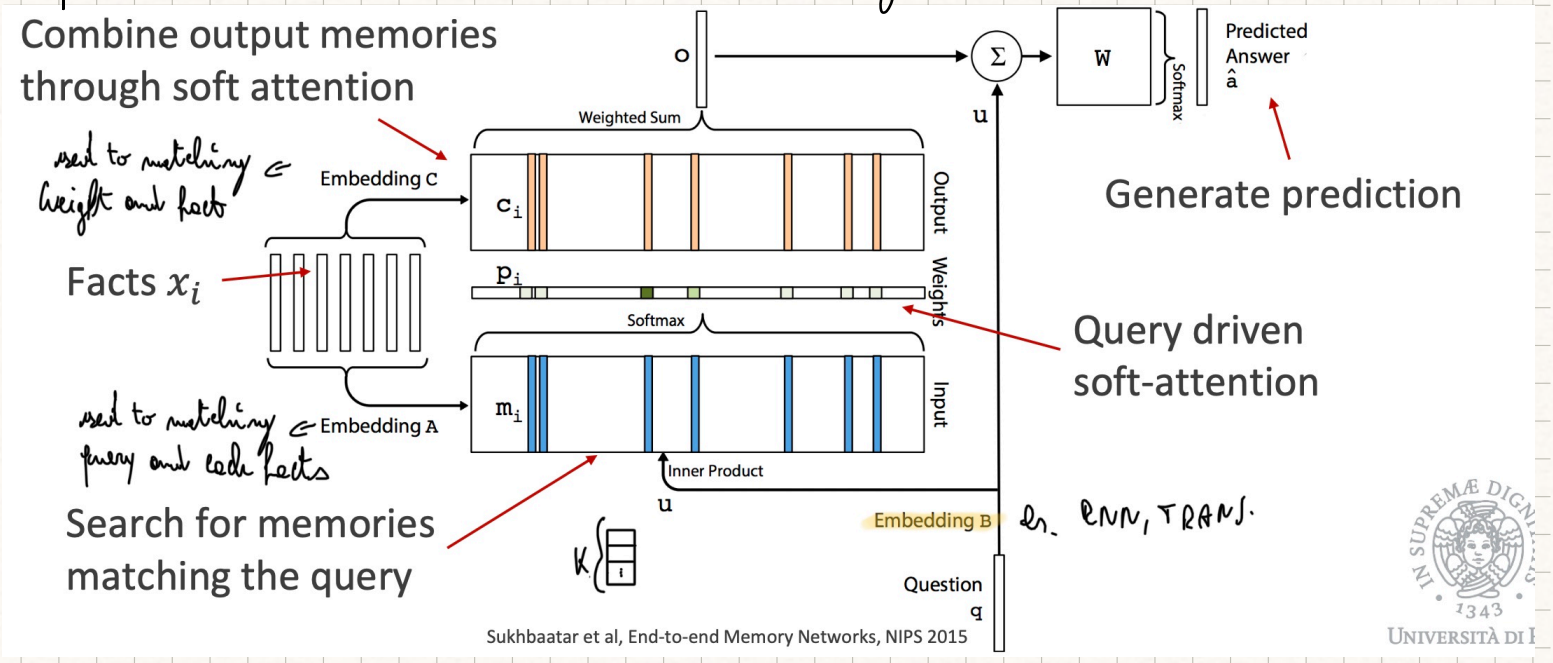
Memory network components:

Input feature map: encodes the input in a feature vector

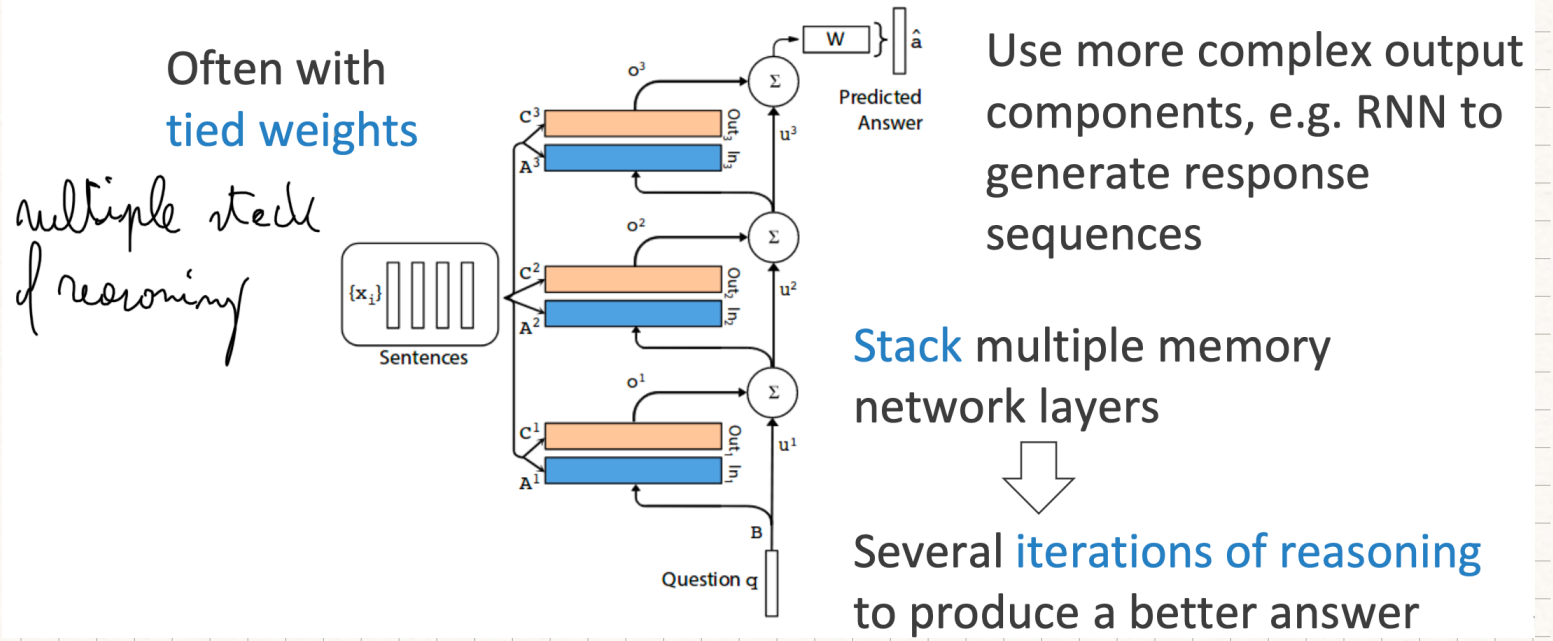
Generalization: decide whether input (or function of it) write into memory

Output feature map: reads the relevant memory slots.

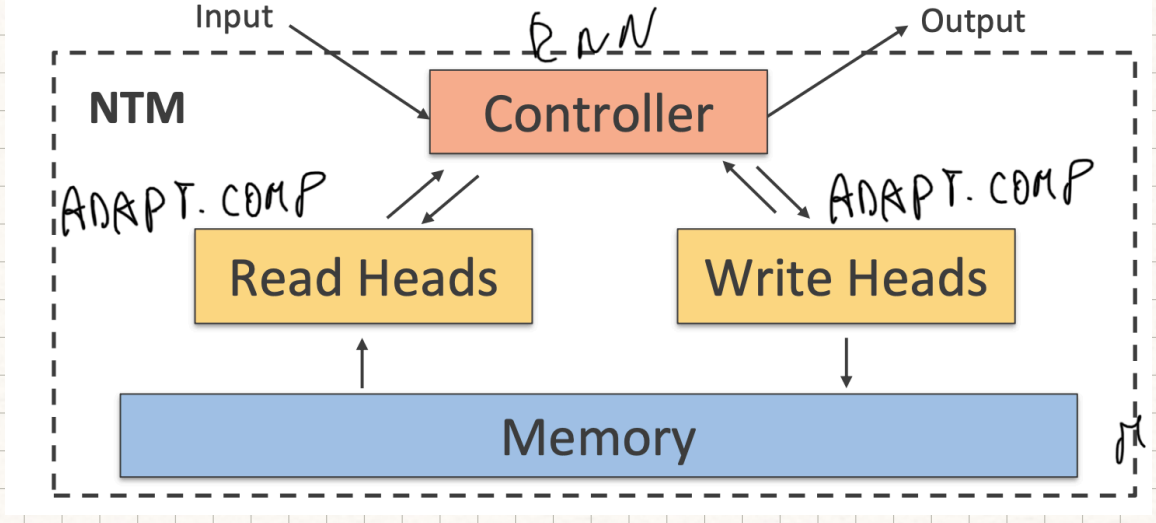
Response: returns the prediction given the retrieved memories



We can stack this model to multiple layer



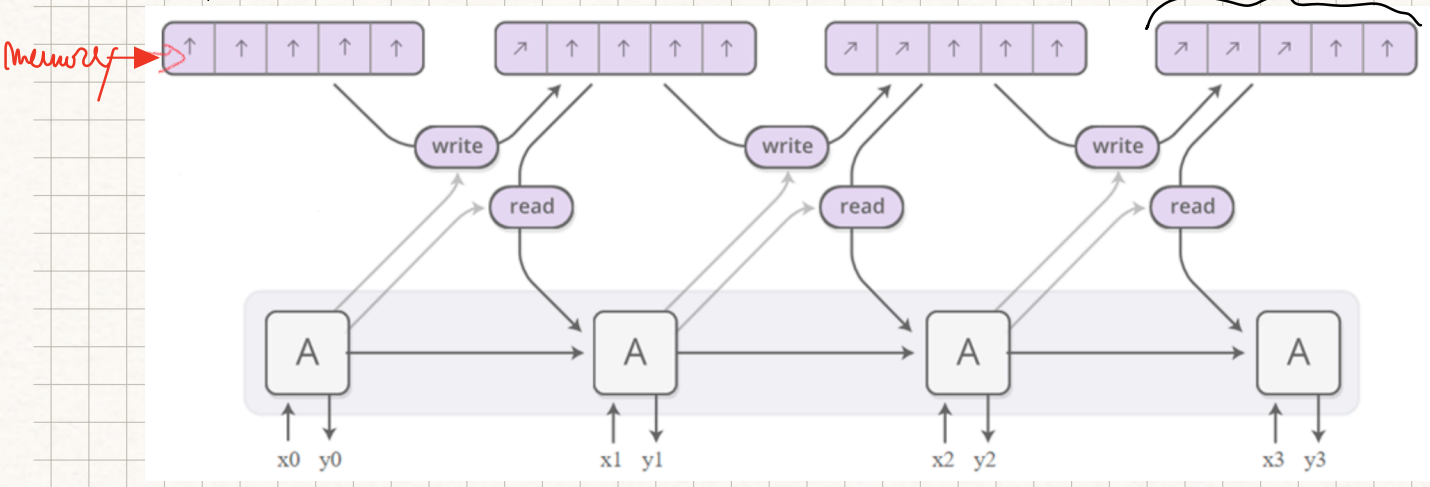
Neural Turing machines: each memory cell can be thought of an extention of the memory cell in LSTM/GRU, and the network outputs an internal state that chooses which cell to read from or write to. NTM reads to or write from many memory cells simultaneously. Each cell is weighted by a value obtained by a softmax function, such that only a small of them are considered relevant when we take a weighted average of the cells. Using these weight with non-zero derivatives allows the function controlling access to the memory to be optimized using gradient descent. the gradient on these coefficients indicates whether each of them should be increased or decreased.



Memory cells contain vector. This allow us to use content-based addressing, where the weight used to read from or write to a cell is a function of that cell. Vector-valued cells allow us to retrieve

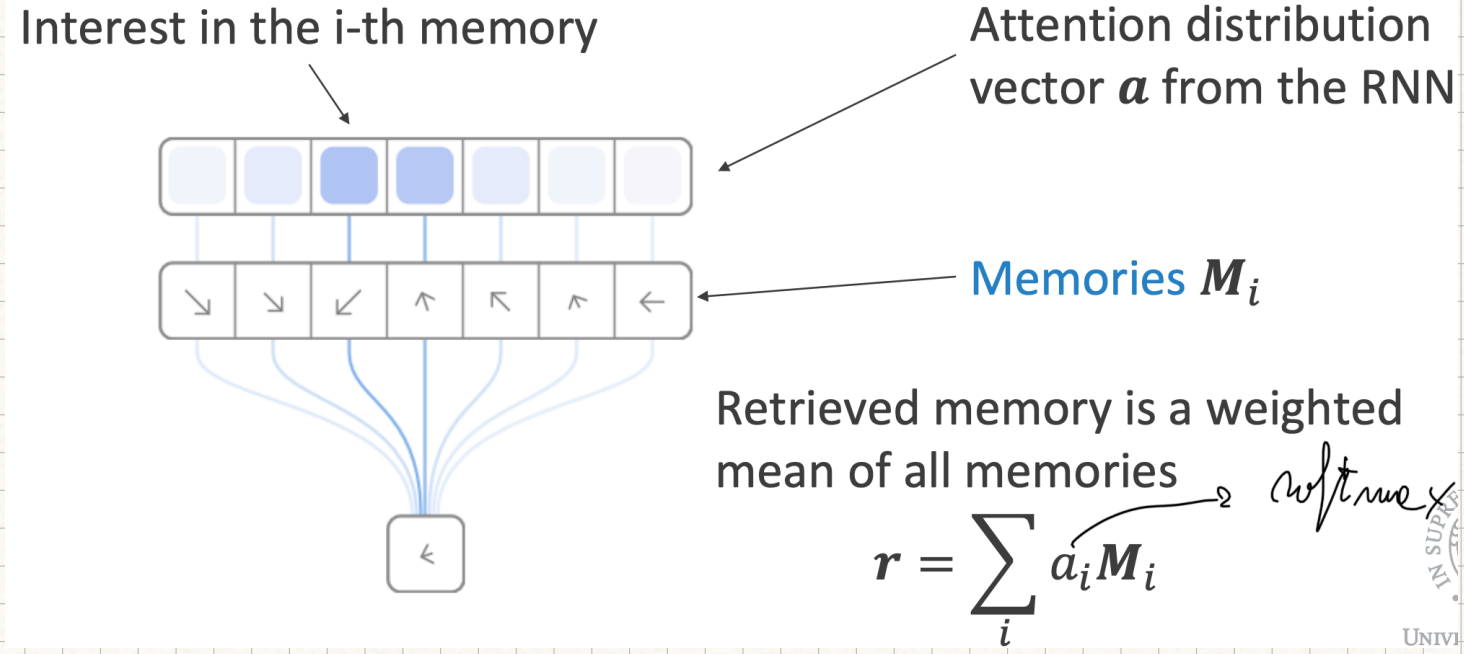
a complete vector-valued memory if we are able to produce a pattern that matches some but not all of its elements.

Neural controller: typically a RNN emitting vectors to control read and write from the memory.

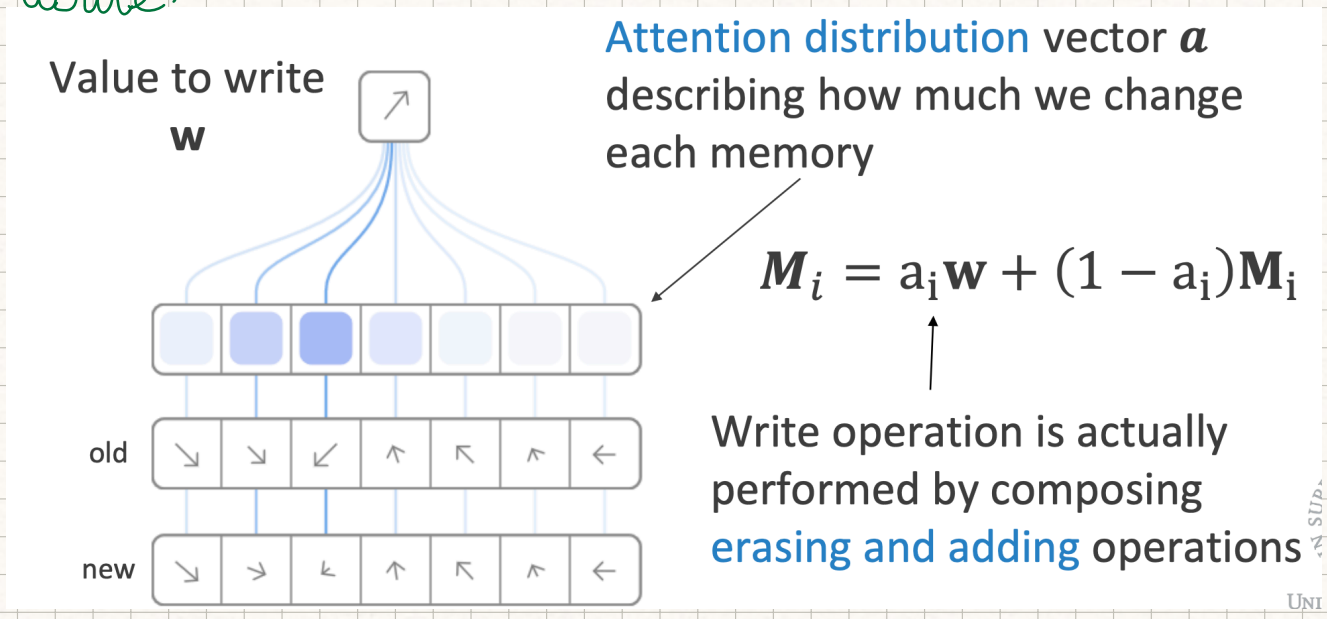


Soft-attention is used to determine how much to read/write from each point.

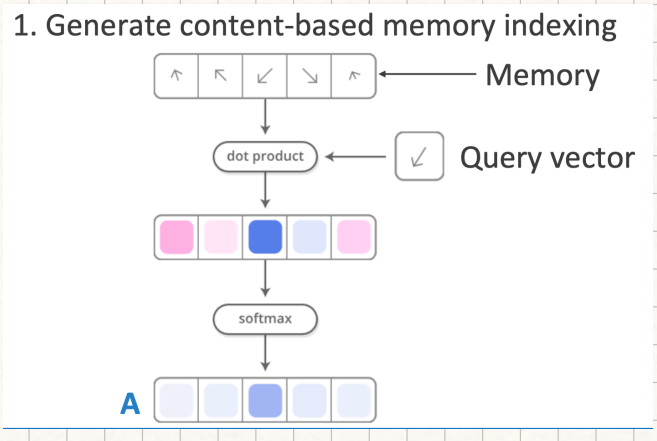
Memory read:

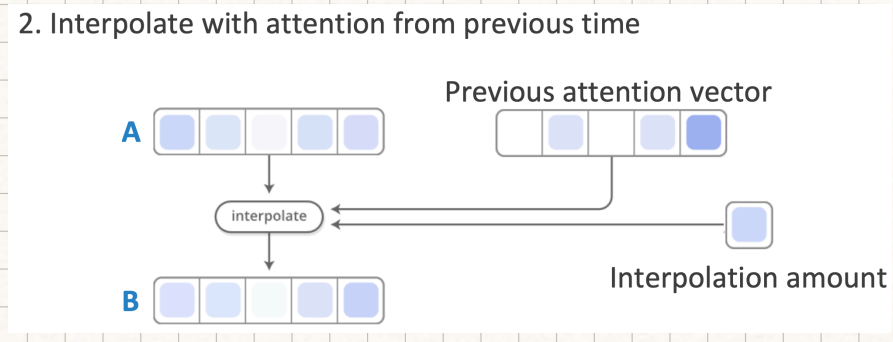


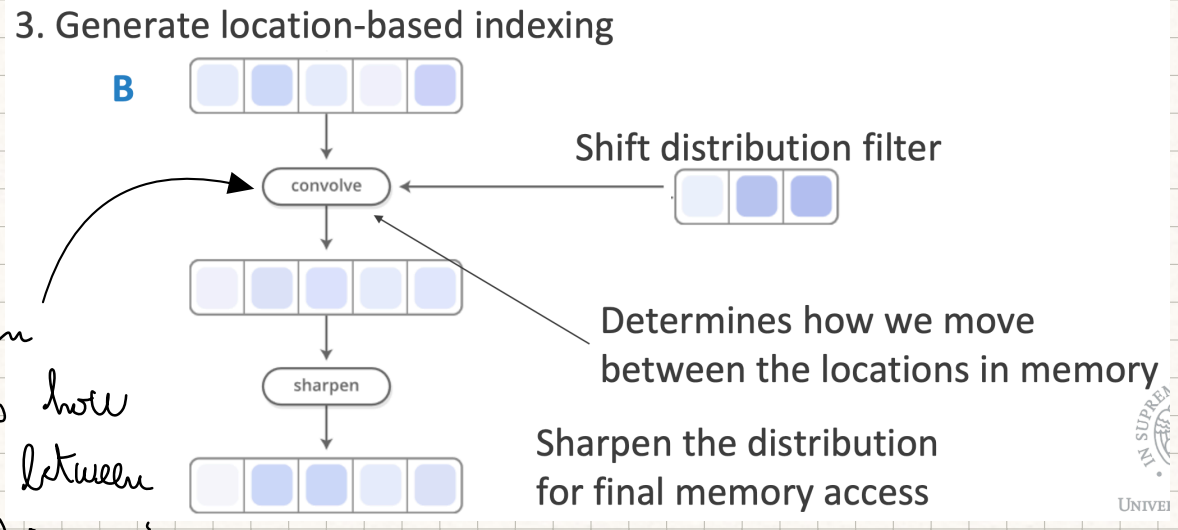
Memory write:



NTM attention focusing







Convolution determines how we move between the locations in memory.

REINFORCEMENT LEARNING

Aim: to maximize the total future reward (R_t a scalar feedback signal), that indicate how good is doing the agent at time t.
All goals can be described by the maximization of expected cumulative reward.

the environment state S_t^e is the environment e private representation at time t, given the history of actions
the agent state S_t^a is the internal representation owned by agent a, given the history of actions

If fully observable ⟶ agent directly observe environment state

$O_t = S_t^a = S_t^e$ Markov decision process

Partial observable ⟶ agent indirectly observes the environment

$S_t^a \neq S_t^e$ Partially observable Markov decision process

A state is markov $\iff P(S_{t+1} | S_1, \dots, S_t) = P(S_{t+1} | S_t)$

$S_t^a = H_t$

$S_t^a = [P(S_t^e = s^1), \dots, P(S_t^e = s^N)]$

$S_t^a = \sigma(W_s S_{t-1}^a + W_o O_t)$

Policy π is the agent behaviour. It's basically a distribution given the state s. Policy are stationary which mean they are time-independent.

deterministic $a = \pi(s)$ uniform distribution.

stochastic $\pi(a|s) = P(A_t = a | S_t = s)$

Value function v is a prediction of future reward. Used to select between actions

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots | S_t = s]$$

Expected cumulative future reward following policy π from state s. γ value between $[0,1]$. weight the reward in the future

Model predict what the environment will do next

$$P(S_{t+1}, R_{t+1} | S_t, A_t) = \underbrace{P(S_{t+1} | S_t, A_t)}_{\text{predict next state}} \cdot \underbrace{P(R_{t+1} | S_t, A_t)}_{\text{predict next reward}}$$

$P^a_{ss'} = P(S_{t+1} = s' | S_t = s, A_t = a)$ next state s' following an action a

$R^a_s = E[R_{t+1} | S_t = s, A_t = a]$ next reward

Markov Decision Process: describe formally an environment for R.L. Environment is fully observable, w the current state completely characterize the process.

Def: a Markov decision process (MDP) is a Markov chain with rewards and actions. It is an environment in which all states are Markov. We have a tuple $\langle S, A, P, R, \gamma \rangle$

- S is a finite set of states
- A is a finite set of actions
- P is a state transition matrix s.t. $P^a_{ss'} = P(S_{t+1} = s' \mid S_t = s, A_t = a)$
- R is a reward function s.t. $R^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
- γ is a discount factor, $\gamma \in [0,1]$

Def: the return G_t is the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \qquad \gamma \in [0,1]$$

the state-value function $v(s)$ of a Markov decision Process is the expected return starting from state s

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

Bellman Equation for MDP

The value function $v(S_t)$ can be decomposed into two parts

1) Immediate reward R_{t+1}

2) Discounted value of successor state $\gamma v(S_{t+1})$

$$v(s) = \mathbb{E}[G_t | S_t = s] = \mathbb{E}\left[\underbrace{\sum_{k=0}^{\infty} \gamma R_{t+k+1}}_{\text{def of } G_t} \,\middle|\, S_t\right] =$$

taking out R_{t+1} $= \mathbb{E}\left[R_{t+1} + \sum_{k=1}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] =$

taking out R_{t+2} $= \mathbb{E}\left[R_{t+1} + \gamma\left(R_{t+2} + \sum_{k=2}^{\infty} \gamma^{k-2} R_{t+k+1}\right) \,\middle|\, S_t = s\right] =$

apply def of G_{t+1} $= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \,\middle|\, S_t = s\right]$

apply def of v_π $= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \,\middle|\, S_t = s\right] =$

$$\underbrace{\mathbb{E}[R_{t+1} | S_t = s]}_{\text{expected value of the immediate reward}} + \gamma \underbrace{\mathbb{E}\left[v_\pi(S_{t+1}) \,\middle|\, S_t = s\right]}_{\text{expected value of the future reward, which can be seen as the sum over all the value functions of all the possible future state } s', \text{ multiplied by the probability of pass from the actual state } s \text{ into that specific } s'}$$

$$= \mathbb{E}[R_{t+1} | S_t = s] + \gamma \mathbb{E}\left[\underbrace{v(S_{t+1})}_{\mathbb{E}[R_{t+2} + \gamma R_{t+3} \dots | S_{t+1}]} \,\middle|\, S_t = s\right]$$

the expected state-value of being in any state reachable from s

$= R_s + \gamma \sum_{s'} P_{ss'} V(s')$

$P_{ss'}$ ↓ state transition probability

Value function with policy: the state-value function $v_\pi(s)$ of an MDP is the expected return starting from state s and following policy π.

$$v_\pi(s) = E_\pi[G_t | S_t = s] = E[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

Action value function: the action-value function $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a]$$

Policy function: $\pi(a|s) = P(A_t = a | S_t = s)$

the state value $v_\pi(s)$ function can be seen as the sum of the possible action values performed on all the possible actions weighted by the policy: $v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] = \sum_{a \in A} \pi(a|s) q_\pi(s, a)$

We can do the same with $q_\pi(s, a)$.

$$q_\pi(s, a) = E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] = R_s^a + \gamma \sum_{s'} P_{ss'}^a v_\pi(s')$$

R_s^a ← immediate reward

expected return of being the future state s' (aka the value function) weighted by the probability of achieve that state s' starting from s and performing action a

Finally we can write:

1) the expected return of being in a state reachable from s through action a and continue following the policy π (we can plug $q_\pi(s,a)$ into $v_\pi(s)$)

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{ss'}^a \, v_\pi(s') \right)$$

2) the expected return of any action a' taken from states reachable from s through action a (and then following policy)

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a \left(\sum_{a' \in A} \pi(a'|s') \, q_\pi(s',a') \right)$$

The optimal policy can be found by maximizing over $q_*(s,a)$, so if we know $q_*(s,a)$, we straightforwardly find the optimal policy:

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in A} q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

Bellman optimality equations: optimal value functions are recursively related Bellman-style

$$v_*(s) = \max_{a \in A} q_*(s,a) = \max_{a \in A} R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s') = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a' \in A} q_*(s',a')$$

Model-based RL

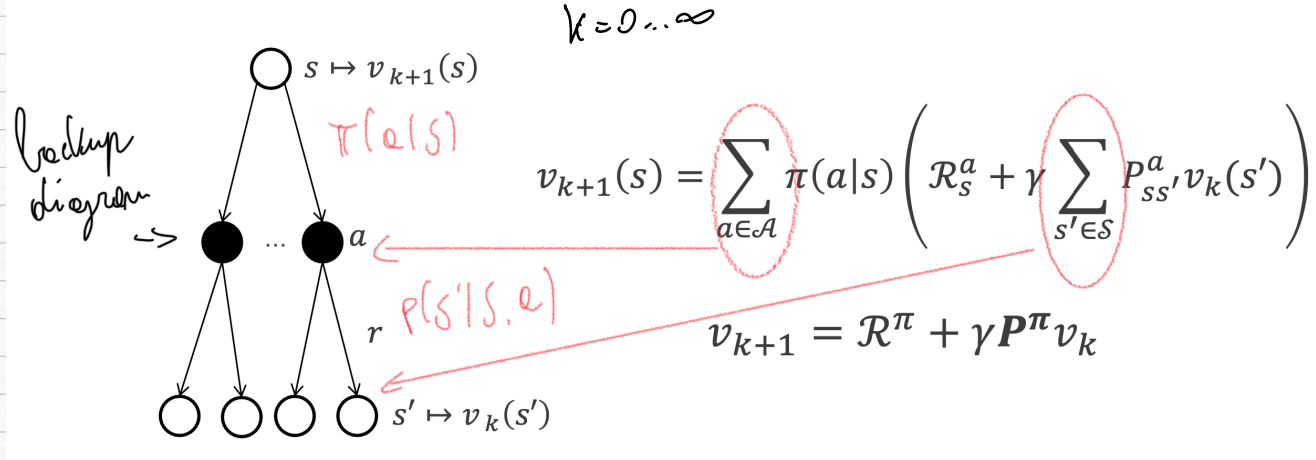
Policy evaluation: evaluate a given policy π, using at each iteration the Bellman expectation. $v_1 \to v_2 \to \dots \to v_\pi$ use v_i to estimate v_2

Synchronous backups

1. at each iteration $k+1$
2. for all states $s \in S$
3. update $v_{k+1}(s)$ from $v_k(s')$ where s' is a successor state of s

Iterative policy evaluation (dynamic programming)

$k = 0 \dots \infty$



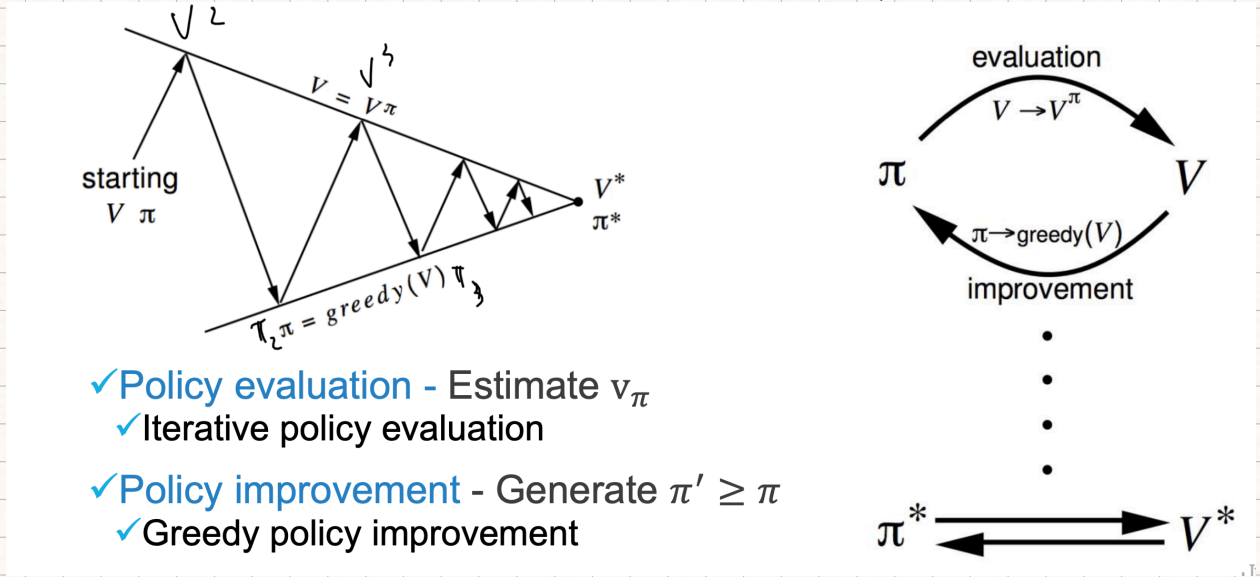
Improve a policy → Given an initial policy π:

1) evaluate the policy $v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \dots \mid S_t = s]$

2) Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(\pi) \Rightarrow \pi'(s) = \underset{a \in A}{\text{argmax}}\ q_\pi(s, a)$$

Policy iteration: given a policy π, we can iterate two steps in order to improve our policy



this iterative policy evaluation is clearly model-based because for every state, I am exploring all the possible actions and all the consecutive states. In order to do that, I need the reward function and the transition function to be able to know with which probability I am going to get in the next state.

When we achieve $q_\pi(s, \pi'(s)) = v_\pi(s)$ we can stop since we have

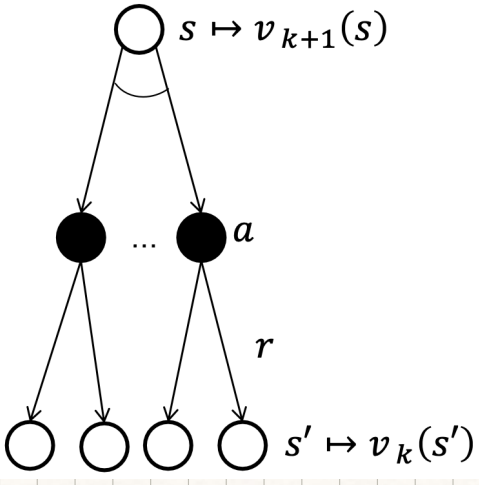
- drive the Bellman optimality, so $v_\pi(s) = v_*(s), \forall s$

Value iteration: find optimal policy π, applying iteratively Bellman optimality $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_\pi$

Synchronous backups

1. at each iteration $k+1$
2. for all states $s \in S$
3. update $v_{k+1}(s)$ from $v_k(s')$

Unlike policy iteration, there is no explicit policy. We couldn't have a starting policy and we could act greedily. In this way we move from an algorithm that evaluates a given policy to an algorithm that finds a policy, because intermidiate value functions may not correspond to any policy



$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

$$v_{k+1} = \max_{a \in \mathcal{A}} (\boldsymbol{\mathcal{R}}^a + \gamma \boldsymbol{P}^a v_k)$$

Model-based Reinforcement Learning

evaluate a policy. learn value function, action value function

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

learn an optimal policy

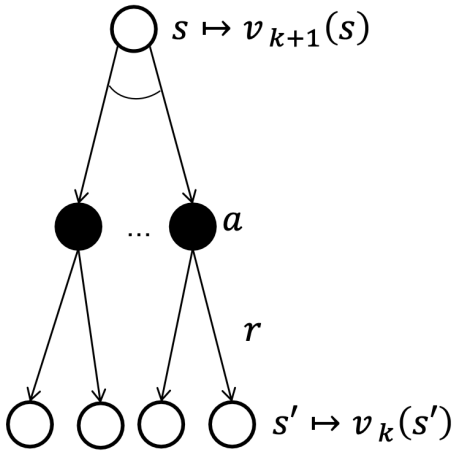
learn a policy, given we one

given one policy

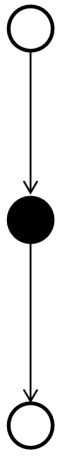
- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
  - Complexity is $O(mn^2)$ per iteration ($m = |\mathcal{A}|$ and $n = |\mathcal{S}|$)
- Could also apply to action-value function $q_\pi(s,a)$ or $q_*(s,a)$
  - Complexity is $O(m^2n^2)$) per iteration

MODEL FREE

Sample backup

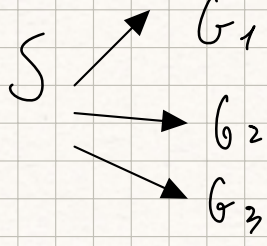


- From now onwards we consider sample backups
  - Using sample rewards and sample transitions $\langle S, A, R, S' \rangle$
  - Instead of reward function $\mathcal{R}$ and transition function P
- Pros
  - Model-free - no advance knowledge of MDP required
  - Breaks the curse of dimensionality through sampling
  - Cost of backup is constant, independent of $n = |\mathcal{S}|$



Monte-Carlo Reinforcement learning

MC learns from episodes of experience, it has no knowledge of MDP transitions / rewards. MC learns from complete episodes (no bootstrapping) Value corresponds to the mean return across different episodes



$V(S) = AVG(G_i)$

MC policy evaluation: learn v_π from episodes of experience under policy π $S_1, A_1, R_2, .. R_k \sim \pi$.

MC policy evaluation uses empirical mean return instead of expected return.

MC updates $V(S)$ incrementally after episode $S_1, A_1, R_2, ... R_T$

For each state S_t with return G_t:

1) Increment counter $N(S) \leftarrow N(S) + 1$

2) Update value function (with incremental mean)

$S_t = S$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)} (G_t - V(S_t))$$

G_t: ground truth

In non-stationary problems track a running mean (forget old episodes)

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$

α: "learning rate"

Temporal-Difference (TD) learning

TD is model-free and learns from incomplete episodes of experience, by bootstrapping. TD updates a guess towards a guess.

TD(0) updates value $V(S_t)$ toward estimated return $R_t + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha \left(\underbrace{R_t + \gamma V(S_{t+1})}_{\text{TD-target}} - V(S_t) \right)$$

$V(S_t)$ → old value

$\underbrace{\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}_{\text{TD-error}}$

Bias-Variance Tradeoff

Return $G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$ is an unbiased estimator of $v_\pi(S_t)$.

True TD target $(R_{t+1} + \gamma v_\pi(S_{t+1}))$ is an unbiased estimator of $v_\pi(S_t)$.

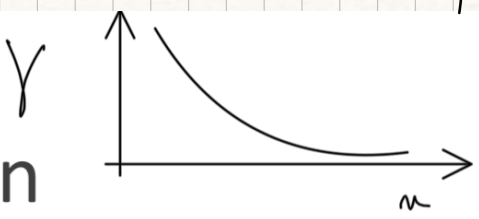
TD target $R_{t+1} + \gamma V(S_{t+1})$ is biased estimate of $v_\pi(S_t)$

For MC you learns after having taken different actions in a single episode (high variance) → (low bias)

MC has good convergence properties, (even with function approx.) is simple inefficient is not very sensible to initial value.

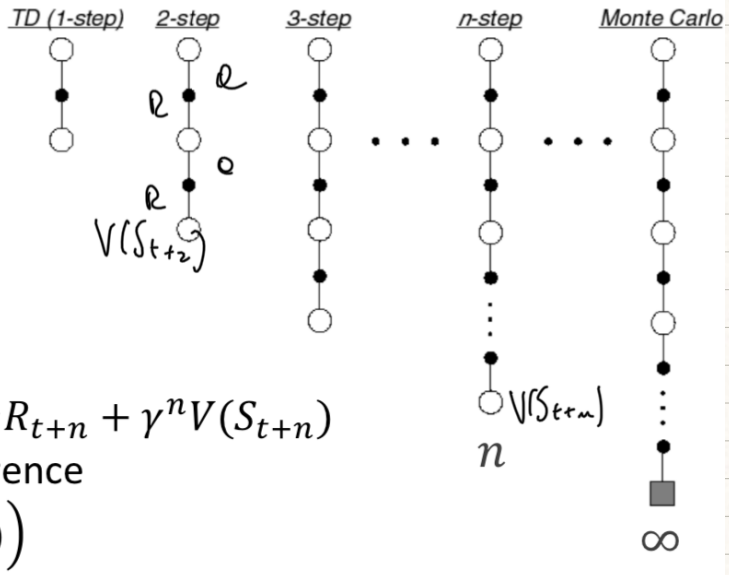
TD target looks from a single action (low variance)
↓
(high bias)

Is more simple efficient than MC. TD(0) converges to $v_\pi(s)$ (but not always with function approximation), and is more sensitive to initial value (due to bootstrapping → high bias)



n-step Prediction

Have TD look and target n steps in the future



✓ Define the n-step return

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

✓ Learn based on the n-step difference

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{(n)} - V(S_t)\right)$$

λ-returns (Forward View)

$\lambda \in [0,1]$

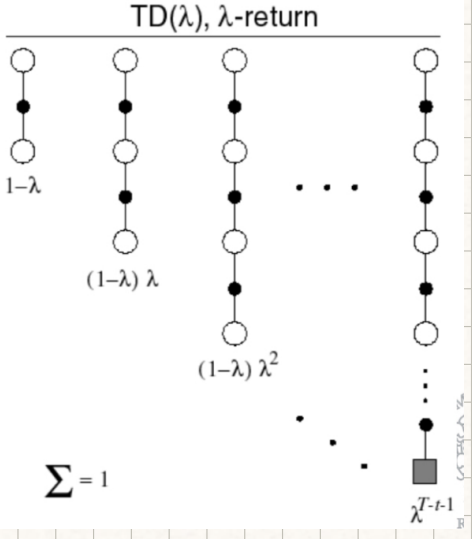
- The λ-return G_t^λ combines all n-step returns $G_t^{(n)}$
- Using weight $(1-\lambda)\lambda^{n-1}$

we don't have access to in practice →

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

- Update as appropriate (TD(λ))

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^\lambda - V(S_t)\right)$$



Forward view doesn't give mechanism. Backward view does it. Update online, every step, from incomplete sequences. It use eligibility traces for solving the credit assignment problem by trading between frequency heuristic (assign credit to most frequent states) and recency heuristic (assign credit to most recent states). λ is the parameter that add the eligibility trace in the algorithm.

Backward view TD(λ): Keep an eligibility trace for every state s and update value $V(s)$ for every state s.

$$\text{TD error } \delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

$$V(s) = V(s) + \alpha \delta_t \underbrace{E_t(s)}$$

$$E_t(s) = \gamma \lambda E_{t-1}(s) + 1(S_t = s)$$

Model-free RL Control

greedy policy improvement over $Q(s,a)$ is model-free

$$\pi'(s) = \arg\max_{a \in A} Q(s,a)$$

ϵ-greedy exploration

we add some randomness to ensure continual state exploration

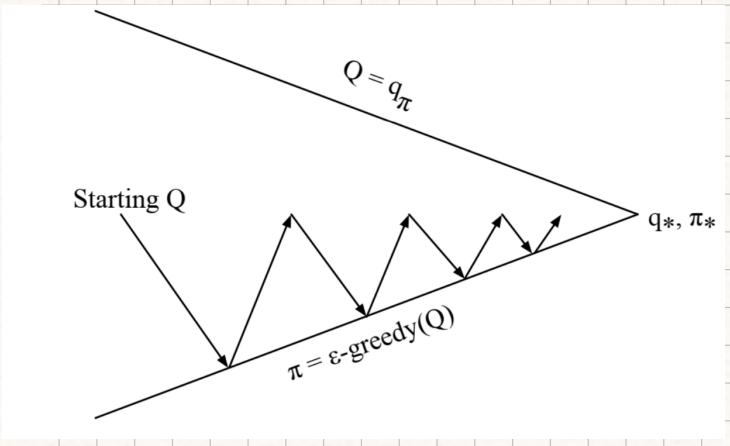
All m actions are tried with non-zero probability

$1-\epsilon$ probability of choosing the greedy action

ϵ probability of choosing an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + (1-\epsilon) & \text{if } a^* = \arg\max_{a \in A} Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

On-Policy control with SARSA

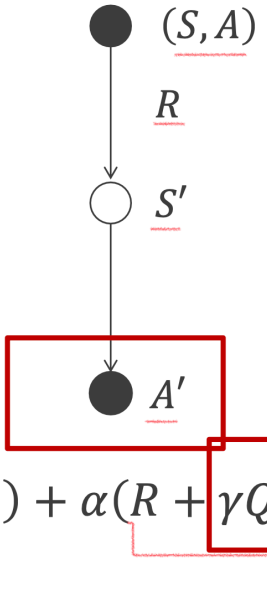


every time-step:

- do policy evaluation - TD, $Q \approx q_\pi$
- do policy improvement - ϵ-greedy policy

TD learning: take one action, arrive at the final state → generate the error $Q =$ 

Updating Action-Value Functions with SARSA



Expected SARSA

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') q_*(s',a')$$

We sample also future action A' (instead of leveraging policy to compute expectation)

old-value

Q function

$$Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$$

error

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

SARSA Algorithm for On-Policy Control

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:

  Initialize S

  Choose A from S using policy derived from Q (e.g., ε-greedy)

  Loop for each step of episode:

    Take action A, observe R, S'

    Choose A' from S' using policy derived from Q (e.g., ε-greedy)

    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$

    $S \leftarrow S'$; $A \leftarrow A'$;

  until S is terminal

SARSA(λ) Algorithm

Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Repeat (for each episode):

  $E(s, a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$ eligibility trace

  Initialize S, A

  Repeat (for each step of episode):

    Take action A, observe R, S'

    Choose A' from S' using policy derived from Q (e.g., ε-greedy)

    $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$

    $E(S, A) \leftarrow E(S, A) + 1$

    For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:

      $Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$

      $E(s, a) \leftarrow \gamma \lambda E(s, a)$ eligibility trace

    $S \leftarrow S'$; $A \leftarrow A'$ → decide how much the eligibility trace forget

  until S is terminal

OFF-Policy learning: an off-policy learning algorithm is able to learn $V_\pi(s)$ or $q_\pi(s, a)$ of a target policy $\pi(a|s)$ using the so called behaviour policy $\mu(a|s)$ from where we sample states, actions and rewards.

Off-policy learning of action-value $Q(s, a)$: Q-learning

- next action is chosen using behaviour policy $A_{t+1} \sim \mu(\cdot|S_t)$
- but we consider alternative successor action $A' \sim \pi(\cdot|S_t)$
- and update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)\right)$$

Off-policy control by Q-learning: allow both behaviour and target policies to improve.

The target policy π is greedy w.r.t. $Q(S_t, A_t)$

$$\pi(S_{t+1}) = \arg\max_{a'} Q(S_{t+1}, a')$$

the behaviour policy μ is ϵ-greedy w.r.t. $Q(S, a)$

the Q-learning target then simplifies to

$$R_{t+1} + \gamma Q(S_{t+1}, A') = R_{t+1} + \gamma Q\left(S_{t+1}, \arg\max_{a'} Q(S_{t+1}, a')\right)$$

$$= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')$$

Th: Q-learning control converges to the optimal action-value function, $Q(S, a) \rightarrow q_*(S, a)$

$$Q(S, A) \leftarrow Q(S, A) + \alpha\left(R + \max_{a'} \gamma Q(S', a') - Q(S, A)\right)$$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$

Repeat (for each episode):

  Initialize S

  Repeat (for each step of episode):

    Choose A from S using policy derived from Q (e.g., ε-greedy)

    Take action A, observe R, S'

    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

    $S \leftarrow S'$;

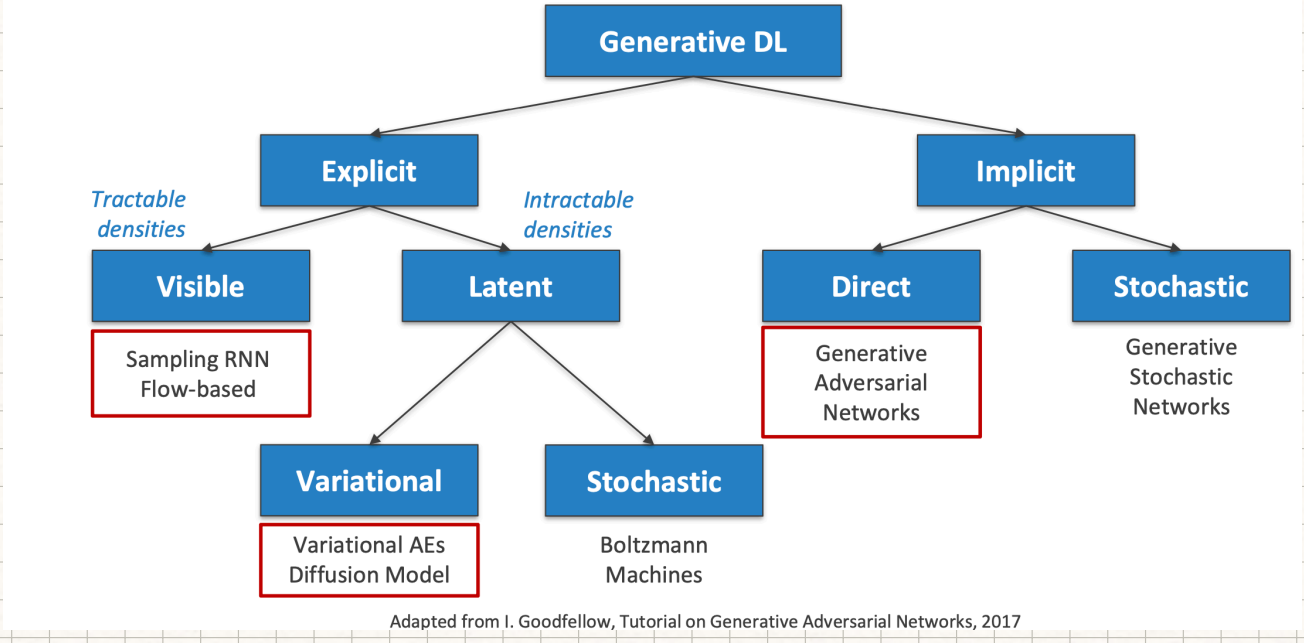
  until S is terminal

DA FINIRE

Generative learning:

Explicit: it learns a model density $P_\theta(x)$. If it's tractable, we can use all the visible data in a regular RNN model. Instead, if the density is not tractable, we can introduce a latent space in a variational way (VAE)

Implicit: It learns a process that sample data from $P_\theta(x) \simeq P(x)$. It can be done in a direct way (GAN) or stochastically



Adapted from I. Goodfellow, Tutorial on Generative Adversarial Networks, 2017

If all information is fully visible the joint distribution can be computed from the chain rule factorization.

Bayesian network → $P(x) = \prod_i^N P(x_i | x_1, \dots, x_{i-1})$

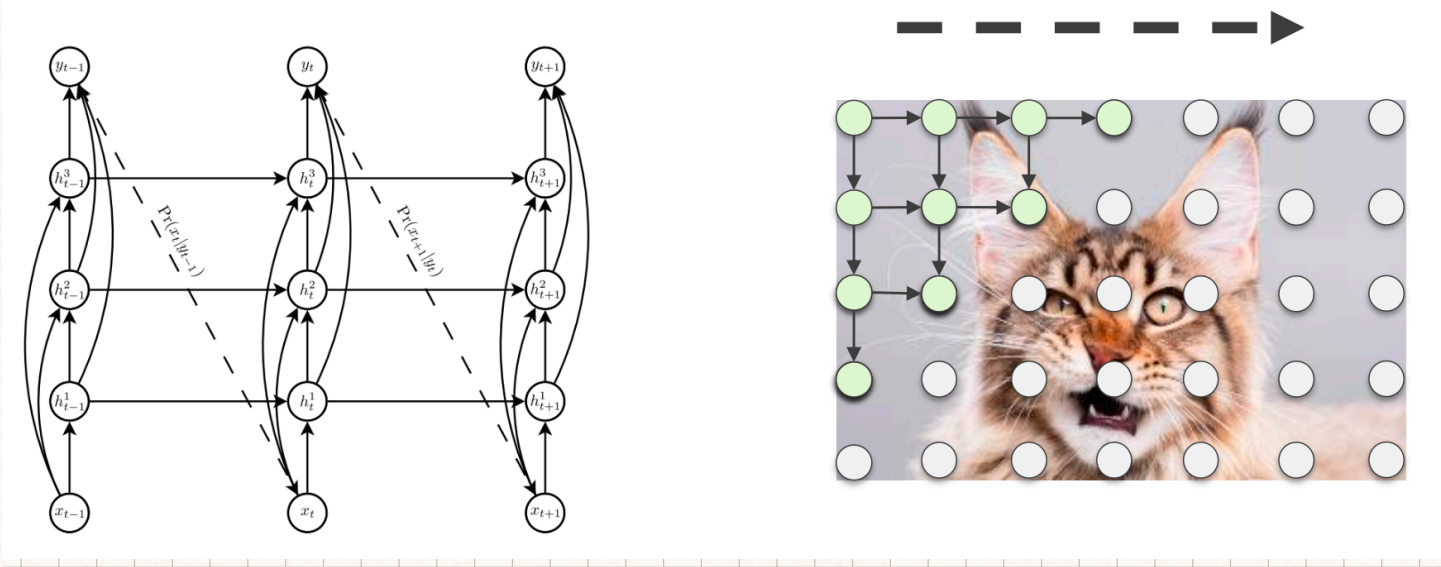
If x is an image, then x_i is the i-th pixel of an image.

$P(x_i | x_1, \dots, x_{i-1})$ is the prob. of a pixel having a certain intensity value, given the known intensity of its predecessor.

Two problems: 1) need to be able to define an ordering for the chain rule

2) conditional distribution is difficult to compute

We can scan the image according to a schedule and encode the dependency from previous pixels in the states of a RNN.



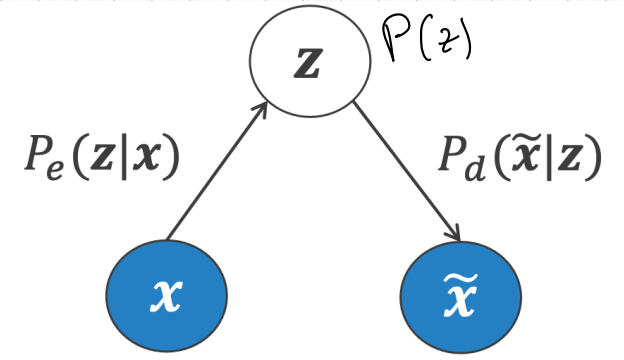
Generating pixels from an image (even small 64×64) can be difficult due to the gradient vanishing prob. (we have a long term dependencies from the first and last pixel).

We can generalize the parameter model $P_\theta(x) = \prod_i^n P_\theta(x_i | x_1, \dots, x_{i-1})$ in a latent space regulated by unobserved variable z.

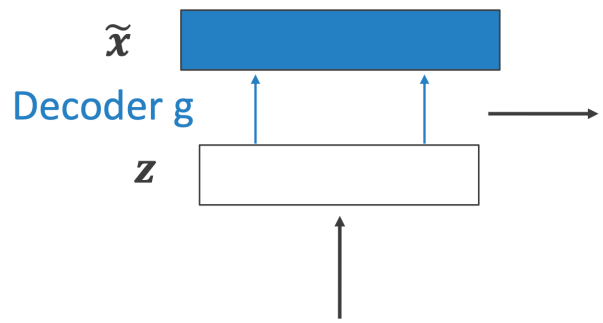
$$P_\theta(x) = \int P_\theta(x|z) P_\theta(z) \, dz$$

The computation of the integral is not trivial.

Variational Autoencoders: it's a NN that, in terms of probabilistic terms, generates $\tilde{x}$ sampling z during the decoder phase, hence, as output we have value sampled from $P(\tilde{x}|z)$. But we don't have access to the true distributions $P(z)$ and $P(\tilde{x}|z)$



The main idea behind VAE used to estimate the parameter θ of $P_\theta(x)$ is:



Represent the $P(\tilde{x}|z)$ distribution through a neural network g (remember the denoising autoencoder)

Sample **z** from a simple distribution such as a Gaussian

$z \sim \mathcal{N}(\mu(x), \sigma(x))$

At training time sample z conditioned on data **x** and train the decoder g to reconstruct **x** itself from **z**

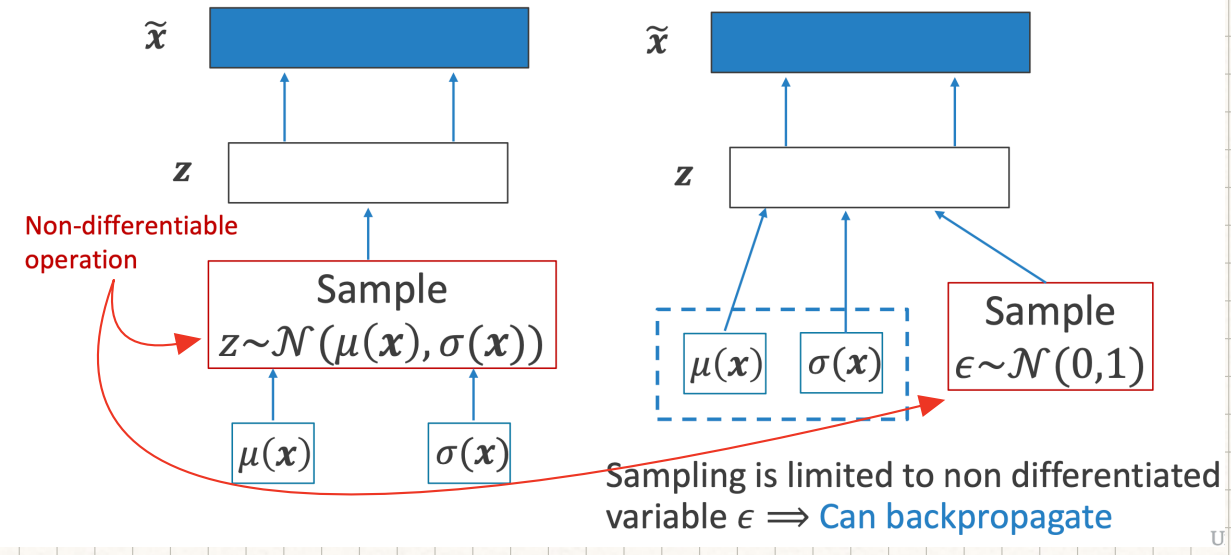
VAE training is NOT easy because one would like to train maximizing

$$L(D) = \prod_i^N P(x_i) = \prod_i^N \int P(x_i|z)\, P(z)\, dz$$

intractable — non differentiable

solved by: variational approximation — reparametrization

Reparameterization trick:



we can rescale white noise with μ and σ

Variational approximation: in practice we maximize the ELBO $\rightarrow$ $\log P(x|\theta) \geq E_q[\log P(x,z)] - E_q[\log q(z)] = \mathcal{L}(x,\theta,\phi)$

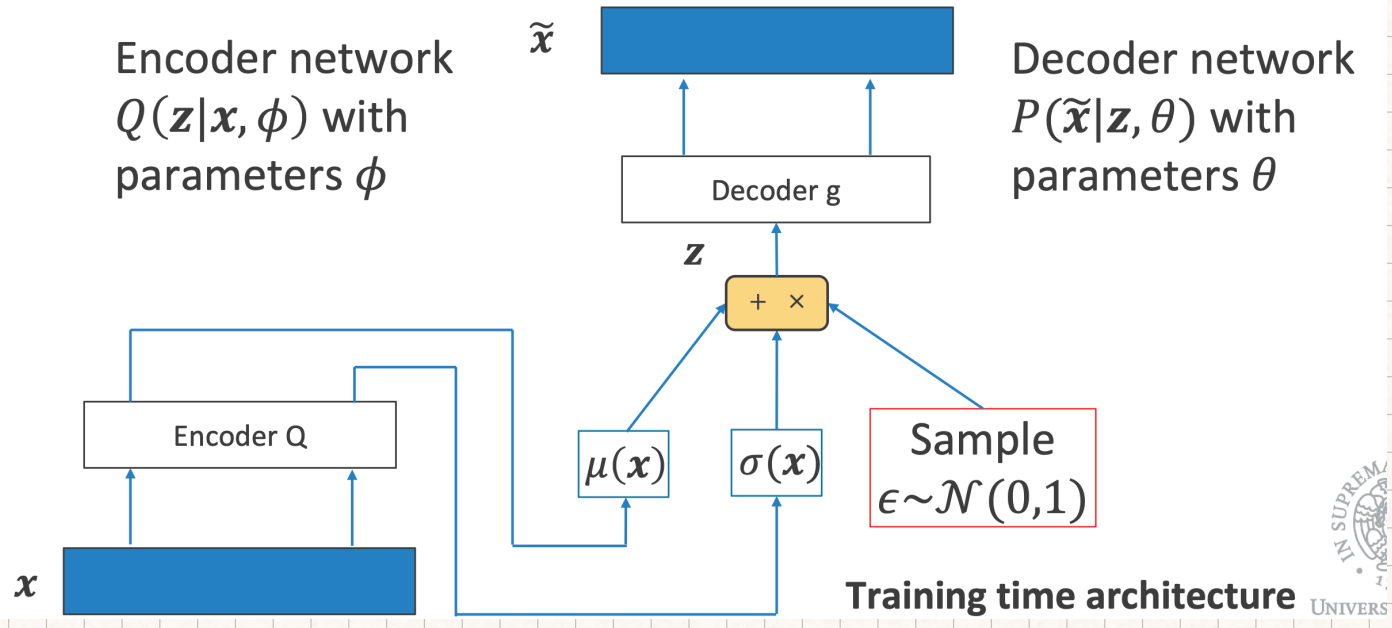
maximizing the ELBO allows approximating from below the intractable log-likelihood $\log P(x)$

$$\mathcal{L}(x,\theta,\phi) = E_q[\log P(x|z)] + E_q[\log P(z)] - E_q[\log q(z)]$$

decoder estimate of the conditional, made possible and differentiable through the reparametrization trick

$KL(Q(z|\phi)\,||\,P(z|\theta))$ need a $Q(z)$ function to approximate $P(z)$

Variational Autoencoder – The Full Picture



VAE Training: training is performed by backpropagation on θ and ϕ in order to maximize the ELBO

$$\mathcal{L}(x,\theta,\phi) = E_{\epsilon}\left[\log P(x|z=\mu(x)+\sigma(x)*\epsilon,\theta)\right] - KL\left(Q(z|x,\phi)\,||\,P(z|\theta)\right)$$

encoder

reconstruction (decoder)

regularization: force z to be more simple as possible and it can be computed easily when both $Q(z)$ and $P(z)$ are Gaussian

Can be computed in closed form when both $Q(z)$ and $P(z)$ are Gaussians

$$KL\left(\mathcal{N}(\mu(x),\sigma(x))\,||\,\mathcal{N}(0,1)\right)$$

VAE final Loss:

$$E_{x\sim D}\left[E_{\epsilon\sim N(0,1)}\left[\log P(x|z=\mu(x)+\sigma^{1/2}(x)*\epsilon,\theta)\right]\right.$$
$$\left.-KL\left(Q_\phi(z|x,\phi)\,||\,P(z)\right)\right]$$

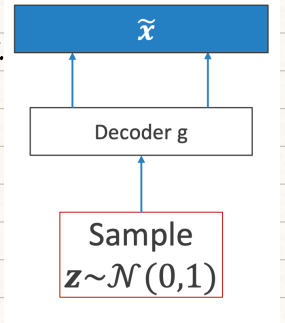
After the reparametrization trick the expectations don't depend on distributions of model parameters so we can perform SGD

The first part of the equation indicates the number of bits required to reconstruct x from z under the ideal encoding (i.e. $Q_\phi(z|x)$ is generally suboptimal)

$$E_{z\sim Q}\left[\log P(x|z)\right]$$

The second part is the number of bits required to convert an uninformative sample from $P(z)$ into a sample from $Q_\phi(z|x)$. The information gain is the amount of extra information that we get about x when z comes from $Q_\phi(z|x)$ instead of from $P(z)$

VAE testing:

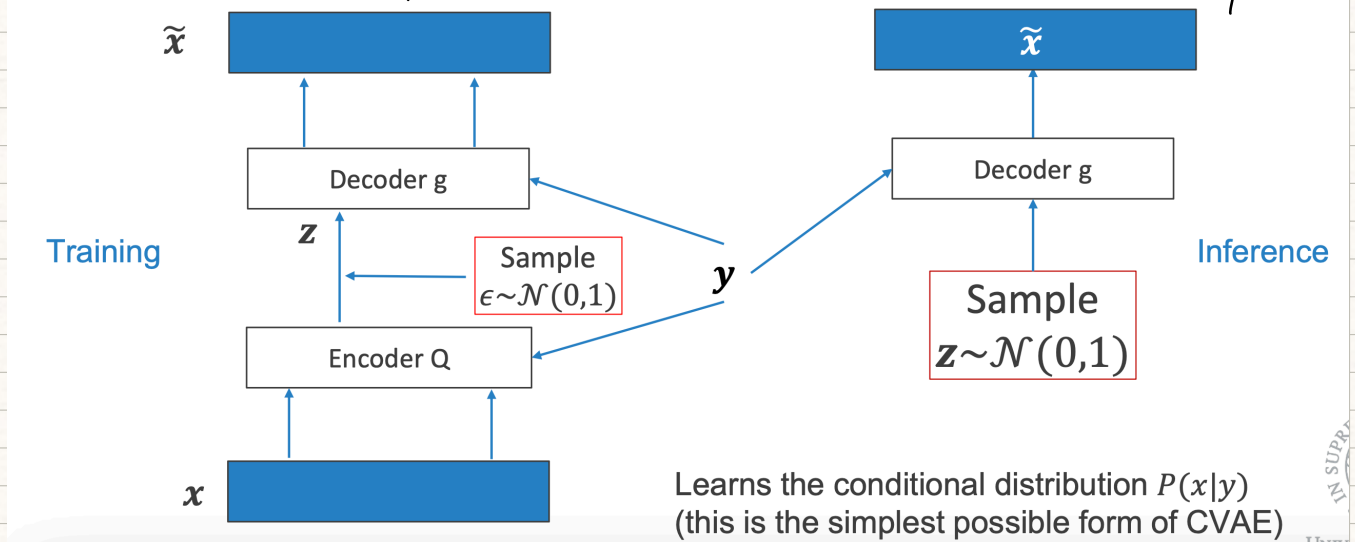


At test time detach the encoder, sample a random encoding and generate the sample or the corresponding reconstruction

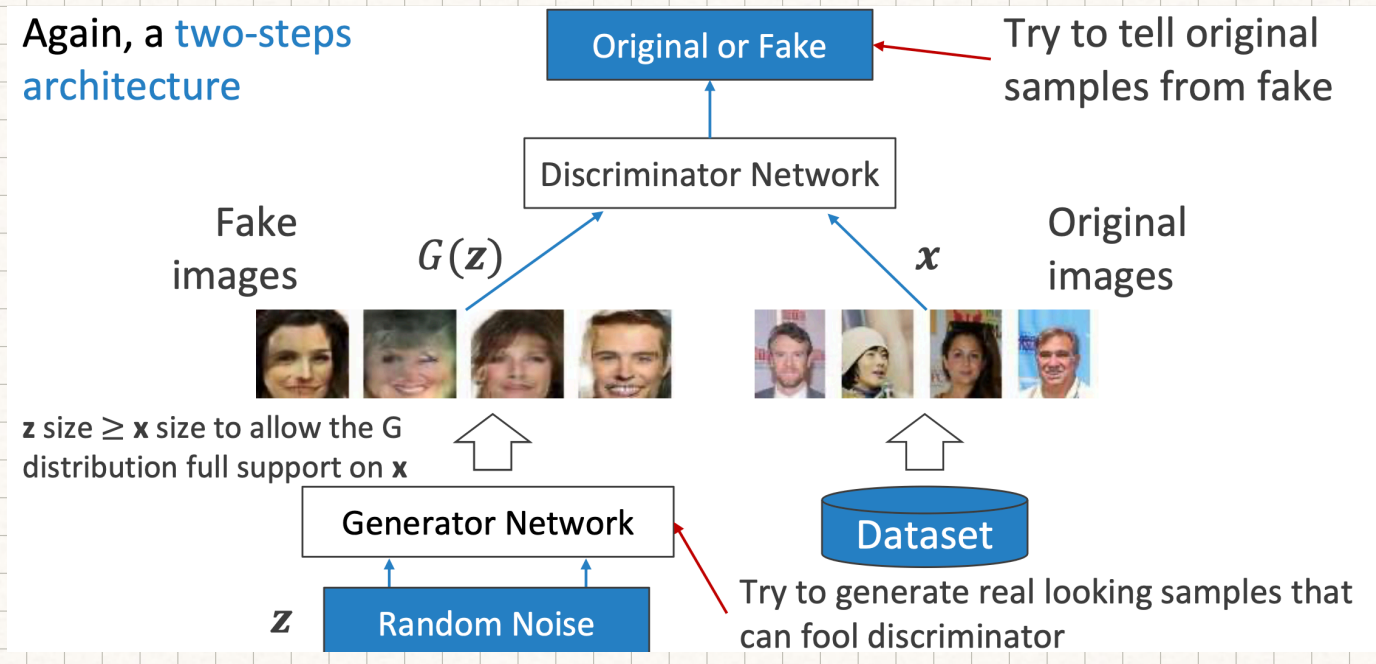
VAE vs DAE: in DAE, given an input data, we create a more robust version of that input data with respect to a lot of very small variations, generating noisy data which are mapped all on the same input.

VAE, instead, tells us that given an input x, we can obtain the distribution of the latent space. This distribution can be mapped into the original space in order to obtain the reconstructed input

Conditional Generation (CVAE): add in the training phase of the VAE an additional variable y (both in the decoder and encoder) which let the model to be able to learn the conditional distribution $P(x|y)$. At inference time, this y is used into the decoder part only, in addition to the usual $z \sim N(0,1)$ to obtain new data $\tilde{x}|y$



GAN



Loss: resolve a min-max problem. The generator wants to learn how to generate better and better samples in order to fool the discriminator. This means that the generator wants to minimize the distance between 1 (the value that discriminator uses to label real samples) and the output of the discriminator on generated data $D(G(z))$.

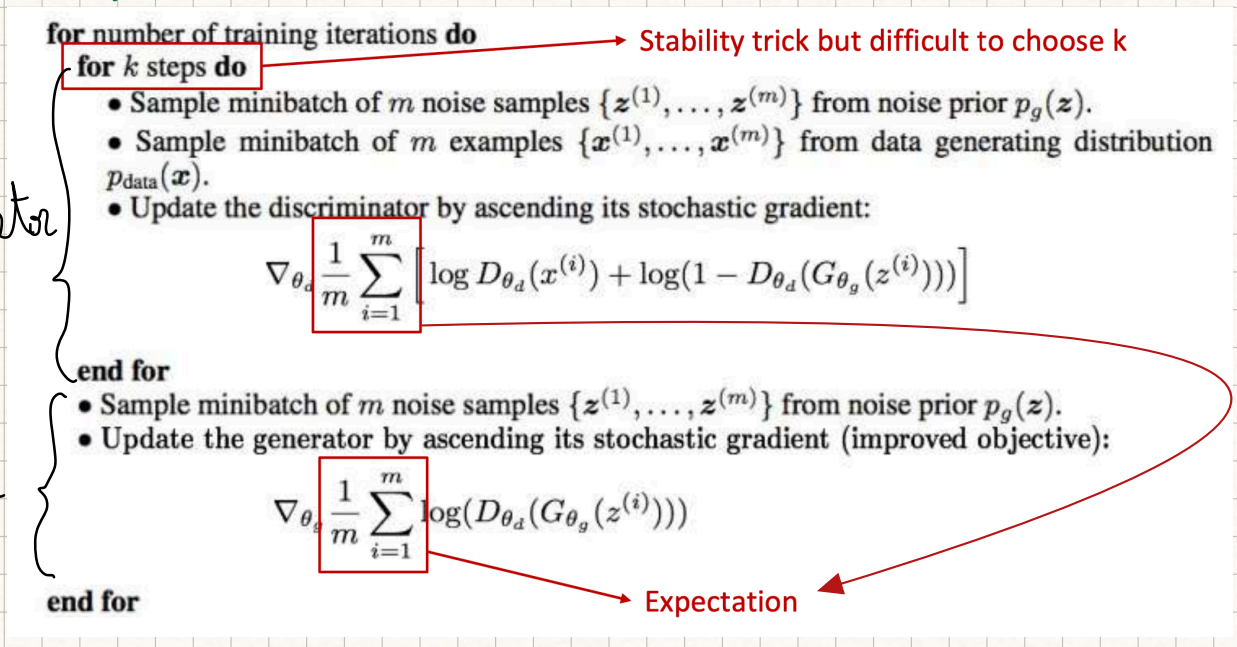
$$C = \min_{\theta_G} \max_{\theta_D} \left[E_{x \sim D}[\log D_{\theta_D}(x)] - E_{z \sim N}[\log(1 - D_{\theta_D}(G_{\theta_G}(z)))] \right]$$

- $\min_{\theta_G}$: gradient descent
- $\max_{\theta_D}$: gradient ascent
- $D_{\theta_D}(x)$: discriminator output for real data x
- $D_{\theta_D}(G_{\theta_G}(z))$: discriminator output for fake data $G(z)$

Optimizing $E_z[\log(1 - D_{\theta_D}(G_{\theta_G}(z)))]$ (generator formula) does not work due to the flat gradient problem for $D(G(z)) = 0$

So we rewrite $C_G = \max_{\theta_G} \mathbb{E}[\log(D_{\theta_D}(G_{\theta_G}(z)))]$ (maximize the likelih. of discr. being wrong)

GAN training pseudo-code



The optimal solutions is a saddle point. We can use the Wasserstein distance between the generator and empirical distribution filtered through the discriminator function D to tackle the hardness of training GAN.

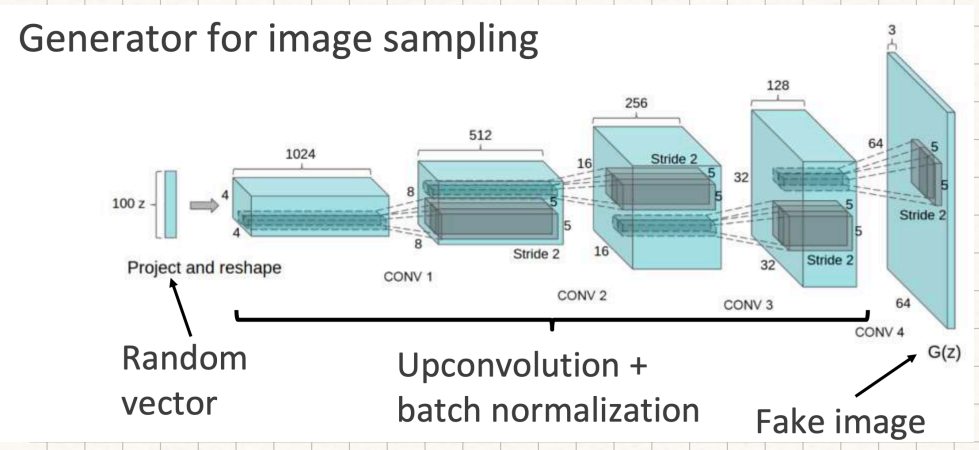
$$G^* = \arg\min_G W(\mu, \mu_G) = \arg\min_G \sup_{\|D\|_L \le 1} [\mathbb{E}_{x \sim \mu}[D(x)] - \mathbb{E}_{x \sim \mu_G}[D(x)]]$$

μ: real data dist. μ_G: generator dist.

Clipping discriminator weights (slow to converge) → the discriminator is contractive because the norm is $\le d$ a constant

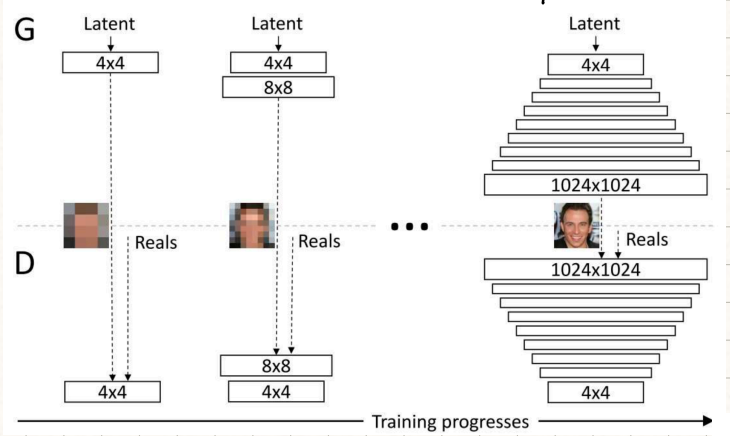
Classical GAN loss results in saturation (discriminator with zero loss). WGAN provide gradient across all the range of training conditions

DCGAN Architecture



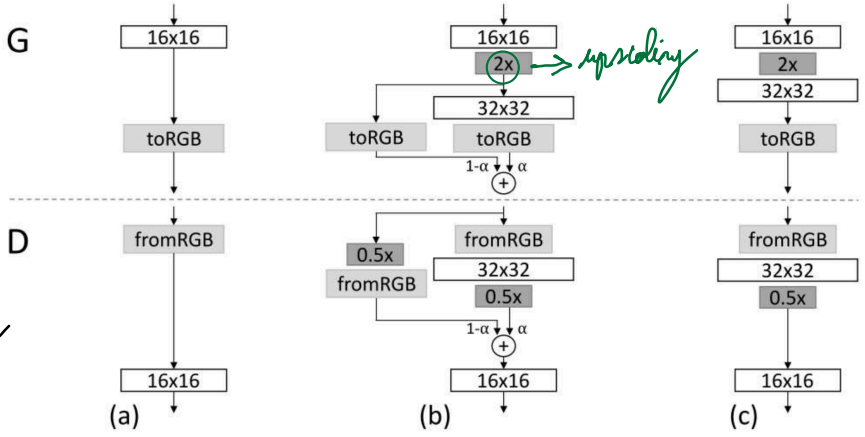
Discriminator is just a Convolution network that takes fake images as input

Progressive GAN: train with very small images, and learn to generate them. Then you move to slightly larger step by step.



But the jump destroys what we've learned in previous steps. We need smooth transitions:

We use residual connections and α to trade between the upscaled smaller residual image and the bigger one, slowly giving

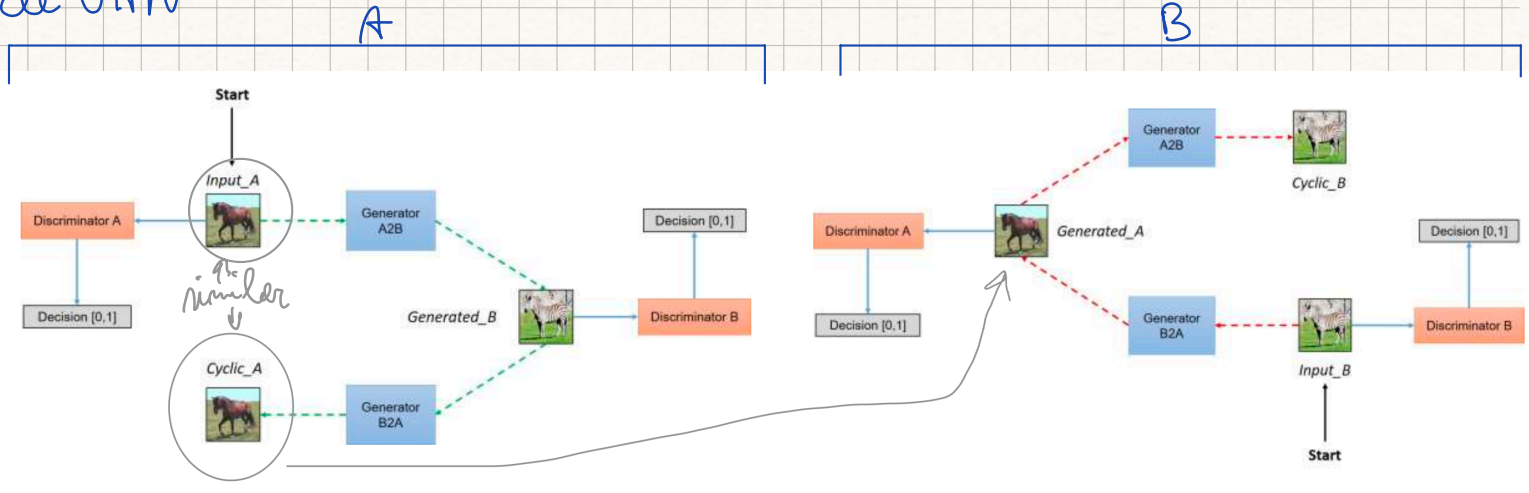


more weight to the larger image as we proceed in the training

Conditional generation: learn a mapping from an observed side information x and a random noise vector z to the fooling samples y $\quad G: \{x, z\} \rightarrow y$

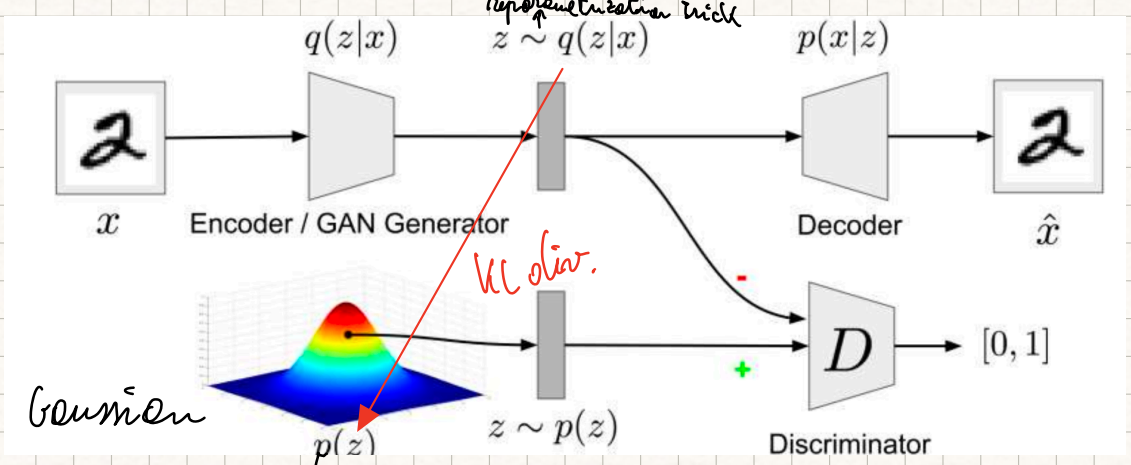
(x: auxiliary information)

Cycle GAN



Enforce alignment by ensuring that generated images in domain B can lead to good fakes in domain A and vice versa

Adversarial autoencoders (AAE)



Force the latent codes to be indistinguishable from samples of a prior distribution. The discriminator discriminates the z from the encoder from the z from the distribution.

$$\mathcal{L}(x) = \tilde{E}_q[\log P(X|z)] - KL(Q(z|x) \| P(z))$$

e.g. (WASSERSTAIN)

replaced by an adversarial loss

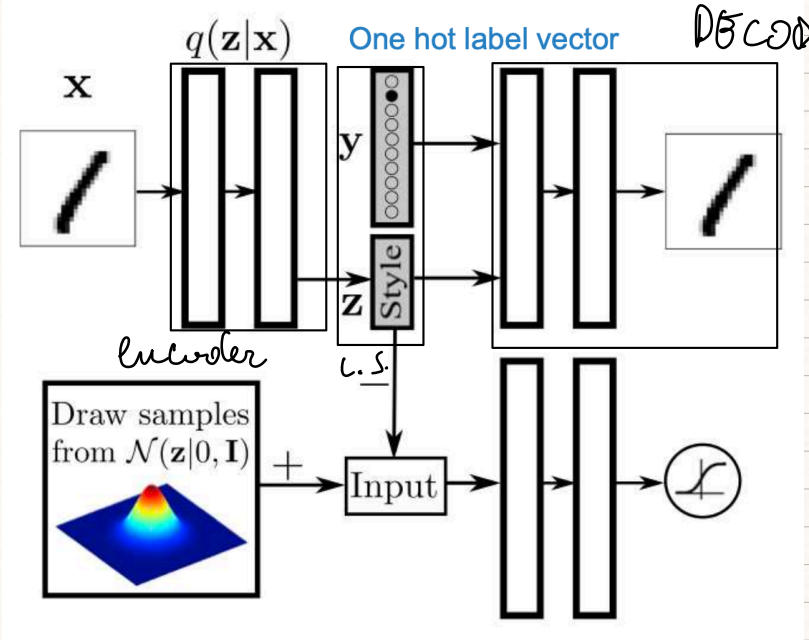
Reconstruction phase: update the encoder and decoder to minimize reconstruction error

Regularization phase: update discriminator to distinguish true prior samples from generated samples; update generator to fool the discriminator.

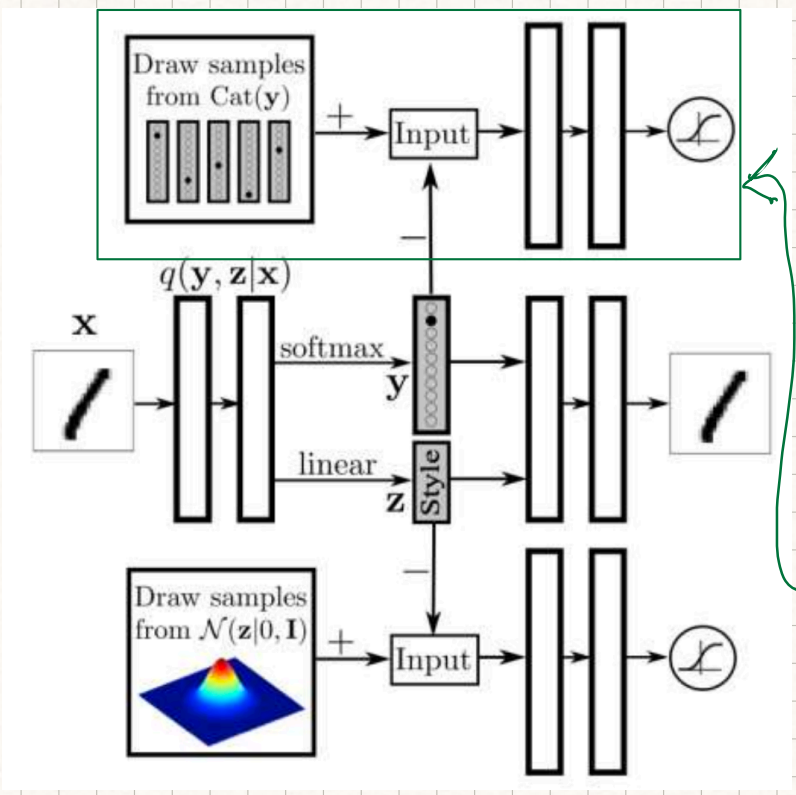
Adversarial regularization allows to impose priors for which we cannot compute KL divergence.

AAE yields a smoother coverage of the latent space.

Style transfer

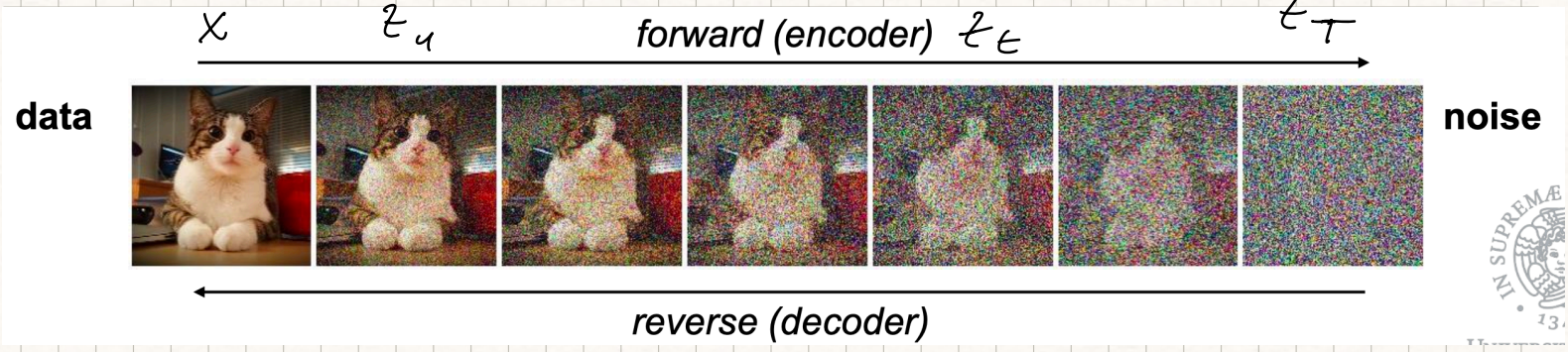


Latent space is formed by a variable that encode the style information and a categorical variable that encode the identity of the number. The decoder uses both information to create a number for a given style. We change the value of z in order to change the style



If I don't have y, I can predict it through a softmax, whose output goes through an adversarial loss. Some images are labeled, some not. The one labeled are used there to confront the prediction of the encoder

Diffusion models



Forward diffusion gradually adding noise to input

reverse process reconstructs data from noise (generation)

In forward diffusion process we add additive noise $\varepsilon_t \sim N(0,1)$ t time to the original data x, obtaining at the end a same-size latent variable z_t.

$$z_1 = \sqrt{1-\beta_1}\, x + \sqrt{\beta_1}\, \varepsilon_1 \qquad z_t = \sqrt{1-\beta_t}\, z_{t-1} + \sqrt{\beta_t}\, \varepsilon_t$$

$\beta_t \in [0,1]$ is the noise schedule

First high frequency information get corrupted, then low frequency

Forward diffusion is clearly a Markov chain

$$q(z_1, \dots, z_T | x) = q(z_1|x) \prod_{t=2}^{T} q(z_t | z_{t-1})$$

$$q(z_t | z_{t-1}) = N\left(\sqrt{1-\beta_t}\, z_{t-1}, \beta_t I\right), \quad z_0 = x$$

Diffusion Kernel: generating z_t sequentially is time consuming so we use a closed-form solution for $q(z_t|\cdot)$

$$q(z_t|x) = \mathcal{N}\left(\sqrt{\alpha_t}\, x, (1-\alpha_t)\, \mathbf{I}\right)$$

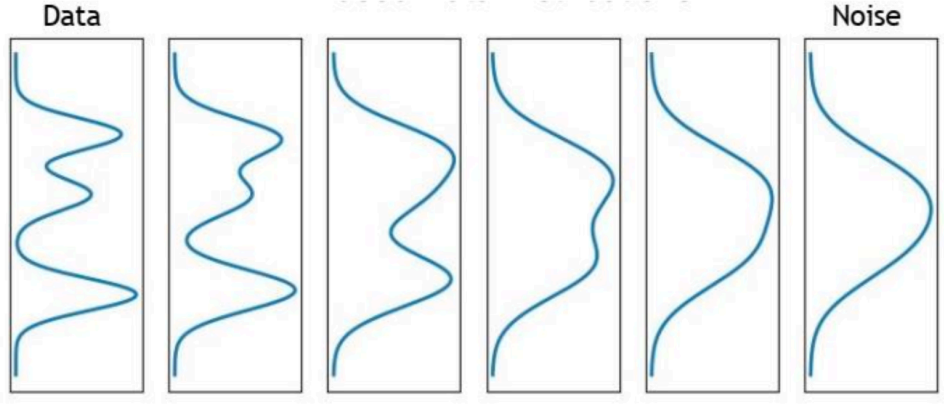
$$\alpha_t = \prod_{s=1}^{t} (1-\beta_s)$$

We draw a sample from $q(z_t|x)$ to obtain z at time t in one shot.

I can write the marginal as: $q(z_t) = \int q(z_t, x)\, dx = \int q(x)\, q(z_t|x)\, dx$

$$q(\mathbf{z}_t) = \int q(\mathbf{z}_t, \boldsymbol{x})\, d\boldsymbol{x} = \int q(\boldsymbol{x}) q(\mathbf{z}_t|\boldsymbol{x})\, d\boldsymbol{x}$$

no parameter in the forward process.



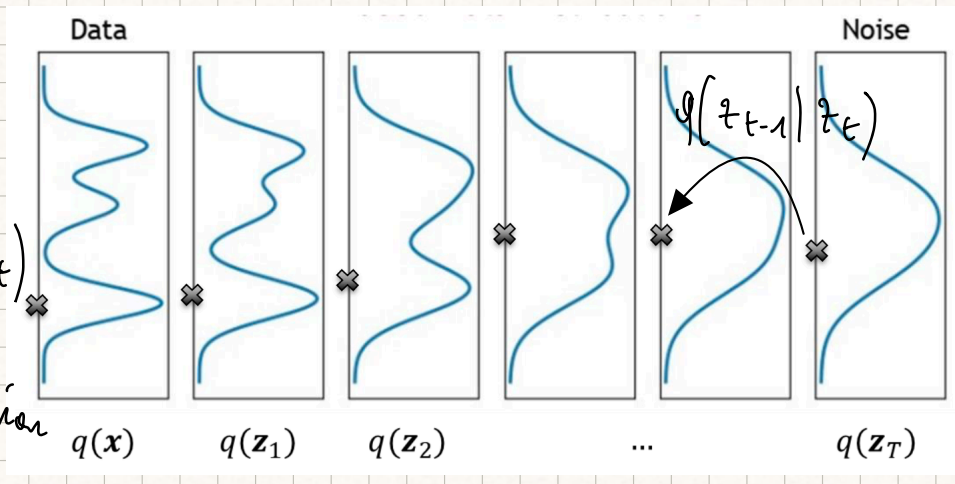
← fourier

Denoising

sample $z_T \sim \mathcal{N}(0, 1)$

iterate $z_{t-1} \sim q(z_{t-1}|z_t)$

true denoising distribution



is intractable $q(z_{t-1}|z_t) = \frac{q(z_{t-1})\,q(z_t|z_{t-1})}{q(z_t)}$

If we know x we can demonstrate that $q(z_{t-1}|z_t, x)$ is Normal

the Reverse process learns an approximated denoising dist. (decoder), assuming reverse distributions are approximately normal.
Basically we have a NN that outputs a mean ~~and a variance~~

$$p(z_t) = \mathcal{N}(0, I)$$

$$p_\theta(z_{t-1}|z_t) = \mathcal{N}(\mu_\theta(z_t, t), \sigma_t^2 I)$$

actually the variance is like the noise schedule, is not a hyperparameter.

mean of the denoised image z_{t-1} predicted by the θ-parametrized model given z_t (and time encoding)

Training follows the classical log-likelihood maximization view

$$\log p_\theta(x) = \log \int p_\theta(z_1, \dots, z_T, x)\, dz_{1,\dots,T} =$$

$$= \log \int p_\theta(x|z_1) \prod_{t=2}^{T} p_\theta(z_{t-1}|z_t)\, p_\theta(z_T)\, dz_{1\dots T}$$

UNTRACTABLE

We use ELBO. introduce the encoder distribution q with $\bar{z} = z_1, \dots, z_T$

$$\theta^* = \arg\max \log \int p_\theta(\bar{z}, x)\, d\bar{z} \geq \int q(\bar{z}|x) \log \left[\frac{p_\theta(\bar{z}, x)}{q(\bar{z}|x)} \right] d\bar{z}$$

We can rewrite as

$$\mathbb{E}_{q(z_1|x)}[\log p_\theta(x|z_1)] - \sum_{t=2}^{T} KL(p_\theta(z_{t-1}|z_t) \| q(z_{t-1}|z_t, x))$$

expected reconstruction quality: how much the model is good at generating samples which are consistent with the data distribution, the data dist. is approximated by the empirical dist. of the TR.

how good the learned denoising dist p_θ is w.r.t. q. Both dist. will be normal so we can solve the KL divergence in closed form.

$$\sum_x \left(\underbrace{(-\log \mathcal{N}(\mu_\theta(\mathbf{z}_1, t), \sigma_1^2 \boldsymbol{I}))}_{\text{Reconstruction}} + \sum_{t=2}^{T} \frac{1}{2\sigma_t^2} \left\| \underbrace{\left(\frac{(1-\alpha_{t-1})}{(1-\alpha_t)} \sqrt{1-\beta_t} \mathbf{z}_t + \frac{\sqrt{\alpha_{t-1}} \beta_t}{(1-\alpha_t)} \boldsymbol{x} \right)}_{\text{Target mean of } q(\mathbf{z}_{t-1}|\mathbf{z}_t, \boldsymbol{x})} - \underbrace{\mu_\theta(\mathbf{z}_t, t)}_{\text{predicted } \mathbf{z}_{t-1}}, \sigma_t^2 \boldsymbol{I} \right\|^2 \right)$$

Minimize the difference between estimate of z_{t-1} and the most likely value from ground truth. obtained data.

In practice, rather than predicting the mean of the distribution at time T, I predict the noise that I have to extract at time T (that I have put in the forward phase). So the NN outputs an error $\mathcal{E}_\theta$.

We can rewrite as: $x = \frac{1}{\sqrt{\alpha_t}} z_t - \frac{\sqrt{1-\alpha_t}}{\sqrt{\alpha_t}} \mathcal{E}$

inserting x in the BBO yields

$$Loss(\theta) = \sum_x \sum_{t=1}^{T} \| \mathcal{E}_\theta (\underbrace{\sqrt{\alpha_t} x + \sqrt{1-\alpha_t}\, \mathcal{E}_t}_{z_t}, t) - \mathcal{E}_t \|^2$$

$\mathcal{E}_\theta$: a network which predict the unit noise given current noised input z_t and an embedding of the time

$\mathcal{E}_t$: add in the forward phase

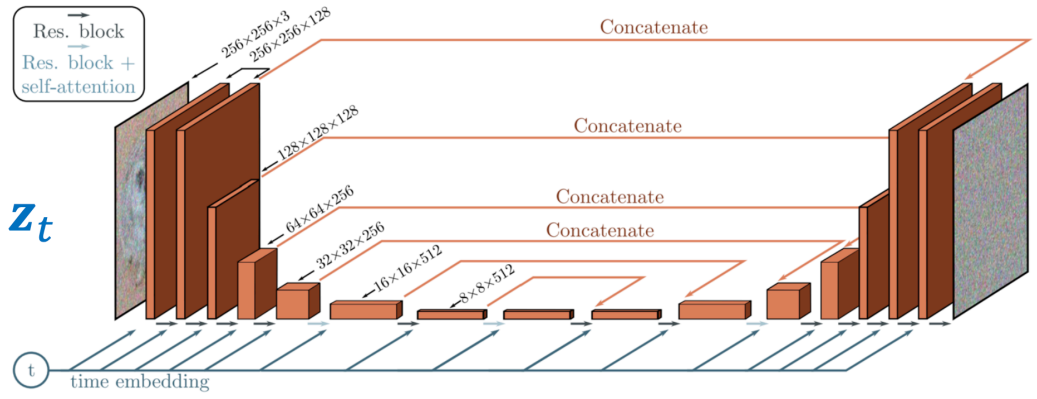
Algorithm 18.1: Diffusion model training

Input: Training data $\mathbf{x}$
Output: Model parameters θ
repeat
  for $i \in \mathcal{B}$ **do** // For every training example index in batch
    $t \sim \text{Uniform}[1, \ldots T]$ // Sample random timestep
    $\boldsymbol{\epsilon} \sim \text{Norm}[\mathbf{0}, \mathbf{I}]$ // Sample noise
    $\ell_i = \left\| \boldsymbol{\epsilon}_\theta(\sqrt{\alpha_t}\mathbf{x}_i + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}, t) - \boldsymbol{\epsilon} \right\|^2$ // Compute individual loss
  Accumulate losses for batch and take gradient step
until converged

During forward we add noise to image.

During reverse we predict that noise with a DNN and then subtract it from the image to denoise it.

U-Net architectures with ResNet blocks and self-attention layers



Sinusoidal positional embeddings

Time features are fed to residual blocks

Terms β_t and σ_t control variance of forward diffusion and reverse denoising, respectively. β_t linear schedule and $\sigma_t^2 = \beta_t$

Slowly increase the amount of added noise (as high-res information is corrupted first).

σ_t can be learned by minimizing the bound.

β_t can be learned by minimizing the variance of the training objective

High res-images → Concatenation of UNET

from z_t → [UNET] → ϵ_θ to [UNET] → (2×, 32×32) [UNET] → (2×, 64×64) [UNET]

Guided generation: guide diffusion process using auxiliary data c, using the gradient of a trained classifier as guidance.

1 train the diffusion model unconditionally

2 train a classifier $P(c|z_t)$, where c are conditioning labels

3 add an extra term when sampling the diffusion model, i.e. when reconstructing the image z_{t-1} from z_t, that modifies the reconstruction in the direction given by the gradient of a classifier.

$$z_{t-1} = \underbrace{\hat{z}_{t-1} + \sigma_t \varepsilon}_{\text{reversed diffusion}} + \underbrace{\sigma_t^2 \frac{\partial P(c|z_t)}{\partial z_t}}_{\text{classifier guidance}}$$

classifier guidance comes from mixing the predicted score function of the unconditioned diffusion model with the classifier gradient

$$\frac{\partial \log P_\gamma(z_t|c)}{\partial z_t} = \frac{\partial \log P(z_t)}{\partial z_t} + \gamma \frac{\partial \log P(c|z_t)}{\partial z_t}$$

give me a (noised) version of a image with label c

guidance scale, higher γ more label c image, low γ not so class c image

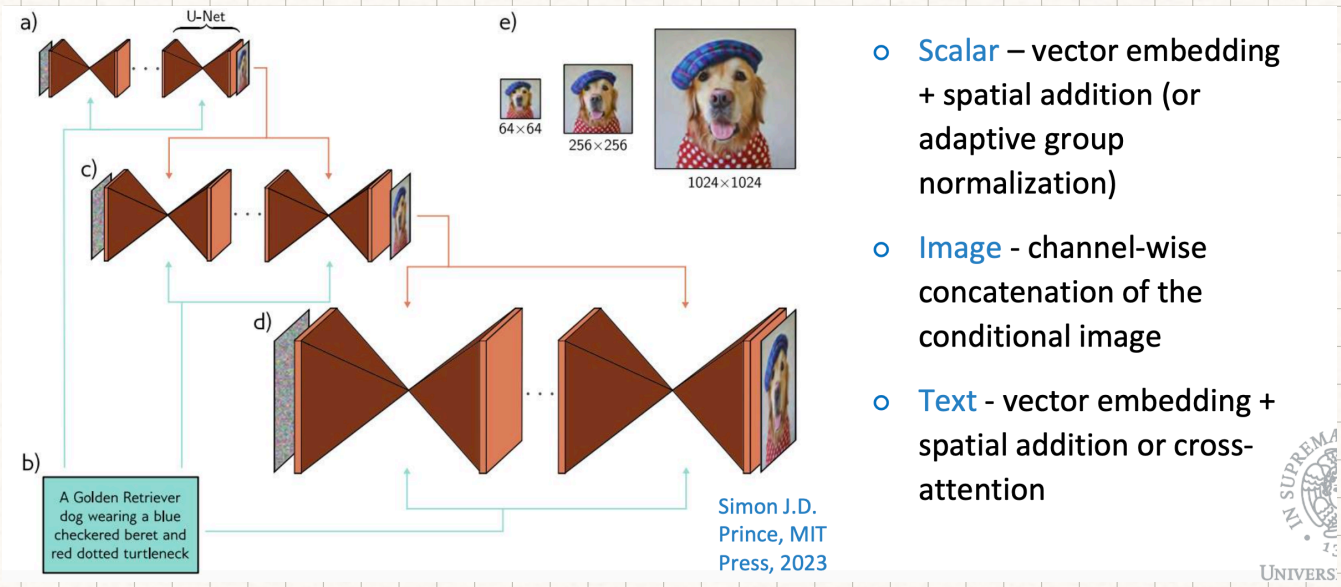
We can obtain a guided version of diff. model mixing an unconditioned one with a conditioned one.

$$\frac{\partial \log P_\gamma(z_t|c)}{\partial z_t} = (1-\gamma) \underbrace{\frac{\partial \log P(z_t)}{\partial z_t}}_{\text{unconditioned}} + \gamma \underbrace{\frac{\partial \log P(z_t|c)}{\partial z_t}}_{\text{conditioned}}$$

We don't need to train two different models. We train conditional diffusion model with dropout (randomly removing conditioning) Conditioning replaced by fixed input (presence/absence of conditioning) c label is given in input with the time embedding.

Diffusion models can be used to do image segmentation, denoising back to the image segmentation space

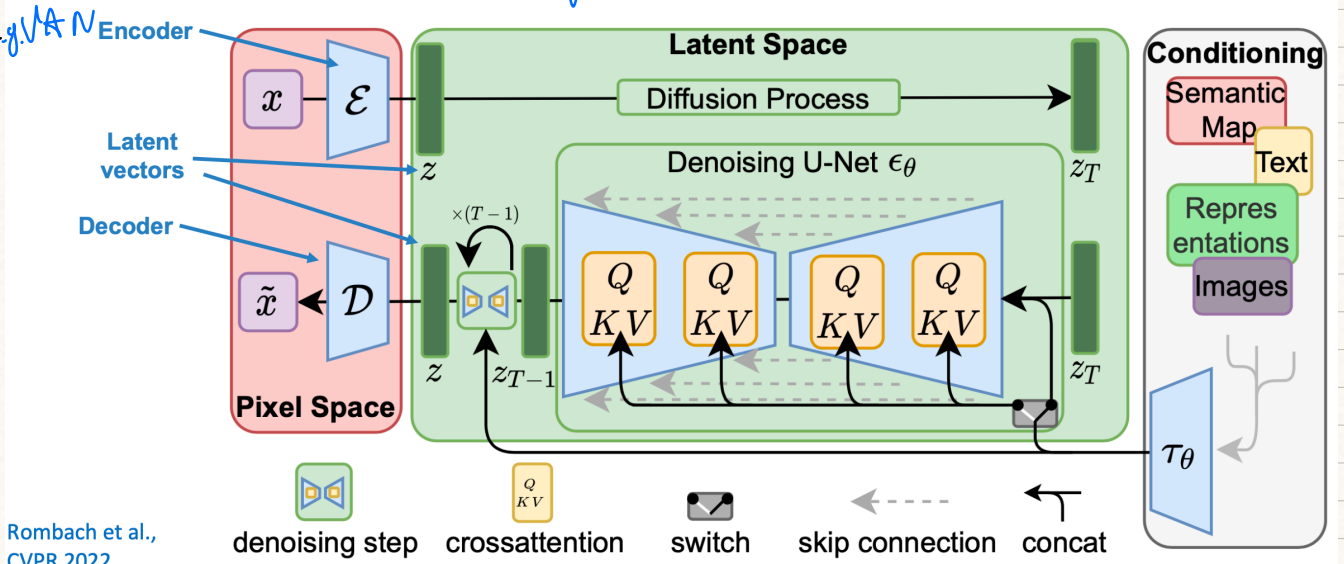
Cascade conditional generation



Simon J.D. Prince, MIT Press, 2023

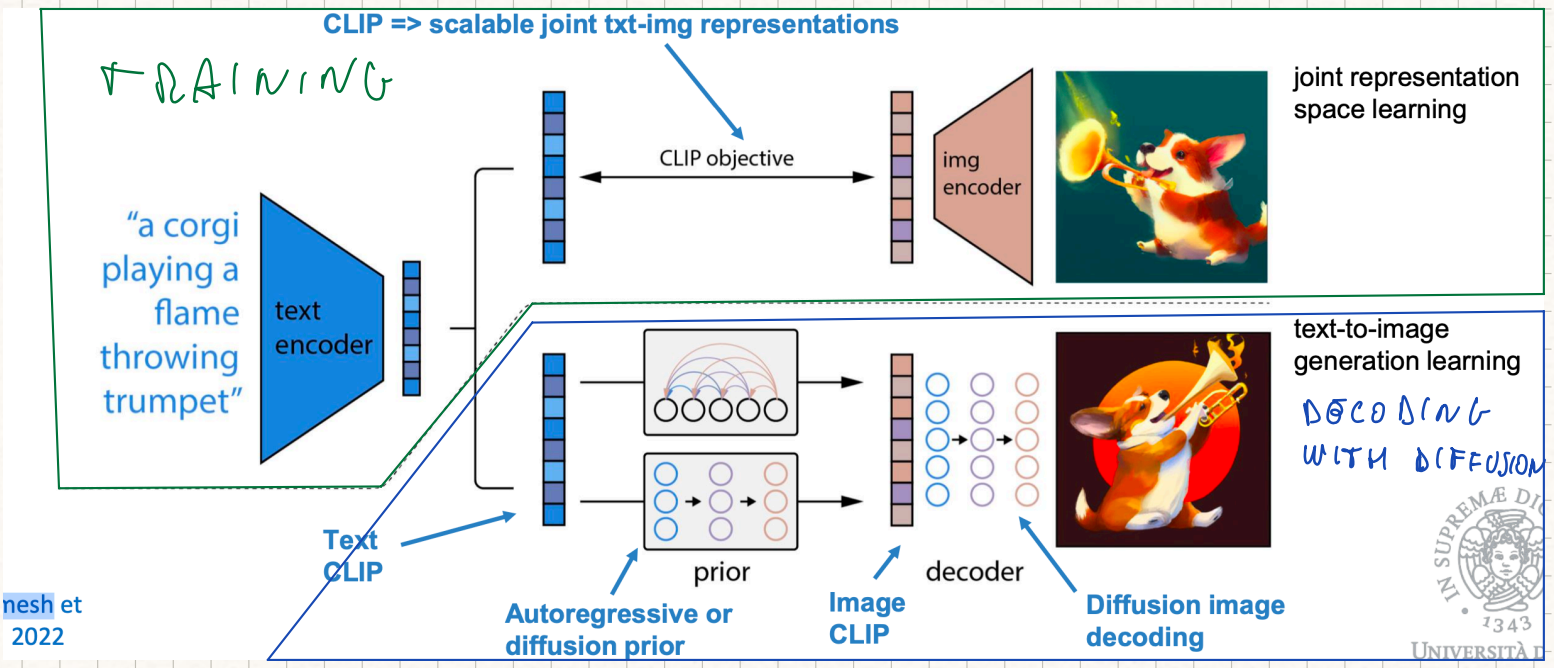
- Scalar – vector embedding + spatial addition (or adaptive group normalization)
- Image - channel-wise concatenation of the conditional image
- Text - vector embedding + spatial addition or cross-attention

Latent space diffusion



Rombach et al., CVPR 2022

DALL-E 2 (diffusion in latent space)



CLIP = encode text and images in a latent space in which they are semantically aligned. The scalar product of the embedding of a text and the embedding of the corresponding image are aligned, the scalar product of the embedding of a text and the embedding of images that not correspond to that text are unaligned.

At generation time, first encode text in TEXT CLIP, then transform it in IMAGE CLIP, and finally decode in a new image.