

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:

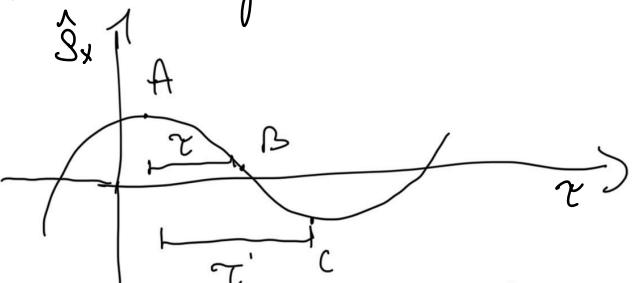
- $E[x_t] = \mu$  (doesn't change with time  $t$ )
- $\text{Cov}(x_{t+\tau}, x_t) = \gamma_\tau \quad \forall t$  ( $\gamma$  depends only on lag  $\tau$ )  
 $\hookrightarrow$  lag between observation

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

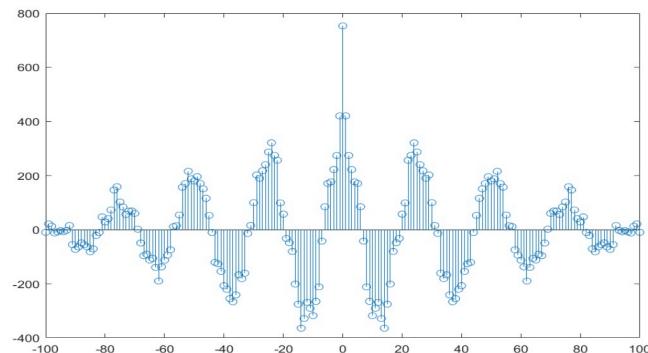
Autocovariance, for lag  $-N \leq \tau \leq N$ :  $\hat{\gamma}_x(\tau) = \frac{1}{N} \sum_{t=1}^{N-|\tau|} (x_{t+|\tau|} - \hat{\mu})(x_t - \hat{\mu})$

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

Autocorrelation analysis can reveal repeating patterns such as periodic signals



$r_x(\tau)$  ha autocorrelazione massima perché il segnale si ripete ma cambiato di segno



Autocorrelogram reveals a sine wave



**Cross-Correlation:**  $x^1, x^2$  time series. A measure of similarity as a function of a time lag  $\tau$

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

$\tau \in [-(T^1-1), \dots, 0, \dots, (T^2-1)]$ . The maximum  $\phi_{x^1 x^2}(\tau)$  w.r.t.  $\tau$

identifies the displacement of  $x^1$  vs  $x^2$  (shiftments)

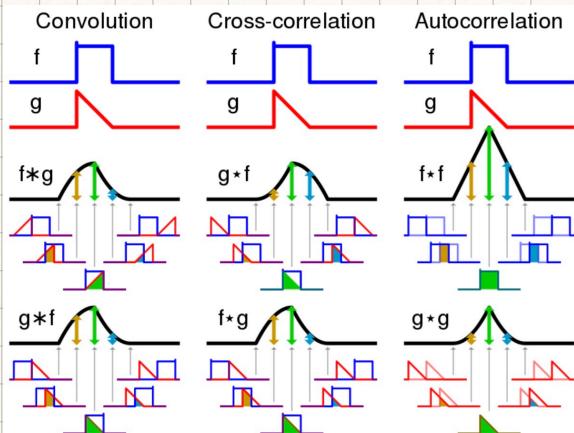
**CC normalized:**  $\bar{\phi}_{x^1 x^2}(\tau) = \frac{\phi_{x^1 x^2}}{\sqrt{\sum_{t=0}^{T^1-1} (x^1(t))^2} \sqrt{\sum_{t=0}^{T^2-1} (x^2(t))^2}} \in [-1, +1]$

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

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

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

$\xrightarrow{K \text{ memory, parameters}}$

$$X_t = \sum_{k=1}^K \lambda_k X_{t-k} + \epsilon_t$$

$$\lambda = \{\lambda_k\}_{k=1}^K$$

$X_t$  regresses on itself.  $\lambda_k \Rightarrow$  linear coefficients s.t.  $|\lambda| < 1$

$\epsilon_t \Rightarrow$  sequence of i.i.d. values with  $E[\epsilon_t] = 0$   $\text{Var}[\epsilon_t] = \sigma^2$

!!!  
..

ARMA (Autoregressive with Moving Average process)

$$X_t = \underbrace{\sum_{k=1}^K \lambda_k X_{t-k}}_{\text{reg. of TS}} + \underbrace{\sum_{q=1}^Q \beta_q \epsilon_{t-q}}_{\text{reg. of the past errors (stochastically uncorrelated information)}} + \epsilon_t \xrightarrow{\text{white noise}}$$

denote new information at time  $t$

Need to estimate :  $\lambda$  and  $\beta$ , the order of  $K$  and  $Q$ .

Can use the set of autoregressive parameter  $\lambda_1^{(i)}, \dots, \lambda_K^{(i)}$   
fitted to a specific time series  $x^{(i)}$  to confront it to other TS

- TS clustering  $\delta(x^1, x^2) = \|\lambda^1 - \lambda^2\|_m^2$

- Anomaly / anomaly detection Test Error  $(x_t, \hat{x}_t) \in \delta$

$\hat{x}_t$  AR predicted value

Spectral analysis: analyzing TS in the frequency domain

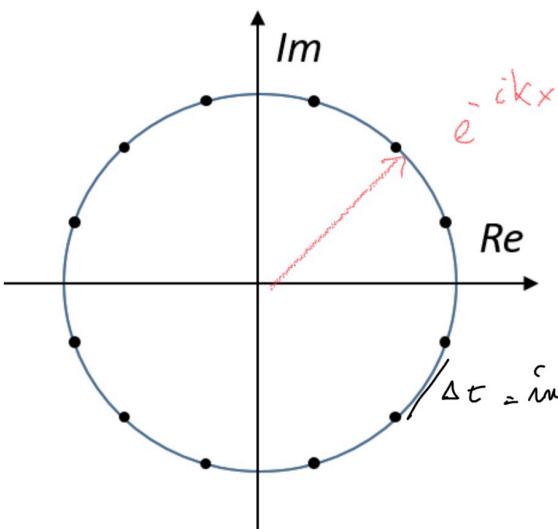
(linear combination of  $\sin(kx)$  and  $\cos(kx)$ )

Given an orthonormal system  $\{l_1, l_2, \dots, l_m\}$  for  $E$ , we can represent any  $f \in E$  by:  $\sum_1^{\infty} \langle f, l_n \rangle l_n$

Fourier Series:  $\frac{Q_0}{2} + \sum_{k=1}^{\infty} [Q_k \cos(kx) + Q_k' \sin(kx)]$

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

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



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

using the exponential formulation, the orthonormal system is made of  $\{l_0, l_1, \dots, l_{N-1}\}, l_k \in \mathbb{C}^N$ . The  $n$ -th element of  $l_k$  is:  $[l_k]_n = e^{-i\pi k n / N}$

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

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

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

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 :  $A_k = |X_k| = \sqrt{R_e^2(X_k) + F_m^2(X_k)}$

Power :

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

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

$\rightarrow$   $k$ -th frequency (Hz)

$\rightarrow$  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  $\mu$

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

- Kurtosis - (4<sup>th</sup> order moment) Measures flatness or non-Gaussianity of the spectrum around the centroid  $\mu$

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



## Image description

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

Intensity vector  $w$  (grey code) 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  $\mathbb{I}$
- 2 Represent Image at scale  $\sigma$ , convolve it with Gaussian filter with std  $\sigma$  like a magnifier

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

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

$\sigma$  high  $\rightarrow$  blur image

$\sigma$  low  $\rightarrow$  clear image

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

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

$$\frac{\partial L_f}{\partial y} = G_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \Rightarrow L_f(x, y)$$

intensity of the pixel

$\uparrow$   
Magnitude  $m_f(x, y) = \sqrt{G_x^2 + G_y^2}$

Orientation  $\theta_f(x, y) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$   
 $\downarrow$   
direction of the pixel

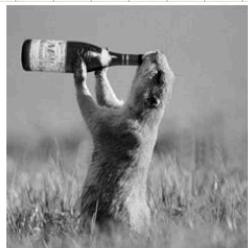
#### 4 Create gradient histogram

- $16 \times 16$  neighbourhood is taken around the point, dividing it in  $4 \times 4$  gradient window
  - Histogram of  $4 \times 4$  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 unify 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 2D-DFT

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

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



$$= c_1 + c_2 + c_3 + \dots$$

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

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

a filter  
(gaussian)

$F^{-1}$  inverse  
fourier transf.

Image, a function returning  
intensity value

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

• Image gradient (graylevel)  $\nabla I = \left[ \frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right]$  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

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

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

Blob detection: blobs are connected pixel regions with little gradient variability. We use Laplacian of Gaussian as a filter  $\gamma_\sigma(x, y)$  has maximum response when centered on a circle of radius  $\sqrt{2}\sigma$

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

normalized multiplying by  $\sigma^2$

- 1 Convolve image with a LoG filter at different scales, starting from a small  $\sigma$ , 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 consider:

- Find maxima on scale-scale  $\Rightarrow$  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 difference of Gaussian

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

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

## Maximally Stable Extremal Regions (MSER)

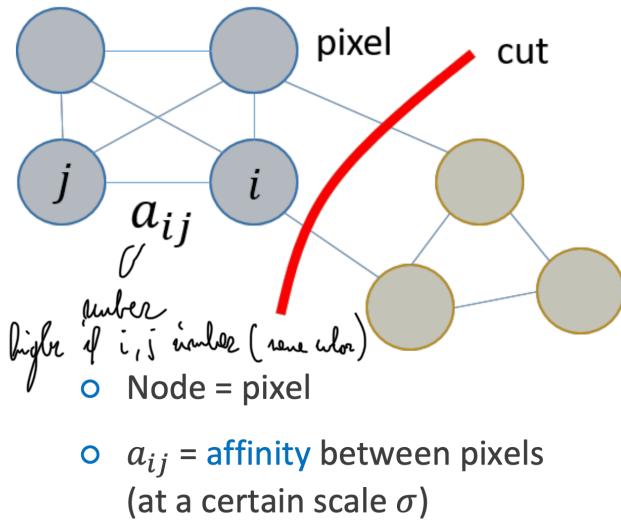
Blobs

- 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 met by scale)
    - Sensitive to local lighting effects, shadows...
- ↳ We can fit an ellipse enclosing the stable region generates frames from the image by thresholding it on all graylevels. Capture those regions that from a small set of pixel grow to a stably connected region. Stability is ensured by looking at derivatives of region masks in time (most stable  $\Rightarrow$  minimum of connected region variation)

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

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

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.

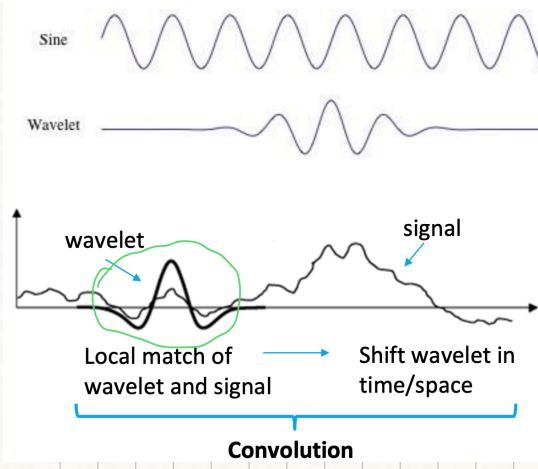


The multilevel cut problem is NP-hard. Approximate solution is an eigenvalue problem, but it's tractable with big images. We can use superpixels: cluster the pixel with K-means and use the cluster as nodes.

We can apply Ncut to Image Segmentation problem  $\Rightarrow$  Superpixel trick

- Group together similar pixels. 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 signal in real
  - 2 Compare signal to a wavelet
  - 3 Compute a coefficient of similarity

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

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

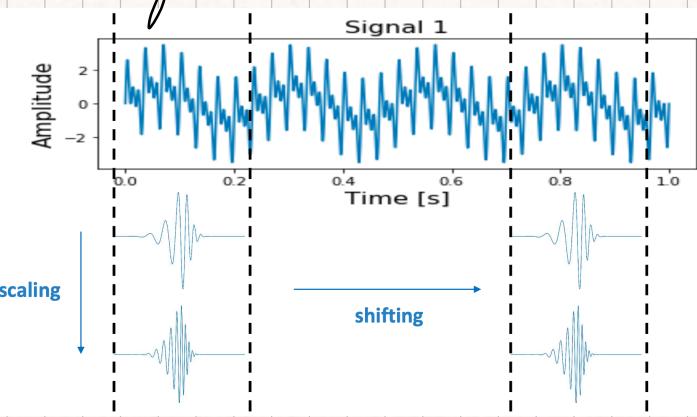
$j, k$  regulate scaling and shifting of

the wavelet

$$\Psi_{j,k}(t) = 2^{\frac{k}{2}} \Psi((t - j2^k)/2^k)$$

$j < 1$  compresses the signal,  $j > 1$  dilates the signal  
another 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.  $P(X=x) \xrightarrow{\text{possible outcome}} \xrightarrow{\text{L} \rightarrow \text{RV}}$

We have discrete/continuous RV

$$\Rightarrow P(X=x) \in [0,1] , \sum_{x \in \Omega} P(X=x) = 1 \quad \xrightarrow{\text{P(t) describe the probability of a RV to take the value t}}$$
$$\int_{-\infty}^{+\infty} p(t) dt = 1$$

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

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

$$P(X_1=x_1, \dots, X_n=x_n) = P(x_1 \wedge \dots \wedge x_n)$$

The joint conditional probability of  $x_1, \dots, x_n$  given  $y$ :

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

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

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

**Marginalization:** marginalizes rispetto a un solo insieme 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 hypothesis  $h_i \in H$  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 hypothesis  $h_i$  is true (likelihood)

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

$P(h_i | d)$  is the posterior probability that hypothesis is true given the state and the previous belief about the hypothesis

**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 independent 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

$$\text{Bayesian} = P(X|d) = \sum_i P(X|h_i) P(h_i|d) \xrightarrow{\text{A}} \text{weight all hypotheses by its posterior prob.}$$

$d$  = DATASET I.I.D.

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

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

MAP regularize w.r.t.  $\text{MCF}$  because it consider the model complexity ( $P(h)$ )

$\text{MLG}$ , used usually to perform model selection. It maximize  $A$ .

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

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

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

MLG 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.

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

MAP implies choosing the hypothesis that provides maximum comprehension

number of bits required to specify h

Maximum likelihood Estimation: find the model  $\theta$  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  $\theta$ .

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

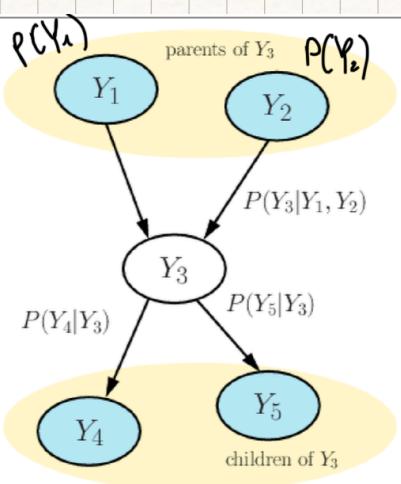
With both observed/hidden variables:

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

2-step size iterative process

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

## Bayesian Network (DAG)



$$\text{DAG } G = (\mathcal{V}, \mathcal{E})$$

nodes  $v \in \mathcal{V}$  represent random variables

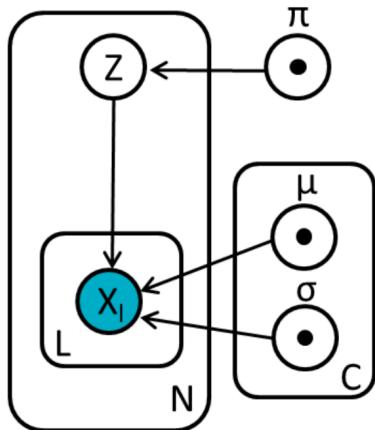
Shaded = observed ; Unshaded = un-observed

edges  $e \in \mathcal{E}$  describe the conditional independence

Conditional Distribution Table (CPT)

$$P(Y_1, \dots, Y_n) = \prod_{i=1}^n P(Y_i | \text{par}(Y_i))$$

If the  $N$  nodes have a maximum of  $L$  children  $\Rightarrow (\underbrace{k-1}_\text{distinct value of R.V.})^L \times N$  independent parameters



- 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



### Gaussian Mixture Model

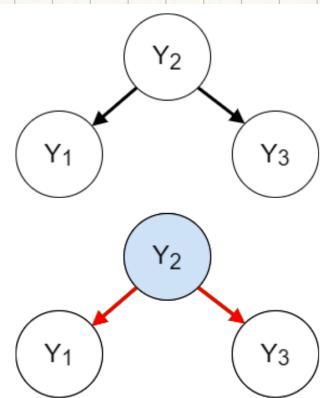
**Local Markov Property:** each RV is conditionally independent of all its non-descendants given a joint state of its parent

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

**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.  $P(A \mid Mb(A), z) = P(A \mid Mb(A)) \wedge z \notin Mb(A)$

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

**Tail-to-Tail Connections:**  $P(Y_1, Y_3 \mid Y_2) P(Y_2) = P(Y_1 \mid Y_2) P(Y_3 \mid Y_2) P(Y_2)$



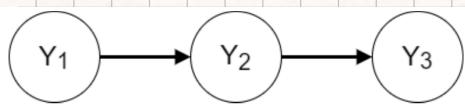
if  $Y_2$  is unobserved,  $Y_1$  and  $Y_3$  are marginally dependent  $Y_1 \not\perp\!\!\!\perp Y_3$

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

$$Y_1 \perp Y_3 | Y_2$$

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

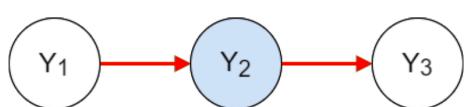
**Head to Tail:**



$$P(Y_1, Y_2, Y_3) = P(Y_1) P(Y_2 | Y_1) P(Y_3 | Y_2) =$$

$$= P(Y_1 | Y_2) P(Y_3 | Y_2) P(Y_2)$$

using Bayes rule



Observed  $Y_2$  blocks  
the path from  $Y_1$  to  $Y_3$

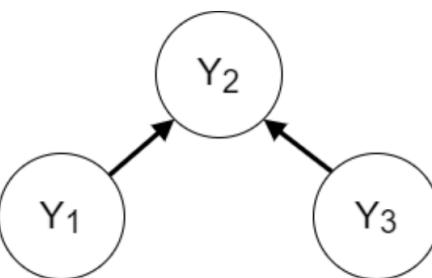
- if  $Y_2$  is unobserved,  $Y_1$  and  $Y_3$  are marginally dependent

$$Y_1 \not\perp Y_3$$

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

$$Y_1 \perp Y_3 | Y_2$$

**Head to Head:**



$$P(Y_1, Y_2, Y_3) = P(Y_1) P(Y_2) P(Y_3 | Y_1, Y_2)$$

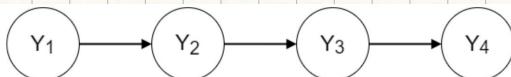
- if  $Y_2$  is observed,  $Y_1$  and  $Y_3$  are conditionally dependent

$$Y_1 \not\perp Y_3 | Y_2$$

- 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

$$Y_1 \perp Y_3 | Y_2$$

$$Y_4 \perp Y_2 | Y_3$$

Derived Relationship

$$Y_1 \perp Y_4 | Y_2$$

**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  $Y_i \rightarrow Y_c \rightarrow Y_j$  and  $Y_c \in Z$
- " " " tail-to-tail ",  $Y_i \leftarrow Y_c \rightarrow Y_j$  and  $Y_c \in Z$
- " " " head-to-head ",  $Y_i \rightarrow Y_c \leftarrow Y_j$  and neither  $Y_c$  nor its descendants are in  $Z$

**Nodes d-separation:** Two nodes  $Y_i$  and  $Y_j$  in OBN  $G$  are said to be d-separated by  $Z \subseteq V$  (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-separate 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 :  $A, B \subset V$  are conditionally independent given  $C \subset V / \{A, B\}$  if all paths between nodes in  $A$  and  $B$  pass through at least one of the nodes in  $C$

$A \perp B | C$

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 convenient and widely used strictly positive representation of the potential function is

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

↳ energy function on  $C$  max clique

Structural learning problem: find the structure of the graph, defining edges between nodes and therefore 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 = ~~both~~ likelihood + model complexity

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

Bayesian = likelihood + prior desire

Search strategy: find maximal scoring structures is NP complete.

Constraint search strategy: starting from a candidate structure modify iterately by local operations (edge/node ins. del.). Each operation has a cost  $\rightarrow$  Cost opt. problem (hill-climb, min. queuing.)

Constraint search space: common node order  $\rightarrow$  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:  $G^2(X_i, X_j | Z) = 2 \sum_{x_i, x_j, z} n_D(x_i, x_j, z) \frac{n_D(x_i, x_j, z) n_D(z)}{n_D(x_i, z) n_D(x_j, z)}$

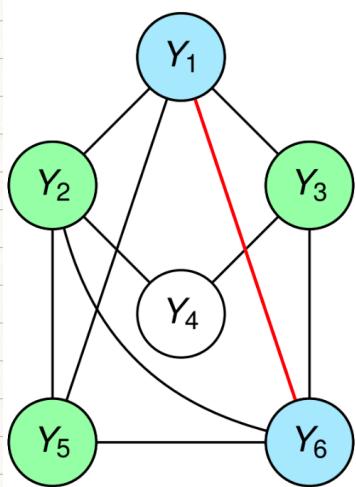
Use deterministic rules based on local Markov dependencies to determine

## edge orientation (DAG)

Testing strategy : Level-wise  $\rightarrow$  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  $G = (\mathcal{V}, \mathcal{E})$

**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**  $G$

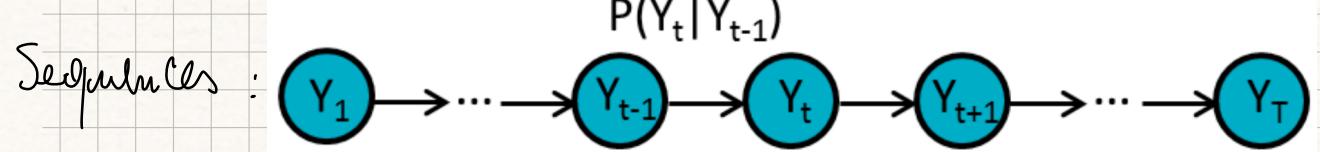
Node-wise : tests are performed on a single edge at the time, exhausting independence checks on all conditioning variables.

TOPA  
Alg

Nodes that enter  $t$  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 refine 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 refine approach

# Hidden Markov Models



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 lengths

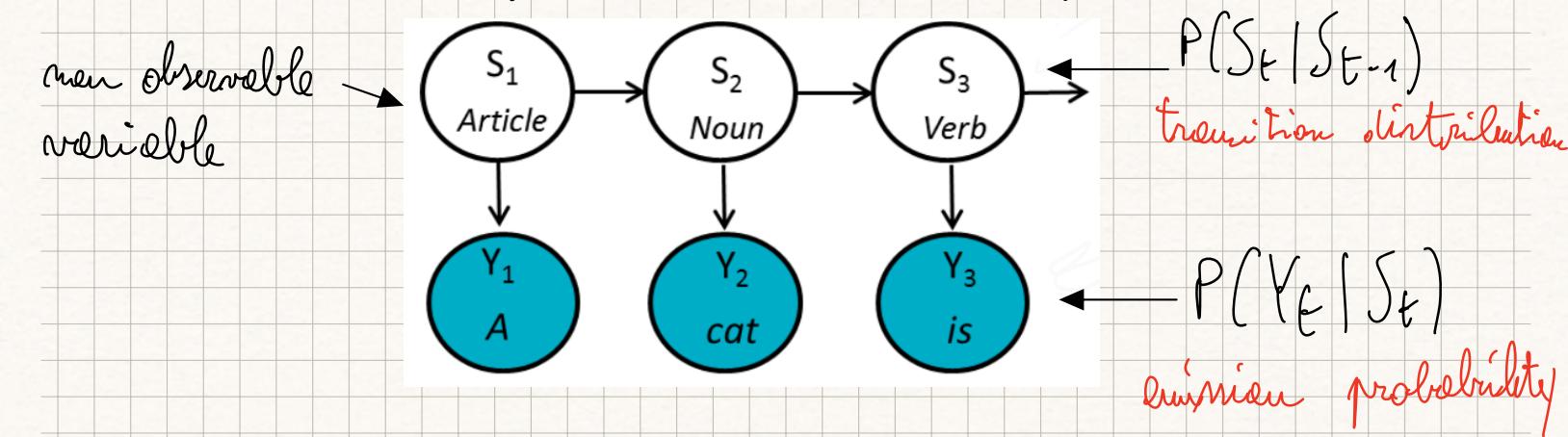
**Markov Chain:** First order MC are directed graphical model for sequences w.r.t. element  $x_t$  only depends on its predecessor in the sequence.

- Joint probability factorizes as

$$P(x) = P(x_1, \dots, x_T) = P(x_1) \prod_{t=2}^T P(x_t | x_{t-1})$$

prior distribution    transition distribution

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



State transition is an observed process characterized by the hidden state variable  $S_t$ , which are often discrete with value in  $\{1 \dots c\}$ . Multinomial state transition and prior probability table  $\leftarrow A_{ij} = P(S_t=i | S_{t-1}=j)$   $\pi_i = P(S_1=i)$

Observation are generated by the emission distribution

stable if  $y$  is discrete, possibly a gaussian if  $y$  is continuous

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

$$P(Y_1 \dots Y_T) = P(Y=y) = \sum_S P(Y=y, S=s) = \\ = \sum_{s_1 \dots s_T} \left\{ P(S_1=s_1) P(Y_1=y_1 | S_1=s_1) \prod_{t=2}^T P(S_t=s_t | S_{t-1}=s_{t-1}) \cdot P(Y_t=y_t | S_t=s_t) \right\}$$

reliability distribution in Markov chain

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

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

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

**Optimal State Assignment:** given a model  $\Theta$  and an observed sequence  $Y$ , find an optimal state assignment  $S = S_1^*, \dots, S_T^*$  for the HMM

**Forward-Backward Algorithm for smoothing**

$$P(S_t = i | Y) \underset{\text{proportional } P(Y)}{\approx} P(S_t = i, Y) = P(S_t = i, Y_{1:t}, Y_{t+1:T}) = P(Y_{t+1:T} | S_t = i) P(S_t = i, Y_{1:t}) = \beta_t(i) \lambda_t(i)$$

↑ future mess.      ↓ past mess.

$\lambda$  computed as part of forward recursion

$$\lambda_t(i) = \overbrace{P(y_1 | S_1 = j)}^{b_i(y_1)} \overbrace{P(S_1 = j)}^{\pi:}$$

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

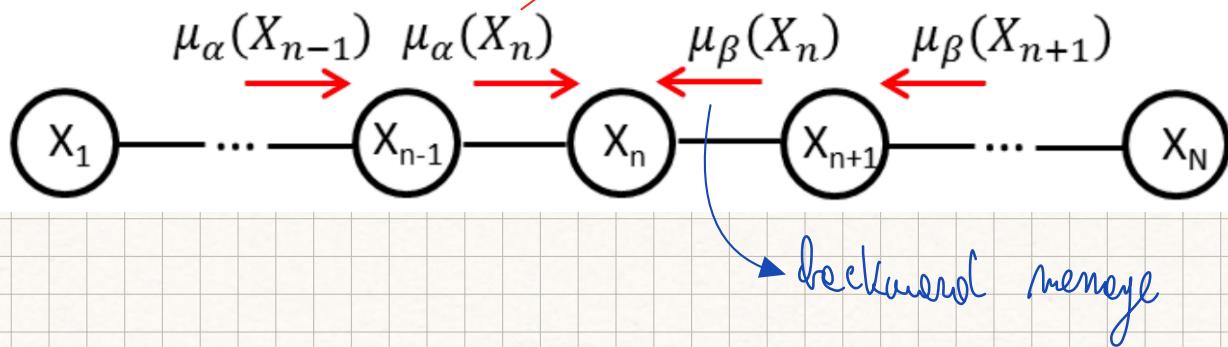
$$= P(y_1 | S_1 = j) \sum_{j=1}^c P(S_t = i | S_{t-1} = j) \lambda_{t-1}(j)$$

$\beta$ -term computed as part of backward recursion

$$\beta_T(j) = 1$$

$$\begin{aligned} \beta_T(j) &= P(Y_{t+1:T} \mid S_t = j) \approx \\ &\approx \sum_{i=1}^c p(S_{t+1} \mid S_t) \underbrace{P(Y_{t+1} \mid S_{t+1})}_{b_i(y_{t+1})} \beta_{t+1}(i) \end{aligned}$$

Sum - Product Menage Roring forward menage



$$\begin{aligned} \mu_\alpha(X_m) &= \sum_{\substack{x_{m-1} \\ \mathcal{A}_t(i)}} \psi(x_{m-1}, x_m) \mu_\alpha(x_{m-1}) \\ &\quad \sum_{j=1}^c b_i(y_t) A_{ij} \end{aligned}$$

$$\begin{aligned} \mu_\beta(X_m) &= \sum_{\substack{x_{m+1} \\ \mathcal{B}_t(j)}} \psi(x_m, x_{m+1}) \mu_\beta(x_{m+1}) \\ &\quad \sum_{i=1}^c b_i(y_{t+1}) A_{ij} \quad \beta_{t+1}(i) \end{aligned}$$

prior

emission

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

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

Expectation maximization algorithm: add indicator variables  $z$

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

1 only for 1  
 combination,  
 $z_{ti}^n$  0 for the others

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

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

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

Expectation - Maximization algorithm: maximize complete likelihood  $\lambda_c(\theta)$

E-Step: given the current estimate of the model parameters  $\theta^{(t)}$ ,

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

variable  $\leftarrow$  constant

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

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

Iterate 2-steps until  $|\hat{L}_c(\theta) - L_c(\theta) - 1| < \epsilon$

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

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

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

I can use forward-backward alg.

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

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

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

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

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

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:  $B_{ki} = \frac{\sum_{n=1}^N \sum_{t=1}^{T^n} P(S_t=i | \tilde{Y}^n) \mathbb{I}(Y_{it}=k)}{\sum_{n=1}^N \sum_{t=1}^{T^n} P(S_t=i | \tilde{Y}^n)}$

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

- 1) Identify single st that maximize posterior  $\pi_t^* = \arg \max_{i=1 \dots c} P(S_t=i | \tilde{Y})$
- 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 next-product memory passing algorithm:

$$\begin{aligned} & \max_{S_1 \dots S_T} \prod_{t=1}^T P(y_t | S_t) P(S_t | S_{t-1}) = \\ & = \max_{S_1 \dots S_{T-1}} \left( \prod_{t=1}^{T-1} P(y_t | S_t) P(S_t | S_{t-1}) \right) \underbrace{\max_{S_T} P(Y_T | S_T) \cdot P(S_T | S_{T-1})}_{E_T(S_{T-1})} \end{aligned}$$

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

$$E_{t-1}(S_{t-1}) = \max_{S_t} P(Y_t | S_t = s_t) P(S_t = s_t | S_{t-1} = s_{t-1}) E_t(s_t)$$

$$E_T(s_T) = 1 \quad 2 \leq t \leq T$$

Root optimal state:  $s_1^* = \arg \max_s P(Y_1 | S_1 = s) P(S_1 = s) E_1(s)$

Recursive forward optimal state:

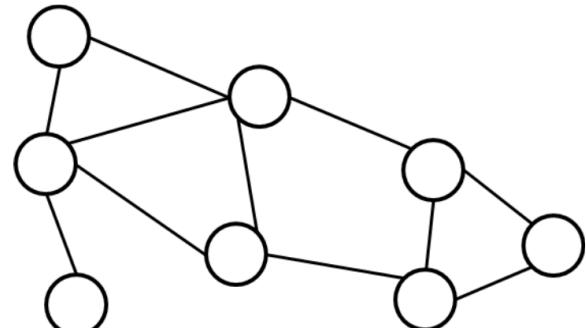
$$s_t^* = \arg \max_s P(Y_t | S_t = s) P(S_t = s | S_{t-1} = s_{t-1}^*) E_t(s)$$

# MARKOV RANDOM FIELD

## Markov Random Fields (MFRs)

- Undirected graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  (a.k.a. **Markov Networks**)  $\cup \mathcal{G}$
- Nodes**  $v \in \mathcal{V}$  represent **random variables**  $X_v$ 
  - Shaded  $\Rightarrow$  observed
  - Empty  $\Rightarrow$  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

$$P(x) = \frac{1}{Z} \prod_c \psi_c(x_c)$$

- $x_c \rightarrow$  RV of the node in the maximal clique  $c$
- $\psi_c(x_c) \rightarrow$  potential function for clique  $c$ , is not a probability
- $Z \rightarrow$  partition function ensuring normalization  $Z = \sum_{\text{over all states}} \prod_c \psi_c(x_c)$

express 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

→ number of feature functions

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

product of exponential

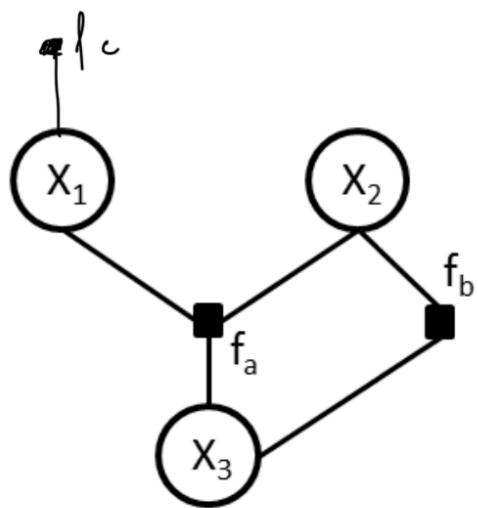
feature function to compute the potential parameter  $\in \mathbb{R}$

clique of RV

clique c

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

$$f_c(x_1) \approx P(x_1)$$



- 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.

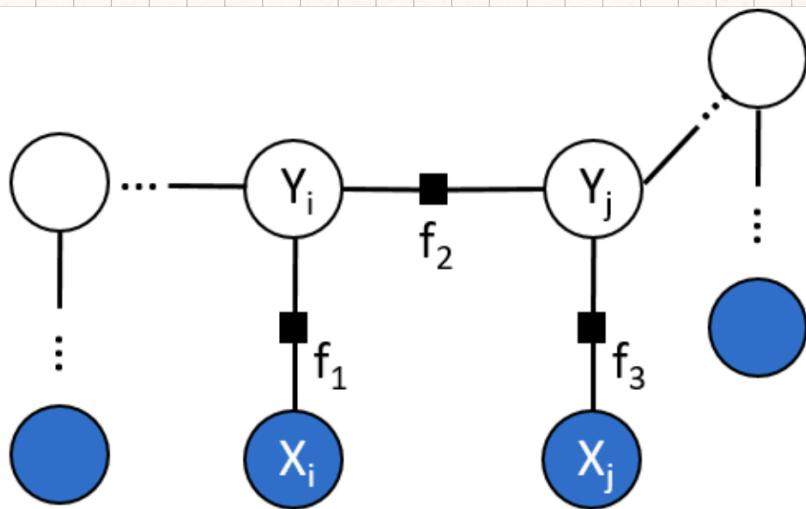
- $x_k$  - observable input in the factor  $k$
- $y_k$  - hidden (or partially observable) RV

•  $f_k(x_k, y_k)$  - 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

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

## CONDITIONAL RANDOM FIELDS



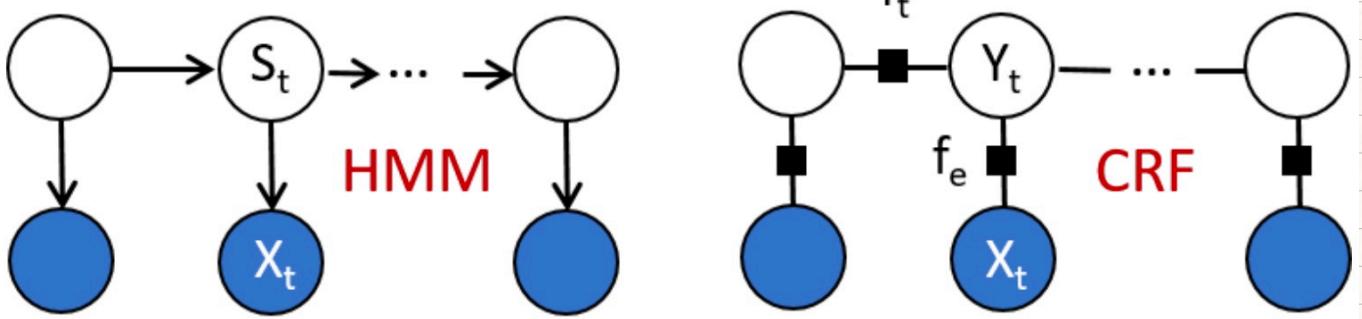
$$P(Y|X, \theta) = \frac{1}{Z(x)} \exp(\theta_1 f_1(X_i, Y_i) + \theta_2 f_2(Y_i, Y_j) + \theta_3 f_3(X_j, Y_j) + \dots)$$

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  $\theta$  in  $P(Y|X, \theta)$

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

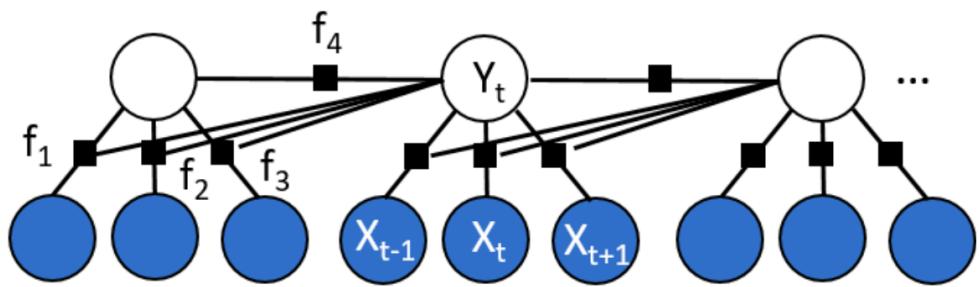
The undirected and discriminative equivalent of a HMM



$f_t(Y_{t-1}, Y_t)$  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  $f_e$  &  $f_t$  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:

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

*parameters*

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

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

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

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

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

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

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

$$\max_{\theta} L(\theta) = \max_{\theta} \sum_{n=1}^N \log \left( P(Y^n | X^n, \theta) P(\theta) \right) =$$

$$= \max_{\theta} \log \prod_n \frac{1}{Z(X^n)} \prod_t \prod_k \exp(\theta_k f_k(Y_t^n, Y_{t-1}^n, X_t^n)) =$$

$$= \sum_n \sum_t \sum_k \theta_k f_k(Y_t^n, Y_{t-1}^n, X_t^n) - \sum_n \log Z(X^n) =$$

↓ latent    ↓ time    ↓ feature  
 function

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

$$= \sum_n \sum_t \sum_k \theta_k f_k(Y_t^n, Y_{t-1}^n, X_t^n) - \sum_n \log Z(X^n) - \sum_k \frac{\theta_k^2}{2\sigma^2}$$

Penalization term

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

$\textcircled{B}$

$\textcircled{C}$

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

A: is  $E[f_K]$  under the empirical distribution

isn't count

B: is  $E[f_K]$  under model distribution

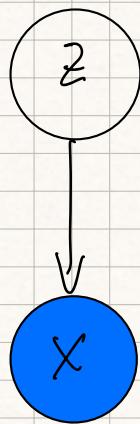
We learn  $\theta$  parameters by SGD:  $\theta^m = \theta^{m-1} - \nabla_{\theta} \mathcal{L}_m(\theta^{m-1})$

where  $\nabla_{\theta} \mathcal{L}_m(\theta) =$

$$= \sum_t f_K(Y_t, \tilde{Y}_{t-1}, X_t) - \sum_t \sum_{y,y'} f_K(y, y', X_t) P(y, y' | X_t) - \frac{\theta_K}{N \sigma^2}$$

Estimated by  
sum-product inference

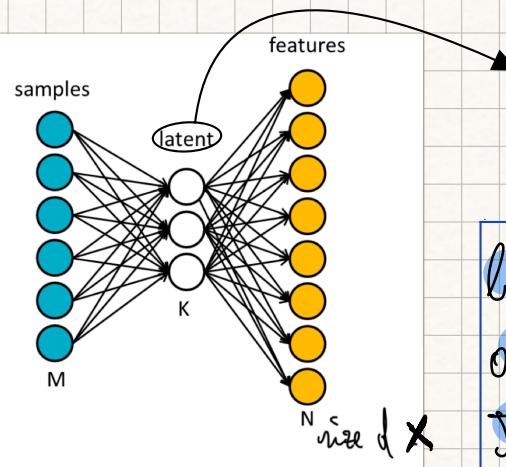
## Latent Variable Models:



Latent variables: unobserved RV that define a hidden generative process of observed data. Practically 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 distributions are more tractable than the joint distribution  $P(X)$   $K \ll N$

Kullback-Leibler (KL) Divergence: An information metric on how close two distributions  $P(z)$  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  $\geq 0$  because it's composed by logarithm of probabilities. Is equal to 0 iff the two distributions are the same.

- $Q$  high and  $P$  high  $\Rightarrow \hat{\cup}$
- $Q$  high and  $P$  low  $\Rightarrow \hat{\cap}$
- $Q$  low  $\Rightarrow$  don't care because expectation is calculated on  $Q$

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

Log-likelihood for a model with single hidden variable  $z$  and  $\theta$  parameters:

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

$\downarrow \neq 0$

$$\text{we can write: } \log P(x|\theta) = \log E_Q \left[ \frac{P(x,z)}{Q(z)} \right] \stackrel{\uparrow}{\geq} E_Q \left[ \log \frac{P(x,z)}{Q(z)} \right] =$$

Jensen

$$= E_Q [\log P(x,z)] - E_Q [\log Q(z)] = L(x, \theta, \phi)$$

pushing to  
overfitting the model

$E_Q [\log P(x,z)]$   
Expectation of joint distribution

$E_Q [\log Q(z)]$   
Entropy

pushing to  
regularization

let's see how good is  $L(x, \theta, \phi)$  or 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 ( $\int_z Q(z) = 1$ )

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

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

more appropriate obj function

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

- We can assume the existence of a probability  $Q(z|\phi)$  which allows to bound the likelihood  $P(x|\theta)$  from below using  $L(x, \theta, \phi) \rightarrow$  called evidence lower bound (ELBO)
- Optimal bound is  $KL(Q(z|\phi) || P(z|x, \theta)) = 0$ , then is  $Q(z|\phi) = P(z|x, \theta)$
- Minimizing KL is equivalent to maximize the ELBO  $\Rightarrow$  change a coupling problem with an optimization problem

Expectation maximization learning : maximum likelihood learning

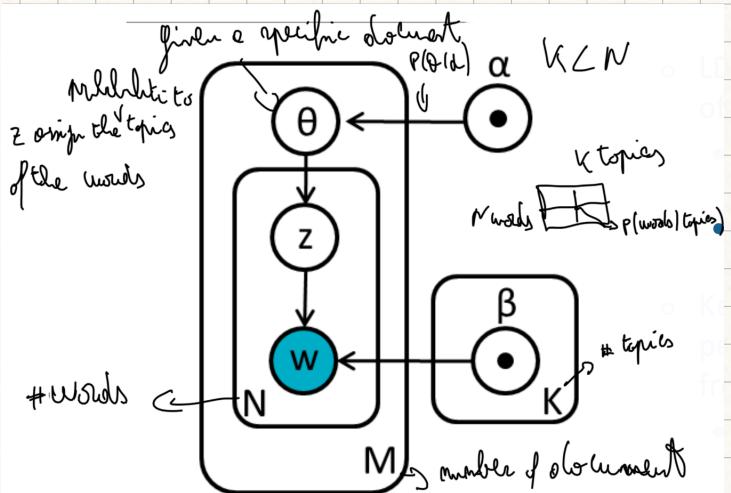
with hidden variables can be approached by maximisation of the ELBO :

$$\max_{\theta, \phi} \sum_{m=1}^N \ell(x_m, \theta, \phi)$$

where  $\theta$  are the model parameters and  $\phi$  are 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  $\phi$  that minimize  $KL(Q(z|\phi) || P(z|x, \theta))$  or maximize  $\ell(x, \theta, \phi)$

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  $P(z|\theta)$

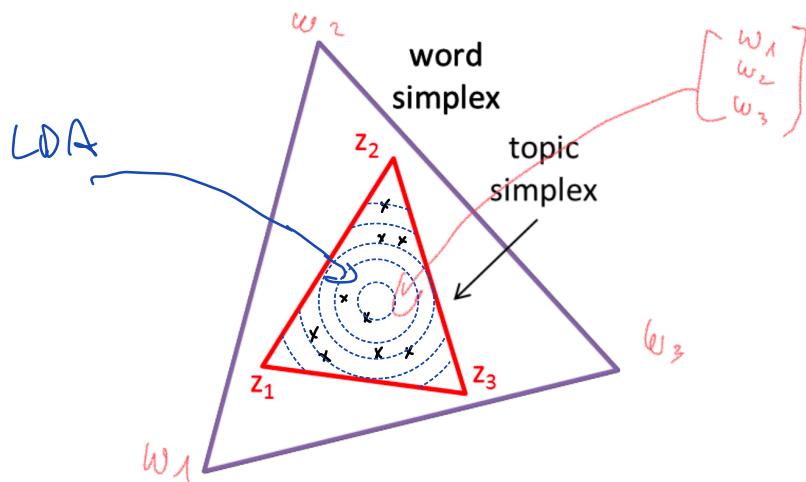
Each document has its personal topic proportion  $\theta$  sampled from a distribution.  $\theta$  defines a multinomial distribution but it is a random variable as well.

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

- $P(\theta | \lambda)$  Dirichlet distribution with hyperparameter  $\lambda$ . Returns  $K$  vectors that sum to 1

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

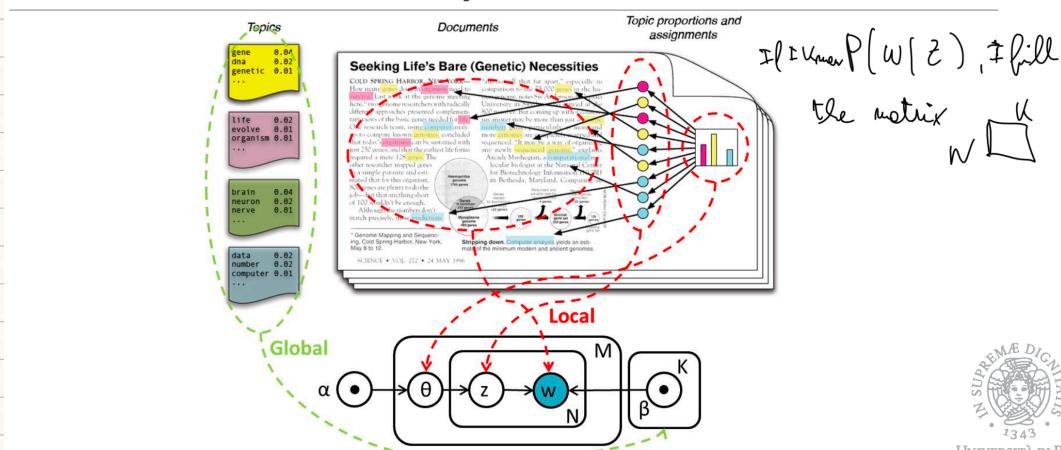
$\lambda_k$  is a prior count of the  $k$ -th topic. It controls the mean shape and sparsity of multinomial parameters  $\theta$ .



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

$\lambda$  controls the degree of sparsity of the state within the triangle  
High  $\lambda \Rightarrow$  uniform distribution. Small  $\lambda \Rightarrow$  impulse, one word where  $\alpha P=1$

## LDA and Text Analysis



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  $[\beta]_{K \times V}$

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

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

↑  
Dirichlet      ↑  
                    ↓  
                    Multinomial

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

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

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

Key problem is inferring latent (unobserved) variables posterior:

$$P(A|B, \Theta) = \frac{P(A \cap B \cap \Theta)}{P(B \cap \Theta)}$$

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

Learning with hidden variables corresponds to expectation-maximization

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

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

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

**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  $\beta$  and  $\theta$  are no longer coupled. Fit  $\phi$  parameter so that  $Q(z|\phi)$  is close to  $P(w | \lambda, \beta)$  according to KL divergence.

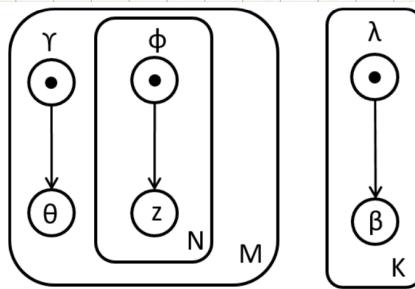
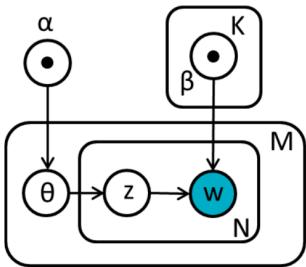
**Sampling approach:** construct Markov chain on the hidden variables whose limiting distribution is the posterior

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

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

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

## VARIATIONAL LDA DISTRIBUTION:



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

$$Q(\theta, z, \beta | \Phi) = Q(\theta | \gamma) \prod_{m=1}^N Q(z_m | \phi_m) \prod_{k=1}^K Q(\beta_k | \lambda_k)$$

the model parameter  $\Psi = \{z, \beta\}$  of a sample distribution  $P(\theta, z, w | \vartheta, \beta) = P(\theta, z, w | \Psi)$

find the  $\phi, \psi$  that maximize the ELBO

$$\mathcal{L}(w, \Phi, \Psi) = E_Q[\log P(\theta, z, w | \Psi)] - E_Q[\log Q(\theta, z, \beta | \Phi)]$$

by alternate maximization:

while (little likelihood improvement):

fix  $\Psi$ : update variational parameters  $\Phi^*$  (E-step)

fix  $\Phi = \Phi^*$ : update model parameter  $\psi^*$  (M-step)

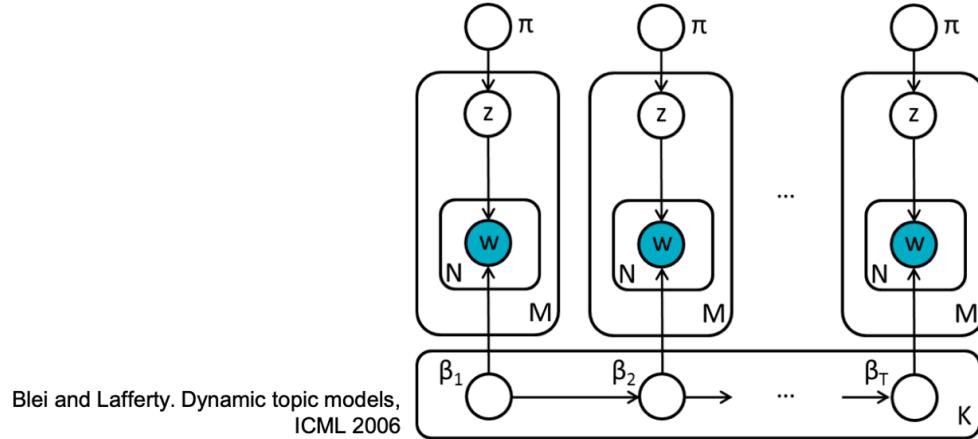
Unlike EM, variational EM doesn't guarantee to reach local maxima of  $\mathcal{L}$

LDA Applications: model for collections of high-dimension vectors where 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



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

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

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

Boltzmann machine

$P(\theta, z | w, d, \beta)$  LDA

posterior when non-conjugate

priors are used.

For 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:  $\theta, z, \beta \sim P(\theta, z, \beta | w, d)$ .

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

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

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

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

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

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

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

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

if we assume:  $\tilde{P}(x^e) \approx P(x^e)$  same marginals (valid simpler)

then approx. variance  $\tilde{P}(x^e, x^{e'}) = \tilde{P}(x^e) \tilde{P}(x^{e'})$  samples independence

we obtain:  $E_{\tilde{P}(x)} \left[ (\hat{f}_x - E_{\tilde{P}(x)}[\hat{f}_x])^2 \right] = \frac{1}{n} \text{Var}_{P(x)}[f(x)]$

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

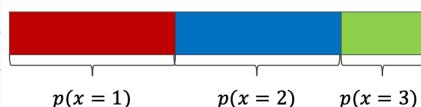
probability to obtain a sample set  $X$       probability of the (unknown) random variable  
 $P(X) \neq P(x)$

$P(x)$ : distribution for sample. Doesn't depend on the sample

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

Univariate resampling: just need a random number generator that computes a value uniformly 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  $C^n$  states. We can use the univariate approach, but  $O(C^n)$  states are computationally infeasible.

We can rewrite the joint distribution with the chain rule:

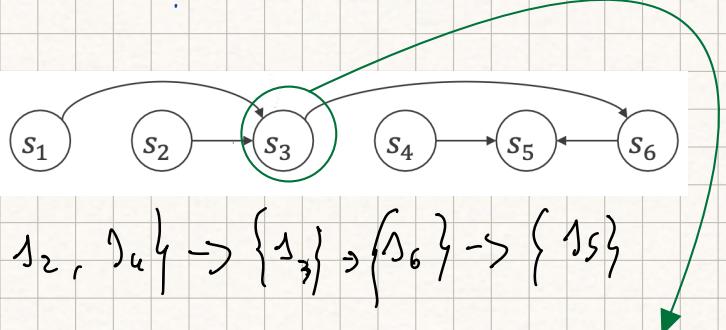
$$p(s_1, \dots, s_n) = p(s_1) \cdot p(s_2 | s_1) \cdot p(s_3 | s_2, s_1) \cdots p(s_n | s_{n-1}, \dots, s_1)$$

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  $\tilde{s}_3 \sim p(s_3 | \tilde{s}_1, \tilde{s}_2)$  / ANCESTRAL SAMPLING
- $\vdots$
- 4 //  $\tilde{s}_n \sim p(s_n | \tilde{s}_1, \dots, \tilde{s}_{n-1})$

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

If I knew the belief network,



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

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

V-shape: if I drew  $s_3$ ,  $s_1$  and  $s_2$  become conditional dependent.

AS guarantees 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:  $S = S_E \cup S_{\text{e}} ; P(S_{\text{e}} | S_E) = \frac{P(S_E, S_{\text{e}})}{P(S_E)}$

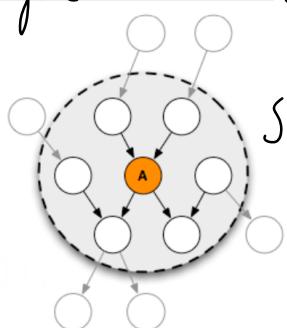
If we run AS we discount the fact that  $\Rightarrow$  V-shape: if I observe  $S_3$ ,  $S_1$  and  $S_2$  become conditional dependent  $\Rightarrow$  no more independent

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:



$$s_j^{l+1} \sim P(s_j | S_{\setminus j}) = \frac{1}{Z} P(s_j | p_0(s_j)) \prod_{k \in \text{ch}(j)} P(s_k | p_0(s_k))$$

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  $\{z_{i,j}^0, \theta^0, \beta^0\}$

$$1 z_{i,j}^{l+1} \sim P(z_{i,j} | w, z_{i,j}^{l-1}, \theta^l, \beta^l, \lambda)$$

$$2 \theta_i^{l+1} \sim P(\theta_i | w, z^{l+1}, \beta^l, \lambda) \quad \text{learning part}$$

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

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

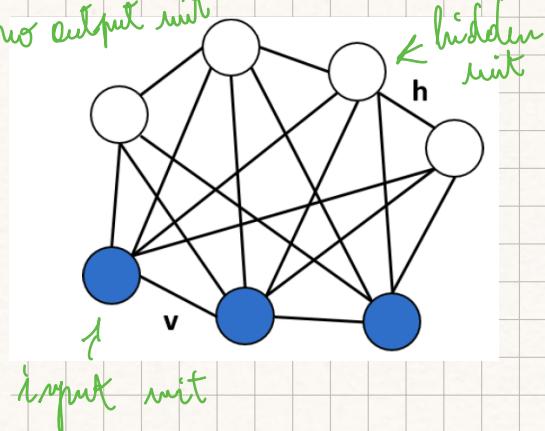
The Gibbs sampling draws a new sample  $x^l$  from  $q(x^l | x^{l-1})$ , we are not resampling from  $p(x)$ ! If we compute the limit to  $l \rightarrow \infty$ , the series  $\{x_1, \dots, x_l\}$  converges to sample taken from  $p(x)$

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

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

Hence, the MC has a stationary distribution. There are different  $q()$  which converge to  $p()$ .

Boltzmann Machines: An example of Markov random field



- visible RV  $v \in \{0,1\}$  • latent RV  $h \in \{0,1\}$

- $s = [v, h] \rightarrow$  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" ↑ "bias"

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 neurons with binary output  $s_j$ . At each time interval ( $t+1 \equiv t + \Delta t$ ), the neuron can emit a spike signal with

probability  $P_j^{(t)}$

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

$$P_j^{(t)} \approx \sigma(x_j^{(t)}) \quad \text{function of local potential } x_j$$

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

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

a chosen neuron is active with probability:

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

Let's suppose that we update each neurons in parallel (every  $\Delta t$ )

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

yielding a Markov process for state update

$$P(s^{(t+1)} = s') = \sum_s T(s' | s) P(s^{(t)} = s)$$

**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:  $P_{\text{eq}}(s) = \frac{e^{-E(s)}}{Z}$

$\rightarrow$  energy function  
 $\rightarrow$  partition function ensure  $\sum_s P(s) = 1$

Learning of a BM: Boltzmann machines can be trained so that the equilibrium distribution  $P_{\text{eq}}(s)$  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

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

# details  $\leftarrow$  parameters

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

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

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

then the log-likelihood

$$\frac{\partial L}{\partial M_{ij}} = -\langle v_i v_j \rangle + \underbrace{\langle v_i v_j \rangle_c}_{\text{clamped expectation}}$$

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

$$\text{Hebbian learning: } \frac{\partial L}{\partial M_{ij}} = \underbrace{\langle v_i v_j \rangle_c}_{\text{wolve}} - \underbrace{\langle v_i v_j \rangle}_{\text{dream}}$$

wolve part is the standard Hebb rule applied to the empirical distribution of state that the machine sees 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

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

$$\log P(V^l | M) = -\mathbb{E}(V^l) - \log \tau \xrightarrow{\textcircled{1}} \frac{\partial \log l(x)}{\partial x} = \frac{l'(x)}{l(x)}$$

$\Downarrow$

$$+ V^T M V \Rightarrow \sum_{p,q} M_{pq} V_q V_p$$

$$\frac{\partial \log P(V^l | M)}{\partial M_{ij}} = V_i^l V_j^l - \sum_v \underbrace{\frac{\exp}{Z} \cdot V_i^l V_j^l}_{P(V|n)} = V_i^l V_j^l - \sum_v P(V|n) V_i^l V_j^l$$

$E[V_i^l V_j^l]$   
 $V \sim P(V|n)$   
 $\langle V_i^l V_j^l \rangle_n$

$$= V_i^l V_j^l - \langle V_i^l V_j^l \rangle_M$$

$$\boxed{\frac{1}{L} \sum_i (V_i^l V_j^l)} - \langle V_i^l V_j^l \rangle_M$$

$$E_{V \sim \text{Data}} [V_i^l V_j^l]$$

$$\langle V_i^l V_j^l \rangle_0$$

$$V_i^l, V_j^l \in \{0, 1\}$$

$$\Delta M_{ij} = \frac{\partial \log P}{\partial M_{ij}} = \langle V_i^l V_j^l \rangle_0 - \langle V_i^l V_j^l \rangle_M = 0$$

✓

a matching of two expectation  
correlation of the variable w.r.t. data & model should match

$$M'_{ij} = n_{ij} + \Delta M_{ij} \quad \text{bellman learning}$$

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

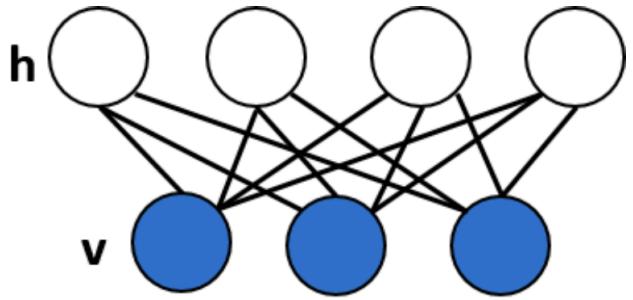
Learning with hidden variable: this time  $s = [v, h]$

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

Expectation is introcetible due to the partition function  $Z$

Restricted Boltzmann machines

connection only between hidden  
and visible variable.



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

bijterte graph

hidden units are conditionally independent given visible units  
and viceversa

$$P(h_j | v) = \sigma \left( \sum_i n_{ij} v_i + c_j \right) \quad \begin{cases} \text{update in} \\ \text{batch} \end{cases}$$
$$P(v_i | h) = \sigma \left( \sum_j n_{ij} h_j + b_i \right)$$

Training RBM:

$$\frac{\partial \mathcal{L}}{\partial M_{ij}} = \langle v_i h_j \rangle_c - \langle v_i h_j \rangle_m$$

• Gibbs sampling approach:

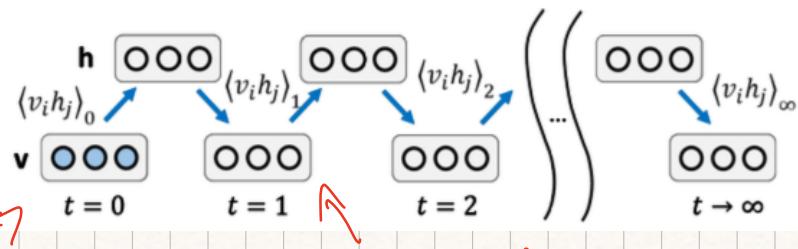
Wet part: clamp state on  $v$

- sample  $\langle v_i h_j \rangle$  for all pairs of connected units
- repeat for all elements of dataset

Dream part: • don't clamp units

- let network reach equilibrium
- sample  $\langle v_i h_j \rangle$  for all pairs of connected units
- repeat many times to get a good estimate

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



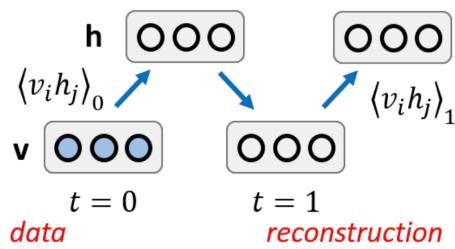
Plugging-in the data

- 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

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

Gibbs sampling is slow to converge (high variance)

Contrasting divergence learning: approximation of the gradient of the



1. Clamp a training vector  $v^l$  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

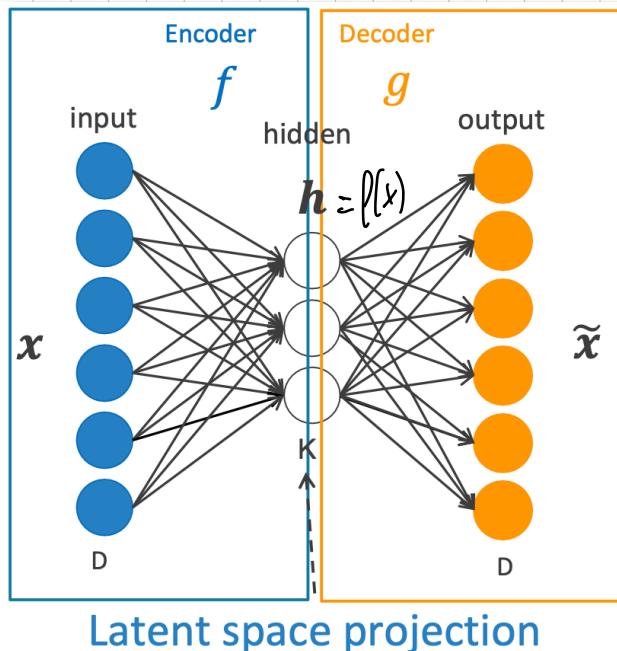
$$\underbrace{\langle v_i h_j \rangle_0}_{\text{data}} - \underbrace{\langle v_i h_j \rangle_1}_{\text{reconstruction}}$$



M A N C H A N O L G

C N N

## Autoencoder:



train a model to reconstruct the input data.  $K \ll D$  or  $h$  can be sparsely active  
 $L(x, \tilde{x}) = L(x, g(f(x)))$ . **On minimization**

In general we train nonlinear AF (ReLU, sigmoid), with  $K \leq D$ , 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

$$J_{\text{JAEG}}(\theta) = \sum_{x \in S} (L(x, \tilde{x}) + \lambda \mathcal{R}(h))$$

$$\mathcal{R}(h) = \mathcal{R}(f(x)) = \sum_j |h_j(x)|$$

$\rightarrow$  penalize activation function, not weight of the AF

Training with regularization corresponds to MAP inference

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

↓  
 likelihood

$$P(h) = \frac{\lambda}{2} \exp\left(-\sum_j \frac{1}{2} |h|_{l_1}\right)$$

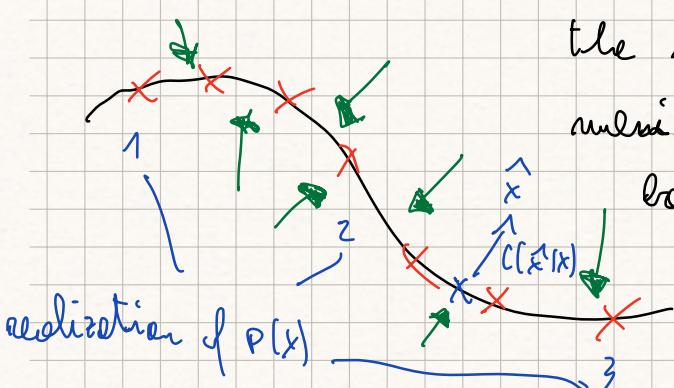
↓ PROR

$$S(h) = \|h\|_{l_1}$$

Denoising Autoencoder (DAE): train to minimize  $L(x, g(f(\tilde{x}))$ )

where  $\tilde{x} = x + \epsilon$ ,  $\epsilon \sim N(0, \sigma)$ . The key point 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 vector field (green arrow) that approximate the gradient of the unknown data generation distribution

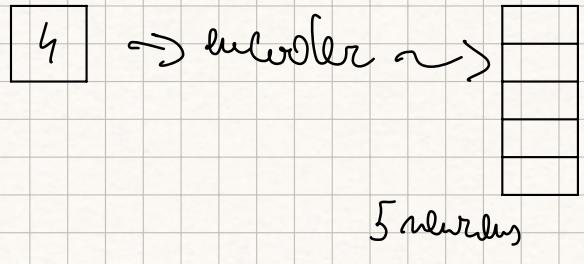
$$C(\tilde{x}|x) = N(\tilde{x}|x, \sigma^2)$$

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

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

move closer the data to the manifold  $\Rightarrow$  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 meaningful 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

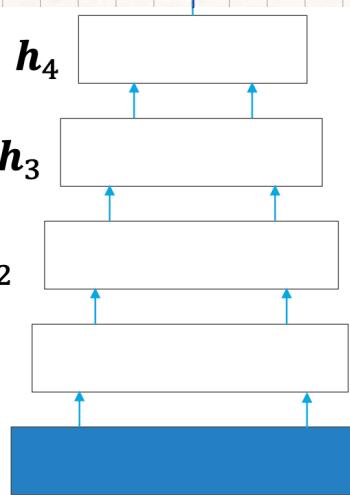
$$J_{\text{CAE}}(\theta) = \sum_{x \in S} (L(x, \hat{x}) + \lambda R(h))$$

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

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

Deep Autoencoders:

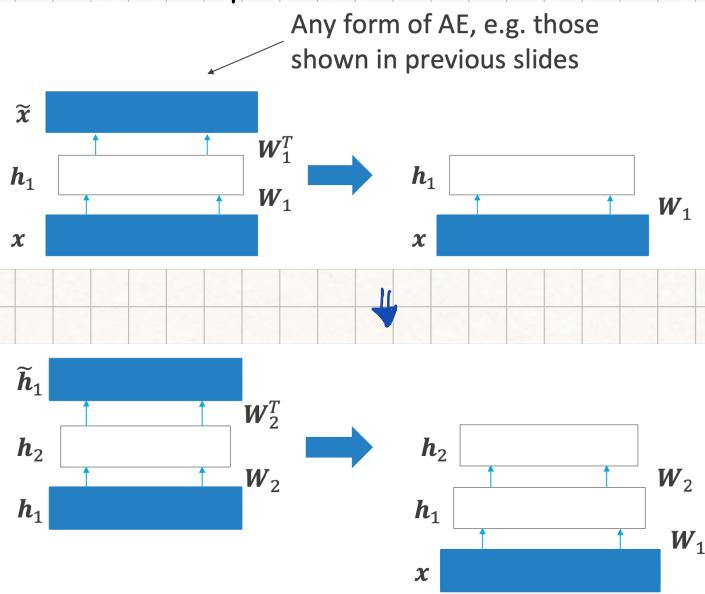
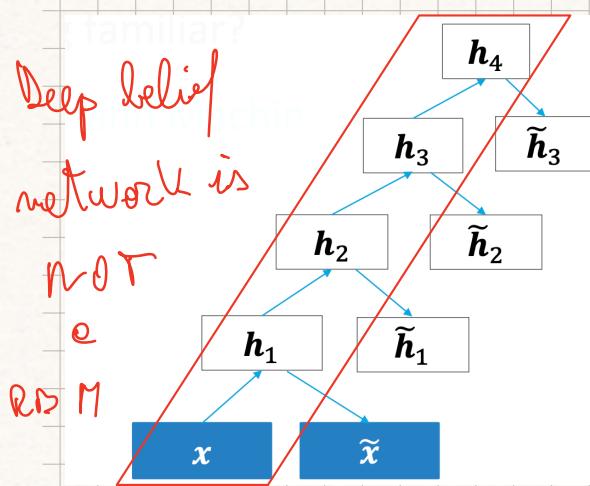
 → Supervised learning



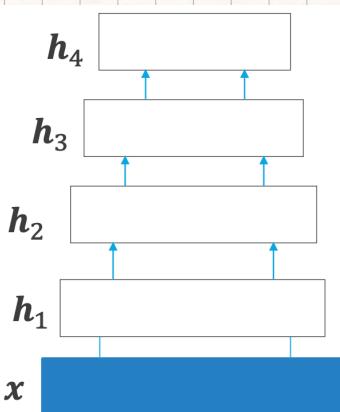
We can stack several autoencoders such that we reduce the input dimension. Useful for:  
Data visualization, exploration, indexing...

→ Unsupervised learning: incremental unsupervised construction of DeepAE

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

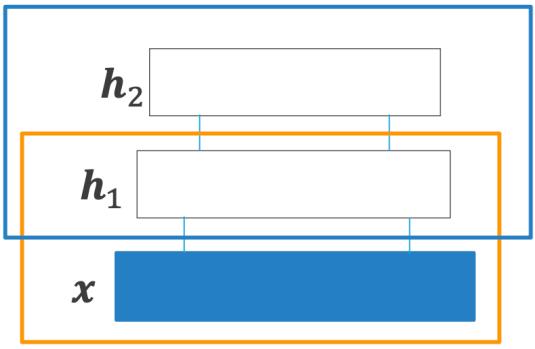


Deep Boltzmann Machine: DBN but without error directions



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

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



1) Pretraining the first layer entails fitting

$$\text{this model: } P(x|\theta) = \sum_{h^1} P(h^1|W) \underbrace{P(x|h^1, w^1)}_{\downarrow}$$

$$P(h^1|W) = \sum_x P(h^1, x|W)$$

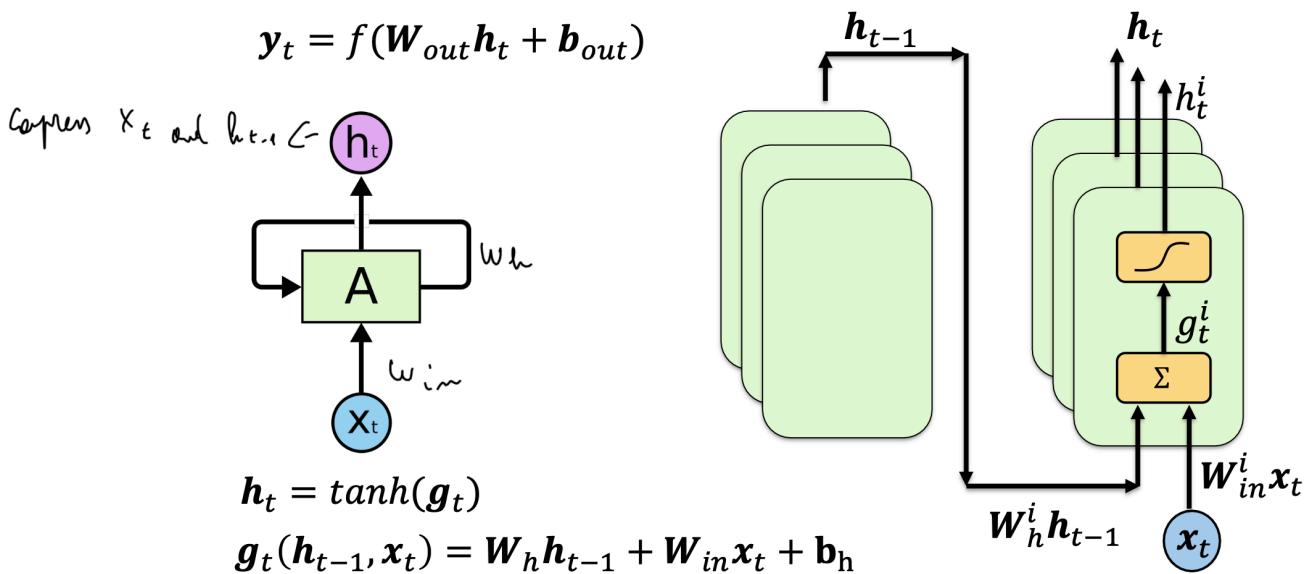
2) Pretraining the second layer changes  $h_1$  prior

$$\text{by: } P(h^1|W^2) = \sum_{h_2} P(h^1, h^2|W^2)$$

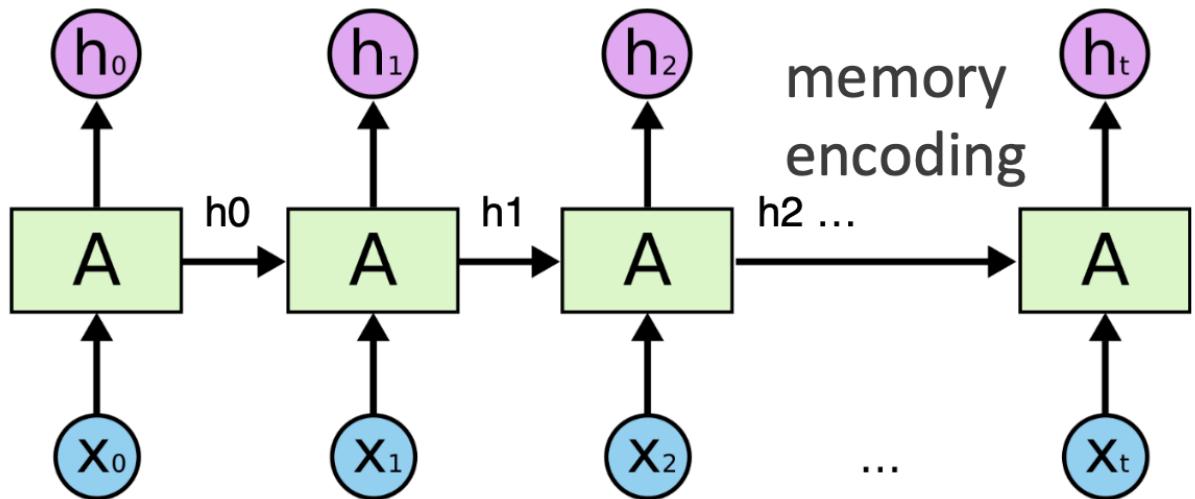
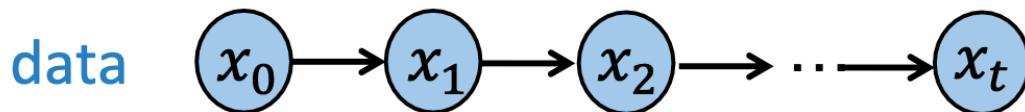
When putting things together we need to average between  
 $P(h^1|W)$  and  $P(h^1|W^2)$

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

RNN



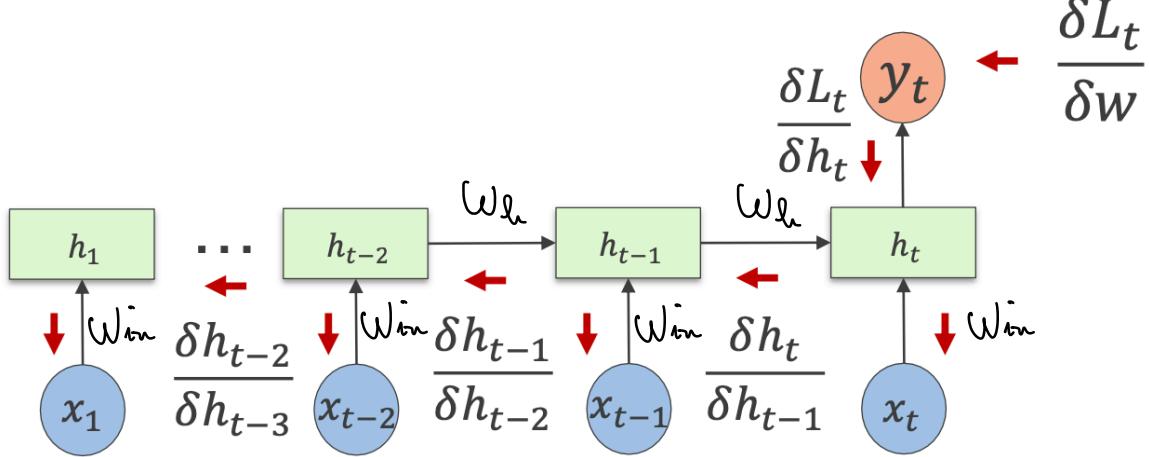
Unfolding the sequence



learning long term dependencies is difficult due to gradient vanishing/exploding  $\Rightarrow$  instability and oscillations

↓

no learning



$$\frac{\delta L_t}{\delta w} = \sum_{k=1}^t \frac{\delta L_t}{\delta h_k} \begin{matrix} \frac{\delta h_k}{\delta h_{k+1}} \\ \vdots \\ \frac{\delta h_1}{\delta h_2} \end{matrix} \rightarrow \text{Jacobian matrix}$$

$$\frac{\delta h_k}{\delta w}$$

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

$$\frac{\delta L_t}{\delta w} = \sum_{k=1}^t \frac{\delta L_t}{\delta h_k} \left( \prod_{l=k}^{t-1} \frac{\delta h_{l+1}}{\delta h_l} \right) \frac{\delta h_k}{\delta w}$$

given  $h_t = \tanh(W_{in}x_t + W_{hi}h_{t-1})$   $\tanh' = \text{sech}^2 = 1 - \tanh^2$

$$\frac{\delta h_{l+1}}{\delta h_l} = D_{l+1} W_{hl}^T$$

$$D_{l+1} = \text{diag}(1 - \tanh^2(W_{in}x_l + W_{hi}h_{l-1}))$$

$$\frac{\delta L_t}{\delta h_k} = \frac{\delta L_t}{\delta h_t} \cdot \left( \prod_{l=k}^{t-1} \frac{\delta h_{l+1}}{\delta h_l} \right) = \frac{\delta L_t}{\delta h_t} \prod_{l=k}^{t-1} D_{l+1} W_{hl}^T$$

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

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

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

$\sigma < 1$  vanishing  
 $\sigma > 1$  exploding

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

$\sigma = 1$  good solution

Gradient clipping (for gradient exploding)

$$g = \frac{\partial L_t}{\partial w} \quad \text{if } \|g\| \geq \theta_0 \text{ then } g = \frac{\theta_0}{\|g\|} \cdot g$$

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

$$\sigma = 1 \text{ for } W^T w = I \text{ otherwise}$$

$$W^H w = I$$

$$w = I$$

Orthogonal matrix + linear activation

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

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

(CEC) Content 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

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

$$\begin{matrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{matrix} \cdot \begin{matrix} f_t \\ h_{t-1} \end{matrix}$$

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

• Avoid saturation, slightly forgets the past, no guarantees about content propagation

# LSTM cell

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

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

logistic sigmoid

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

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

logistic sigmoid

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

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

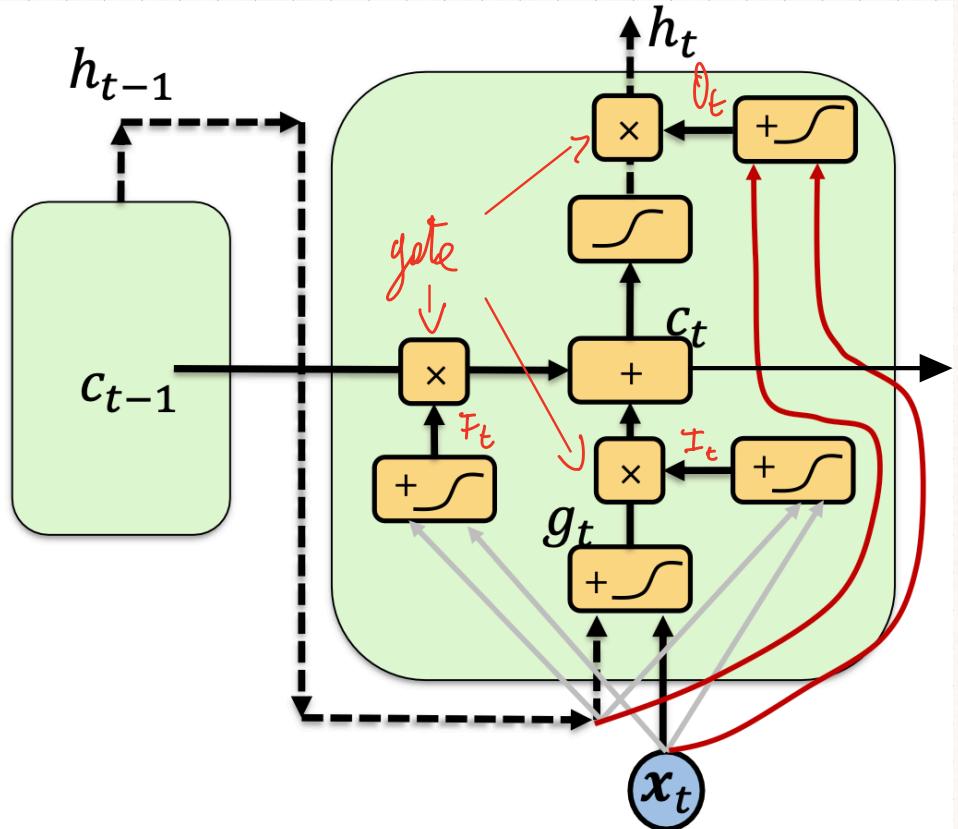
logistic sigmoid

Equations:

- Compute activation of input and forget gates

$$I_t = \sigma (W_{ih} h_{t-1} + W_{ix} x_t + b_i)$$

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



2) Compute input potential and internal state

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

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

3) Compute output gate and output state

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

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

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 committee effect (it's like having more LSTM working together). Need to adapt prediction phase.

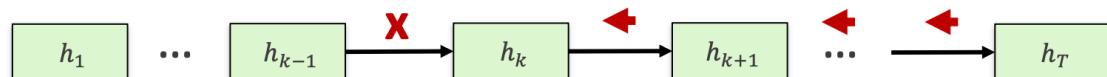
Useful for obtaining confidence interval for a given input.

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.



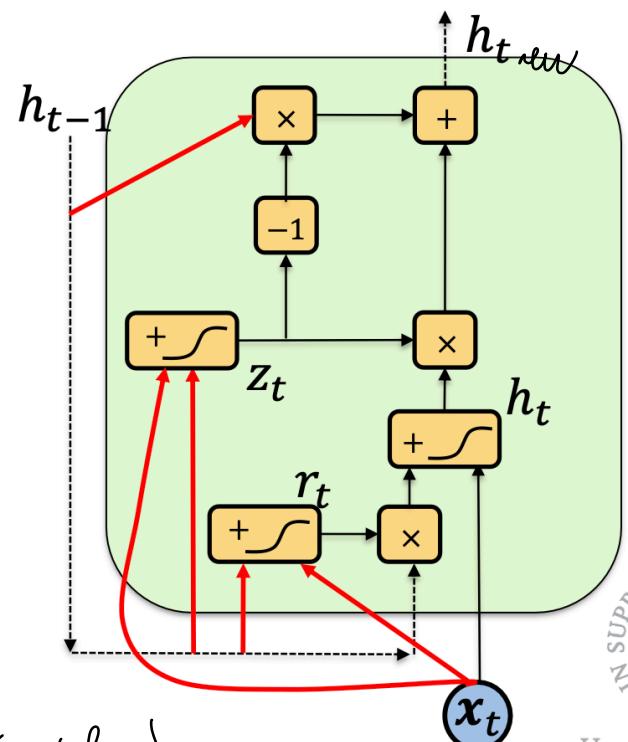
(GRU) Gated recurrent unit

Reset gate acts directly on output

. It controls how much

of the previous state we might still

want to remember.  $r_t$  controls it



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

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.

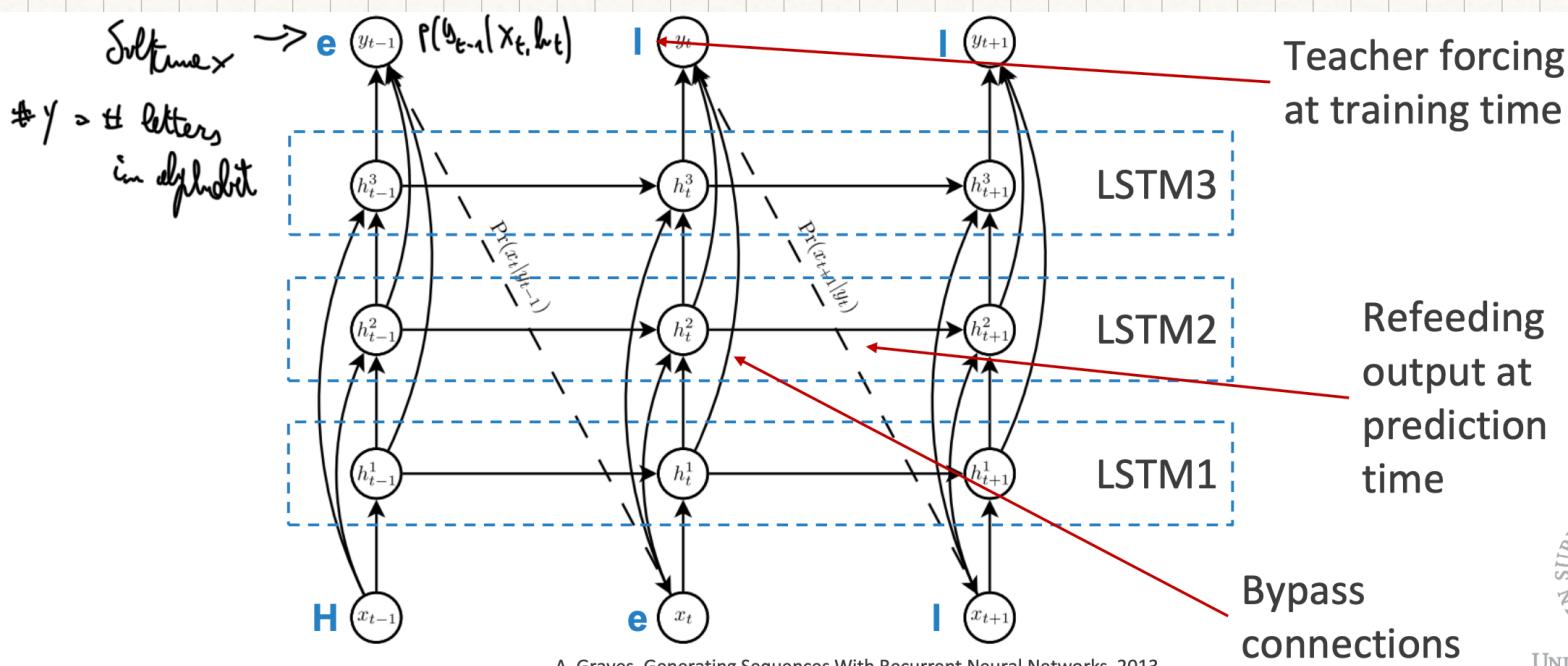
$$z_t = \sigma(W_{zh}h_{t-1} + W_{zin}x_t + b_z)$$

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

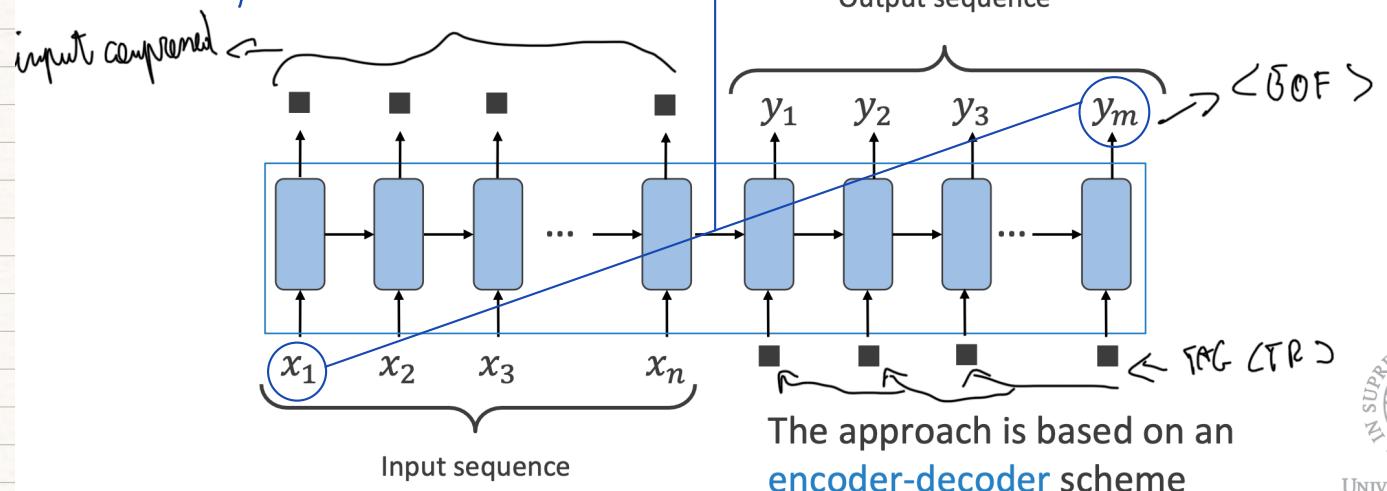
The main difference with LSTM is that a single gating unit simultaneously controls the forgetting factor and the decision to update the output unit.

## LSTM language model

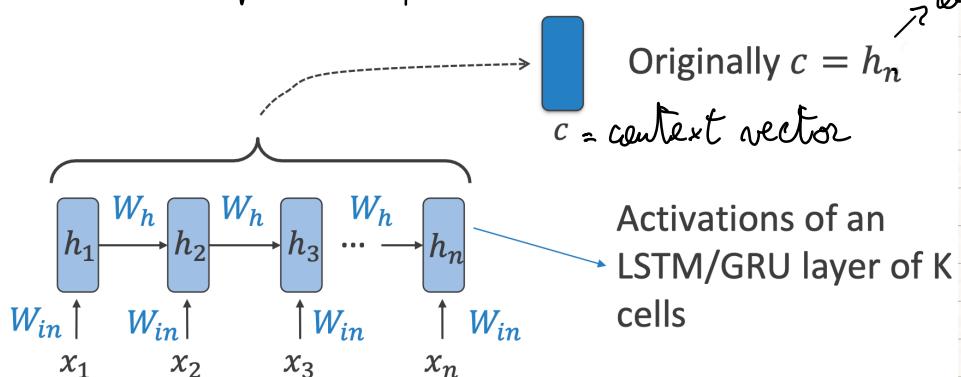
teacher forcing: used at the early phase of training (like simulating annealing). During the training you will reuse the predicted output (probably a wrong prediction) in order to give the model the capability to recover from error.



Sequence to sequence task

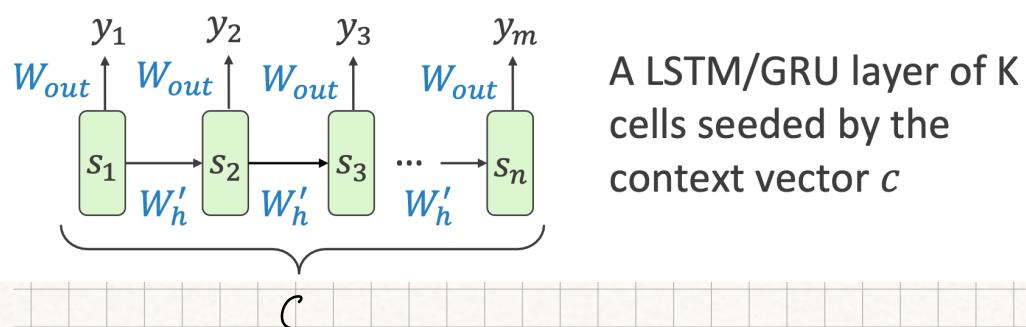


Encoder: produce a compressed and fixed length representation  $c$  of all the input sequence  $x_1, \dots, x_n$



? output for the last token of the sequence

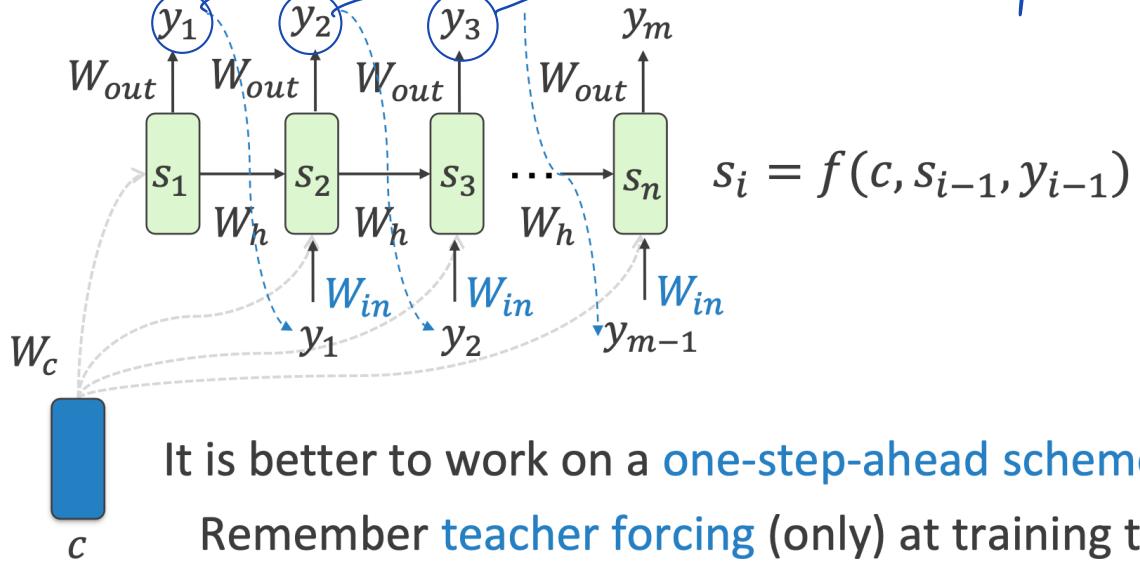
Decoder: starting from the context vector  $c$ , it generates the



output sequences

How to pass to the decoder?

Teacher forcing



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

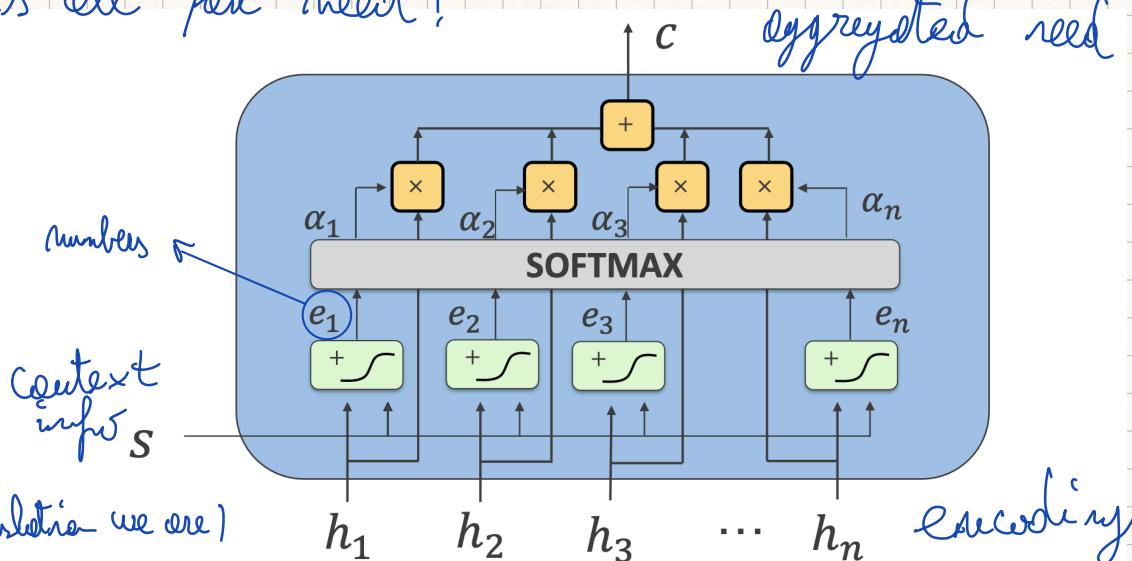
Remember **teacher forcing** (only) at training time

$c$  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 omits the hidden activation of the  
last input element memorizes sufficient information to generate  
the outputs (Bias toward the most recent part). We need attention

Attention is all you need!



(e.g. at which  
point of the translation we are)

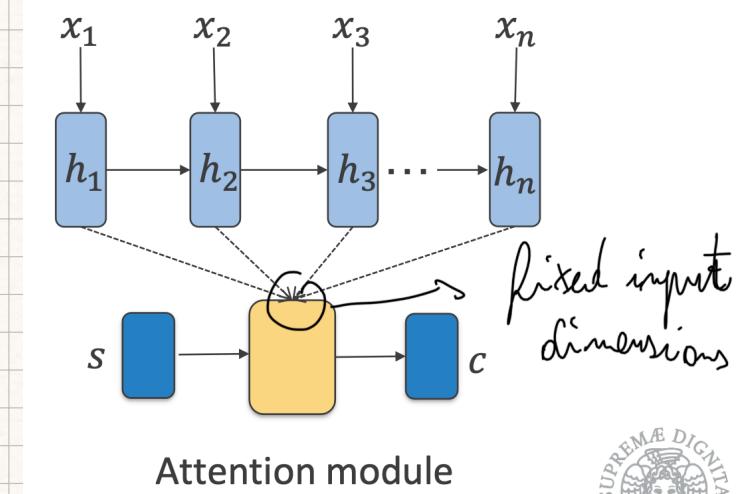
encodings

## RELEVANCE

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

SOFTMAX: differentiable max selector operator  $\lambda_i = \frac{\exp(l_i)}{\sum_j \exp(l_j)}$   
 $\sum_i \lambda_i = 1$  Convex combination

VOTING: aggregated seed by (soft) attention voting  $c = \sum_i \lambda_i h_i$

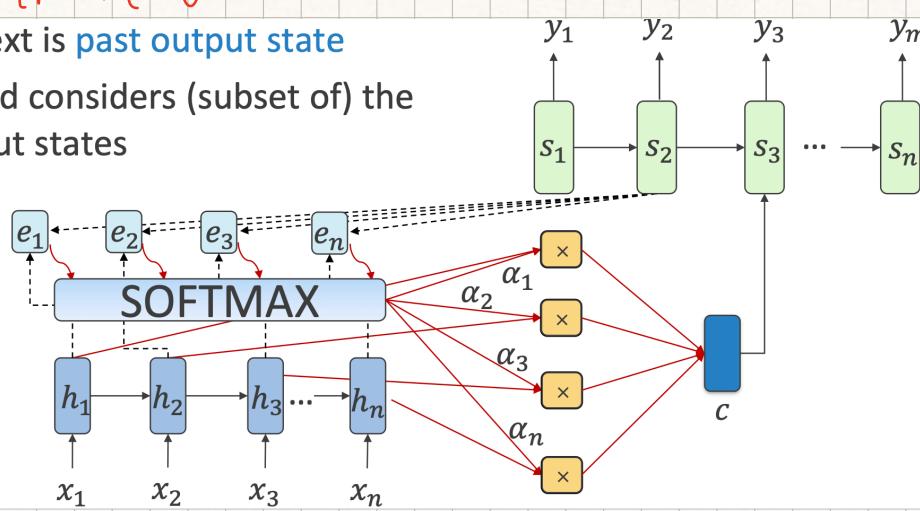


The attention module has a fixed input dimension

## ATTENTION IN SOFT SOFT

Context is past output state

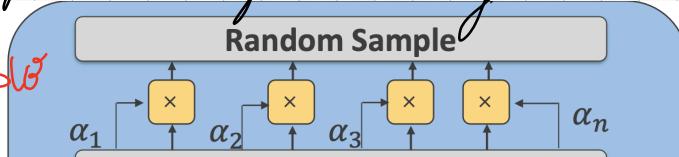
Seed considers (subset of) the input states



context  $s$  is the post output state

Word attention: sample a single encoding  $h_i$  using probability distribution

NOT DIFFERENTIABLE BUT



# 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

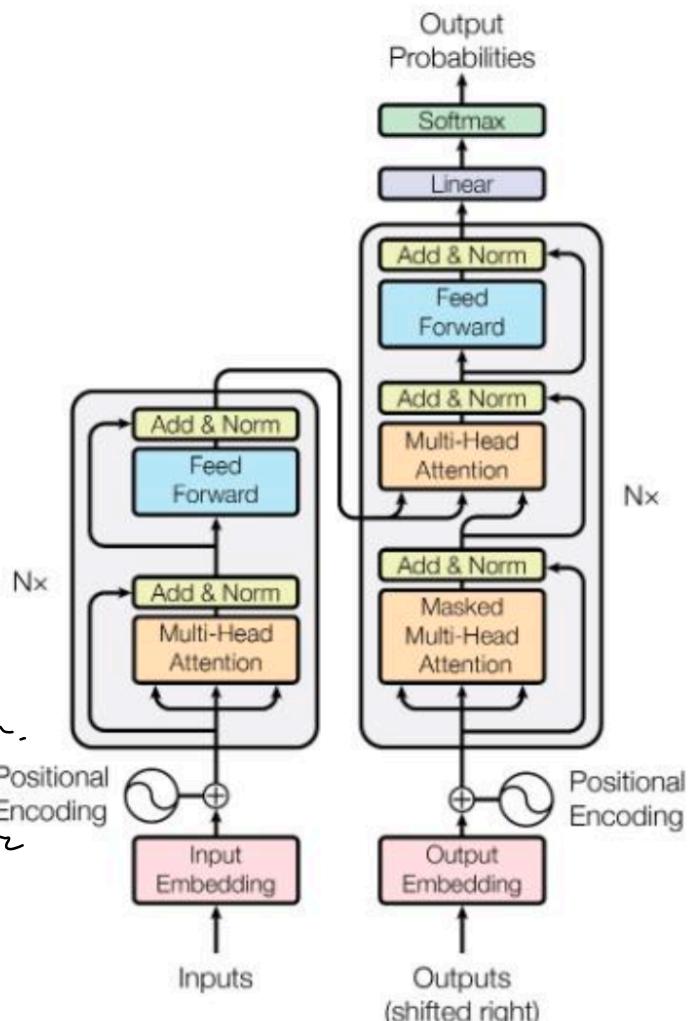
: query weights  $W_q$ , key weights  $W_k$

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:

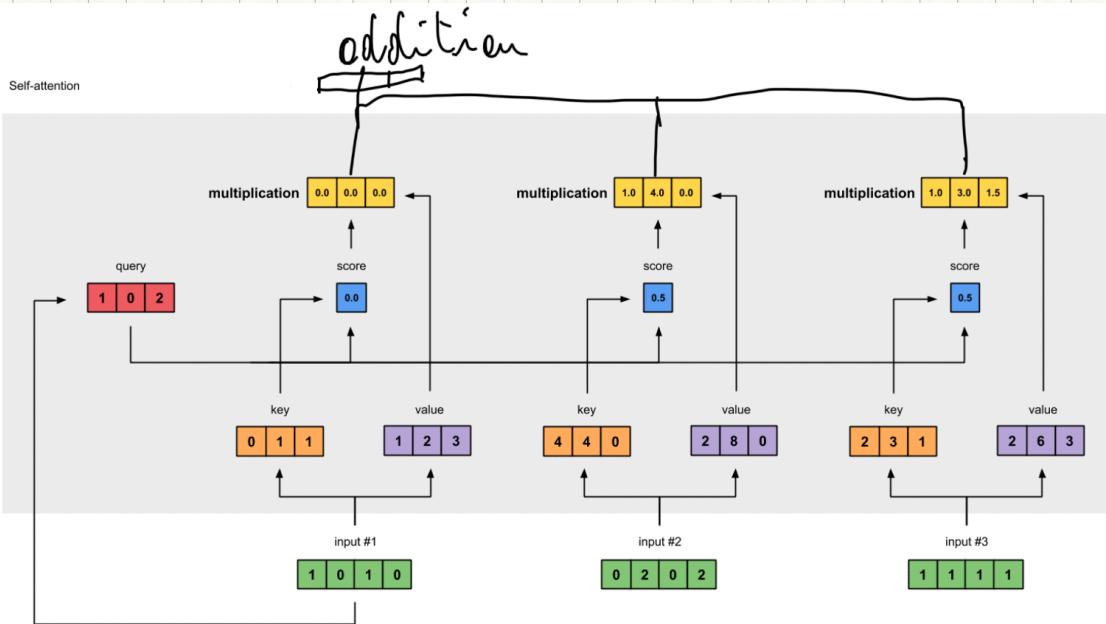
Query vector  $q_i = x_i W_q$ , Key vector  $k_i = x_i W_k$ , Value vector  $v_i = x_i W_v$

Attention weights  $Q_{ij}$  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  $e_{ij}$   
 (attention from token  $i$  to token  $j$ ,  $H_j$ ).



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

$Q \approx S$   
 i-th query vector element

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.

$\text{MultiheadAttention}(Q, K, V) = \text{Concat}(\text{Attention}(QW_i^Q, KW_i^K, VW_i^V))W_o$   
 final  
 projection matrix for the  
 multi-head attention

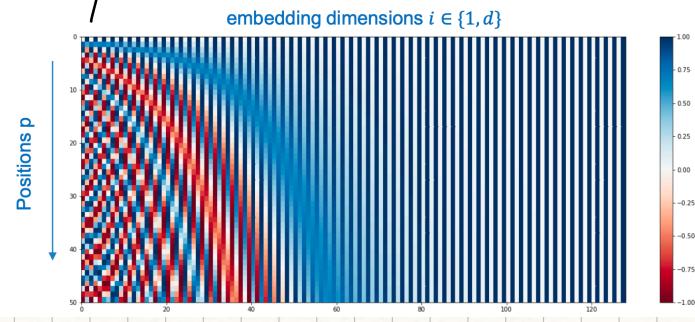
**Masked attention:** It may be necessary to cut out attention links between same word-pairs. In the decoder we would like to cut connection between token at position  $t$  and  $t+1$ . So we add a mask matrix  $M$  before softmax, such that there are negative infinity at entries where the attention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

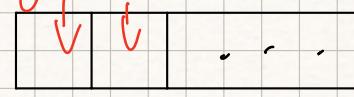
**Positional encoding:** the positional information is necessary for the transformer to make use of the order of the sequence. A positional encoding is a fixed-size vector representation that encapsulates the relative positions of tokens within a target sequence: it provides the transformer with information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

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

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

frequency





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

$P_B$  is a matrix  $[rep\_dim, d]$

$P$  is the position of the token  
in the rep.

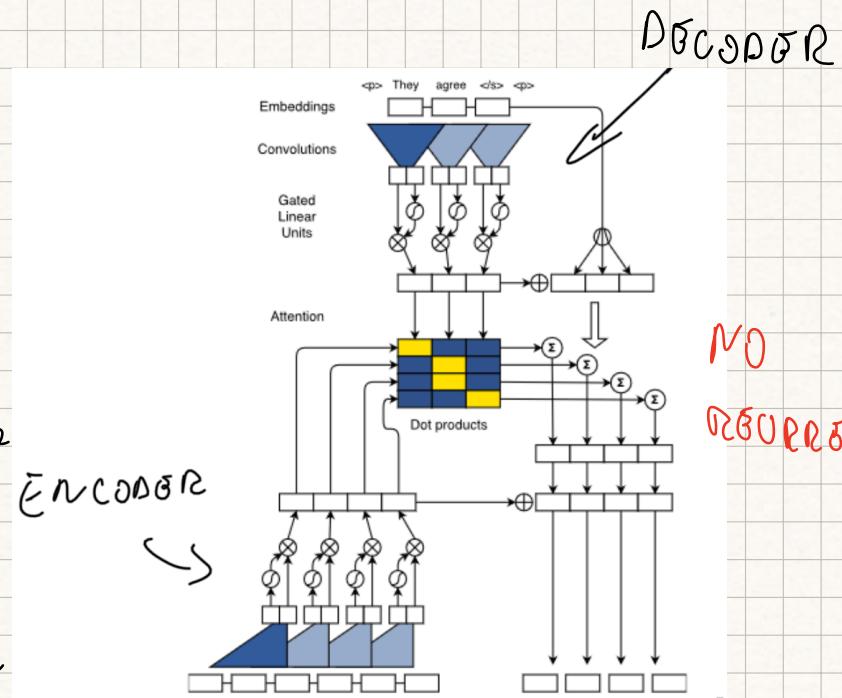
dimension of the output  
word embedding space

MULTIPLY NETWORK 10.12 DC

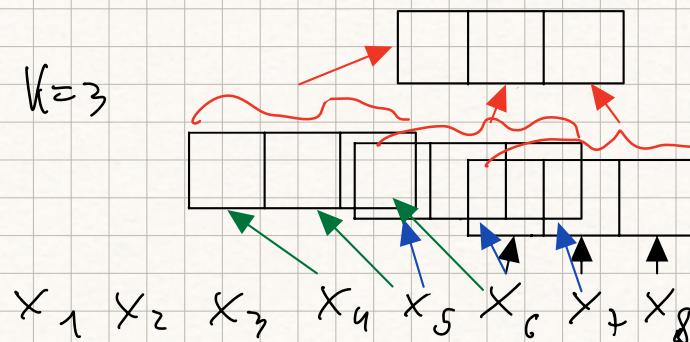
NEURAL MACHINE TRANSLATION 17.4.5 DC

Convolutional Seq2Seq

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



NO  
RECURRENT



extend the window  
from 3 to 6

# DFTM 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	$O(n \cdot d^2)$	$O(n)$	$O(n) \rightarrow$ backpropagate through n layers
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

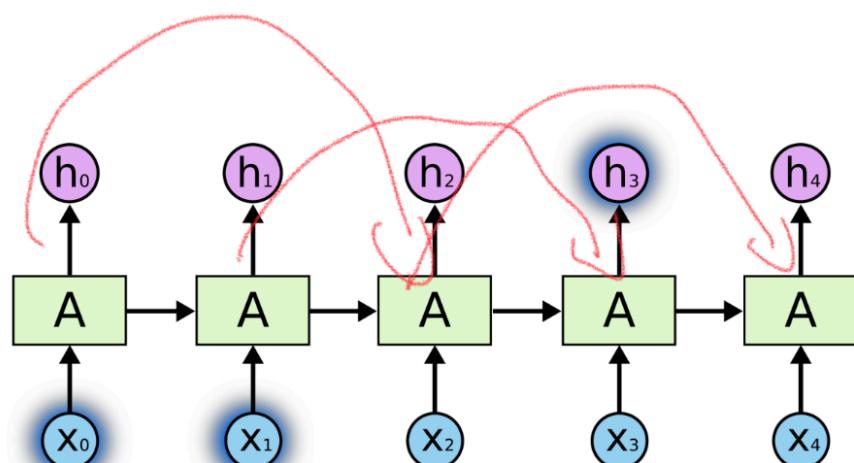
REMA DIGN

## TRANSFORMERS

Vanishing gradients depend on the depth of the network

## Hierarchical RNN

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

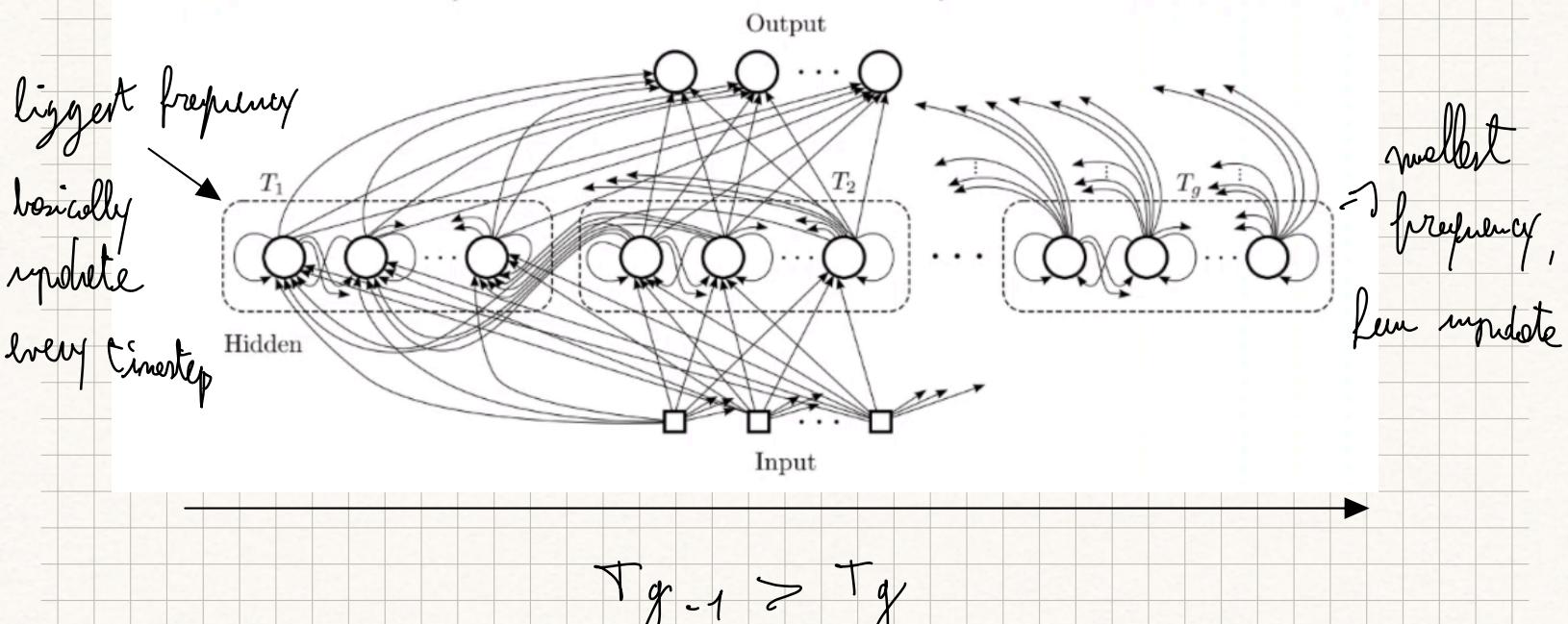


Zone out-(Regularization): reduces chance to update a mode. At each, given a binary mask ( $\omega$  differentiable), we force some units to keep the same value. Owing to the update improves gradient propagation.

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

↳ normal update

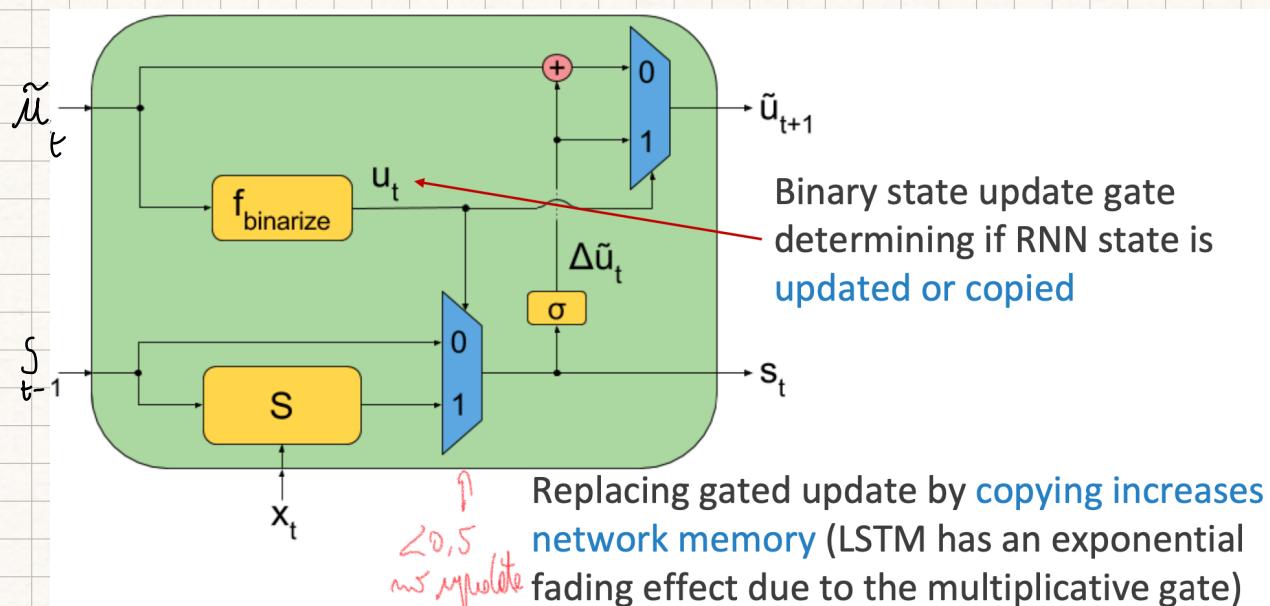
## Clockwork RNN:



$$T_{g-1} > T_g$$

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

## Skip RNN



$$s_t = P(s_{t-1})$$

$$s_t = s_{t-1} \text{ no update}$$

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

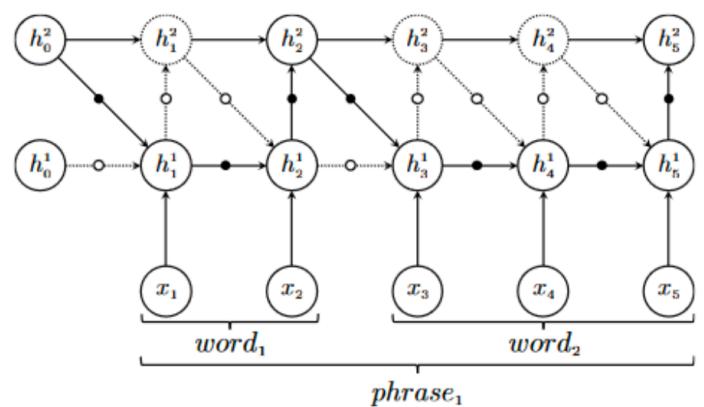
Hierarchical Multiscale RNN (HM-RNN): exploits latent hierarchical structure of sequences. If explicit boundaries are available, we can have different layers for each level of abstraction, combining representations from the lower layer to obtain representation for the higher layer.

## OPERATIONS:

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

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

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



Decap → Recurrence: update at each timestep, linear sum of the recurrence, path length = n

**convolution**: update at each timestep but look at the last  $K$  timesteps, patch length =  $\log_K(n)$

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

**content**: randomly choose unit update

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

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

**HM-RNN**: each layer models more abstract features by learning the dynamics, 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-evt 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 also 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 what input (or function of it) write into memory

**Output feature map:** reads the relevant memory slots.

**Response:** returns the prediction given the retrieved memories

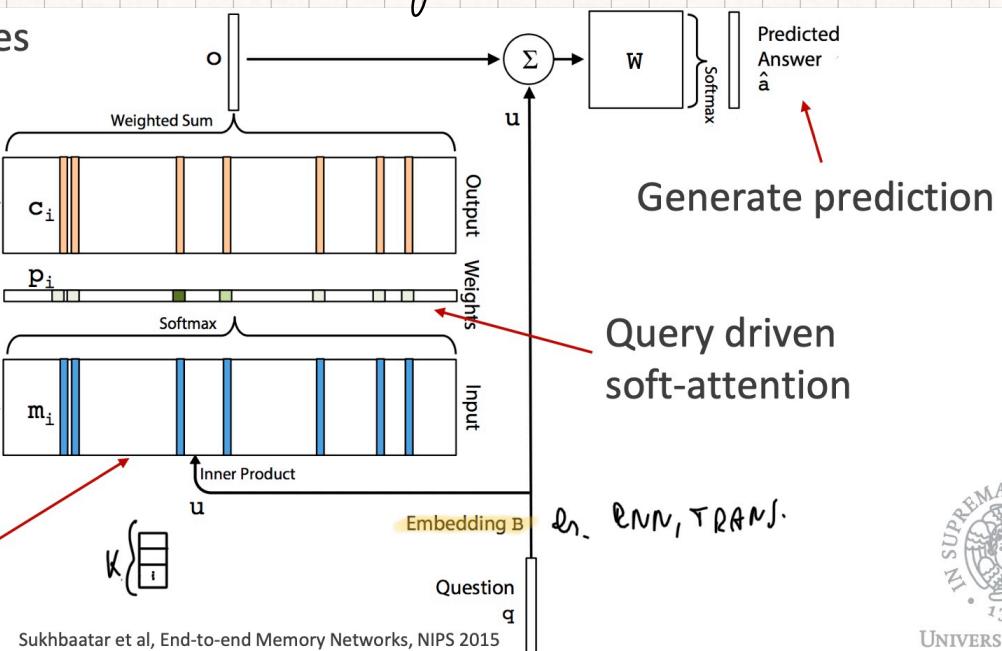
Combine output memories through soft attention

used to matching weight and fact

Facts  $x_i$

used to matching query and each facts

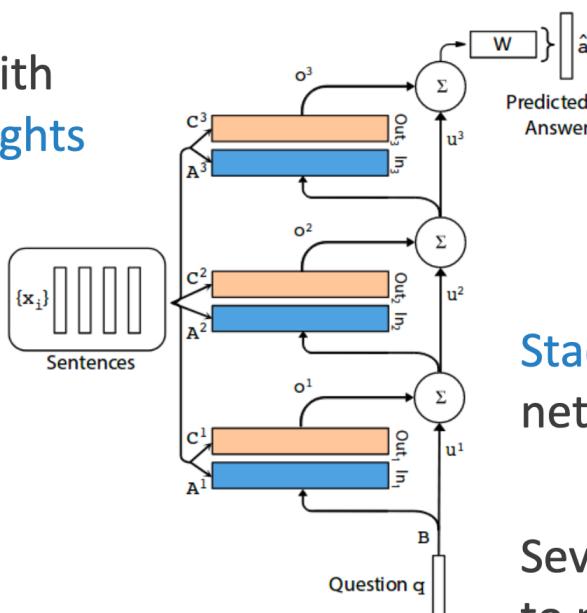
Search for memories matching the query



We can stack this model to multiple layers

Often with tied weights

multiple stack of reasoning

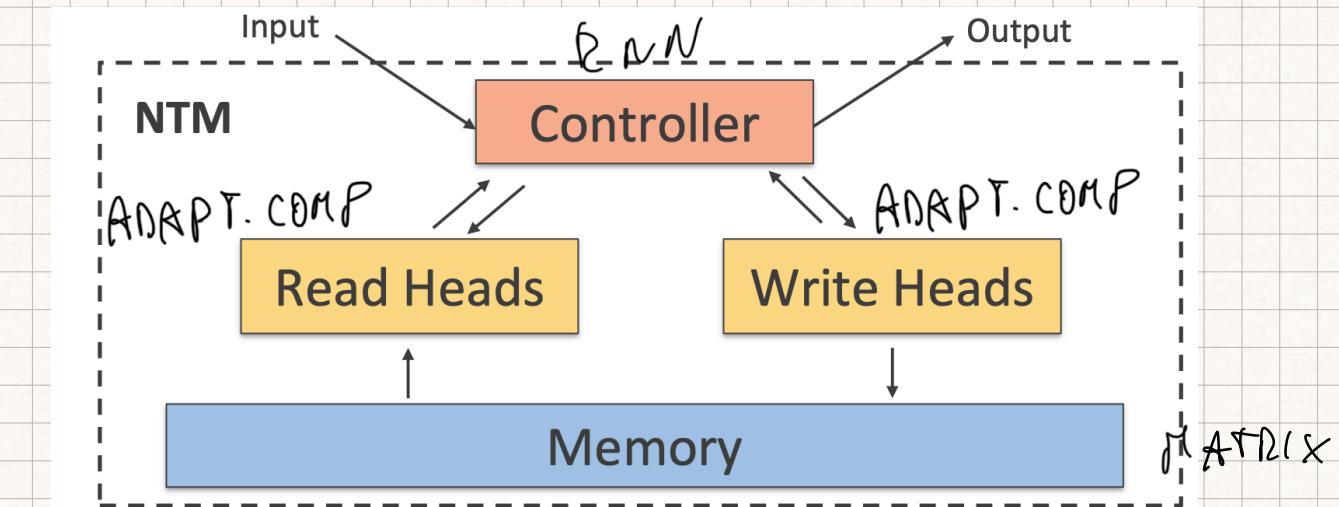


Use more complex output components, e.g. RNN to generate response sequences

Stack multiple memory network layers

Several iterations of reasoning to produce a better answer

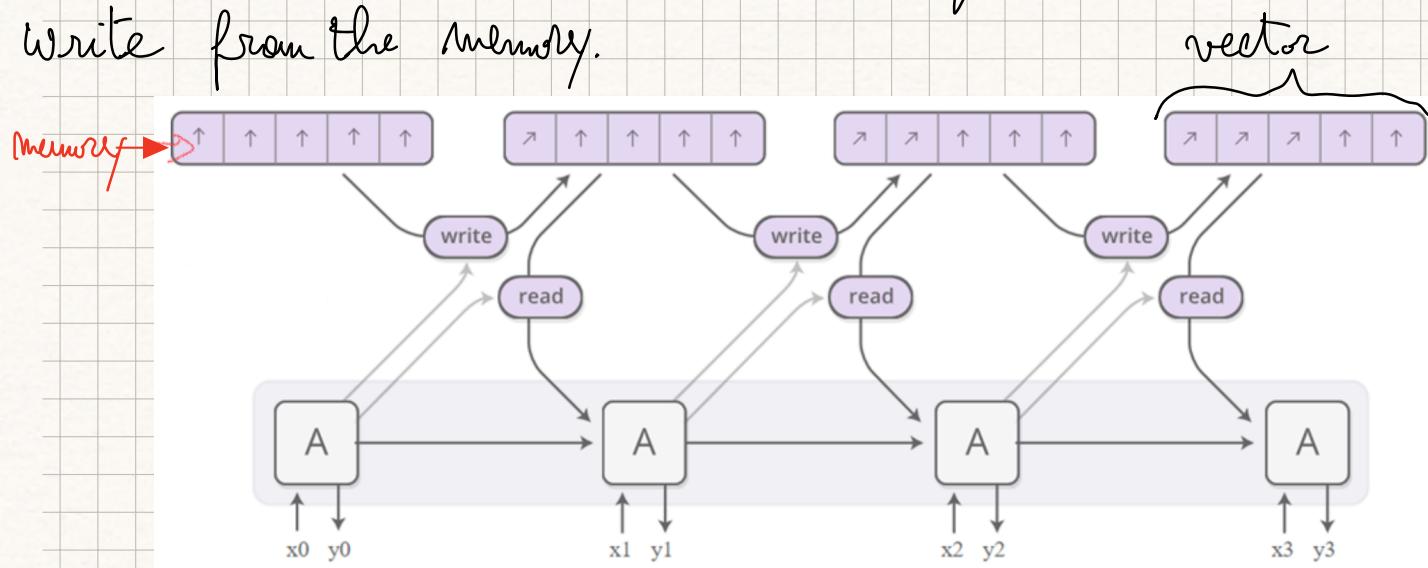
Neural Turing machines: each memory cell can be thought of as an extension of the memory cell in LSTM/GRU, and the network outputs an internal state that chooses which cell to read from or write to. NTM reads to or write from many memory cells simultaneously. Each cell is weighted by a value obtained 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 optimised 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

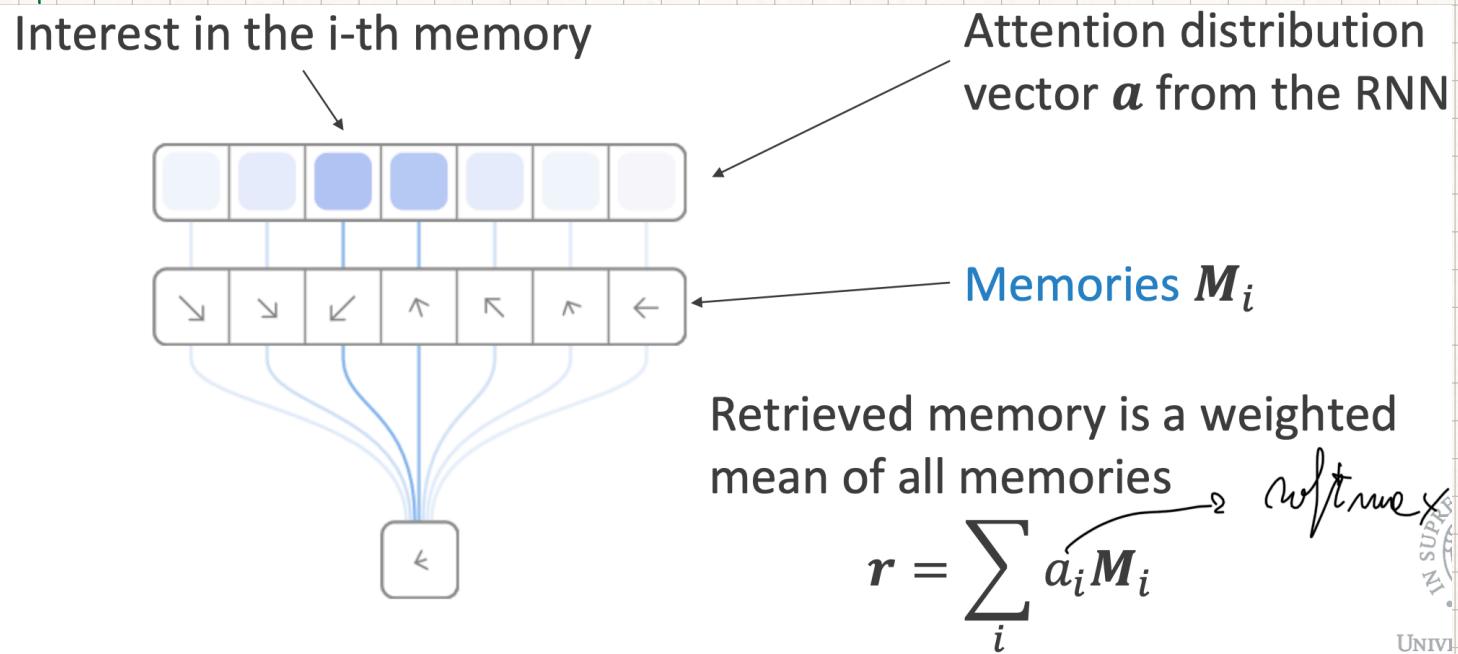
a complete vector-valued memory if we are able to produce a pattern that matches some but not all of its elements.

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

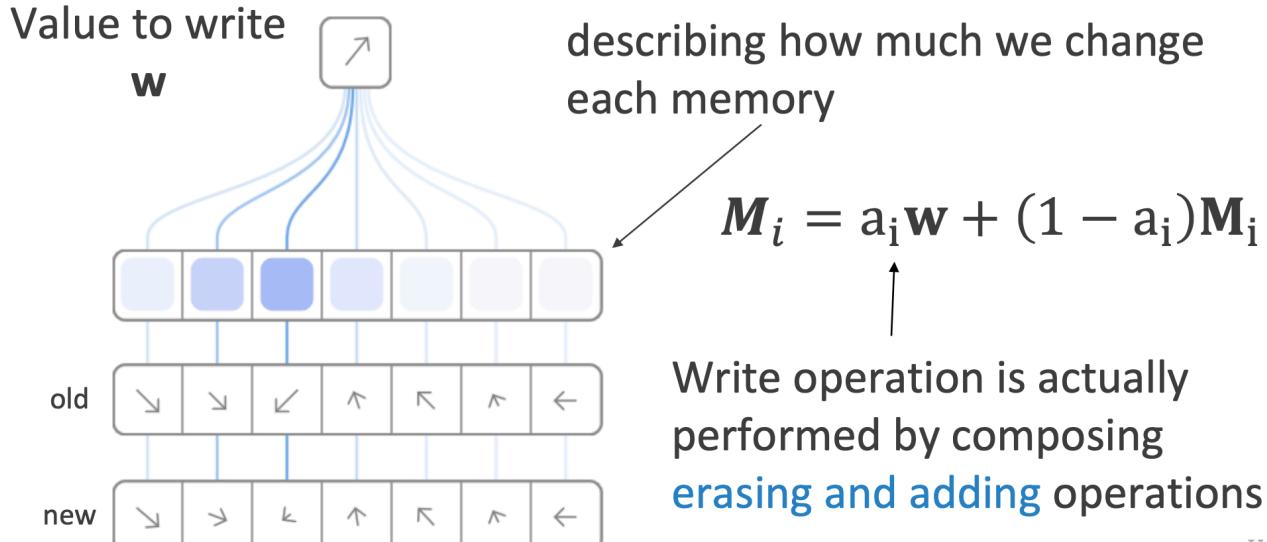


Soft-attention is used to determine how much to read/write from each point.

**Memory read:**

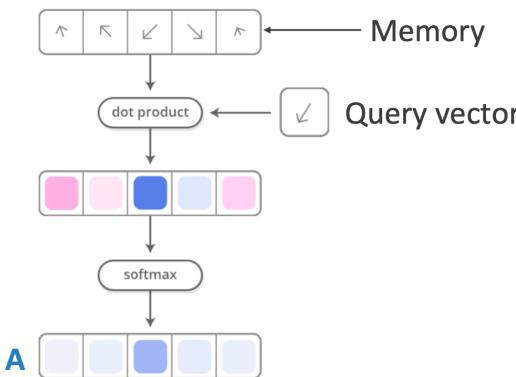


Memory write:

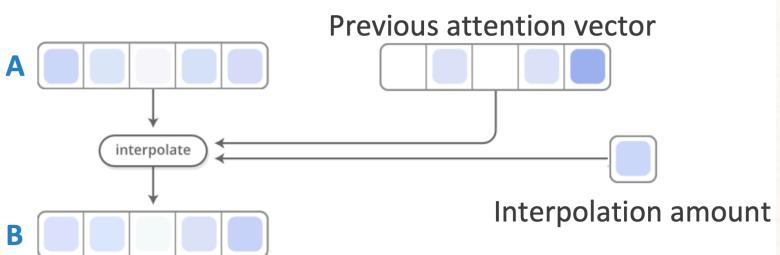


NMT attention focusing

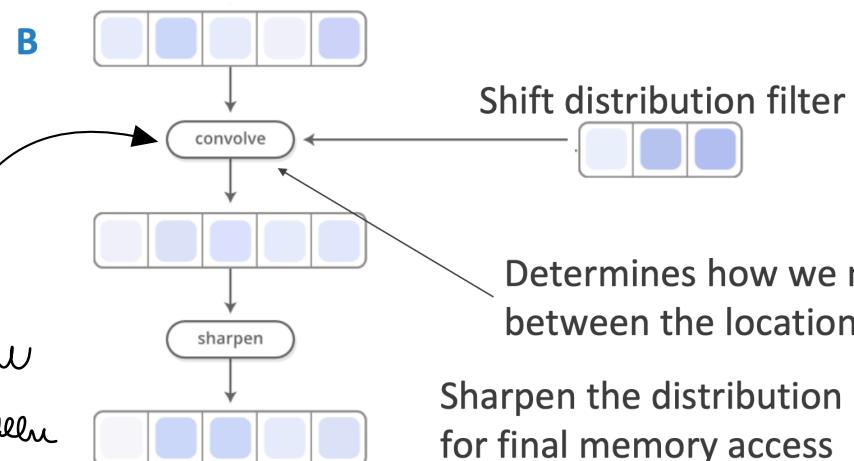
### 1. Generate content-based memory indexing



### 2. Interpolate with attention from previous time



### 3. Generate location-based indexing



Convolution determines how we move between the locations in memory.

Determines how we move between the locations in memory  
Sharpen the distribution for final memory access

# 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's private representation at time  $t$ , given the history of actions

The agent state  $S_t^a$  is the internal representation defined by agent  $a$ , given the history of actions

If fully observable  $\rightarrow$  agent directly observe environment state

$$O_t = S_t^e = S_t^a \quad \text{Markov decision process}$$

Partial observable  $\rightarrow$  agent indirectly observes the environment

$$S_t^a \neq S_t^e \quad \text{Partially observable Markov Decision process}$$

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

$$S_t^a = H_t$$

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

$$S_t^a = \tau(w_s S_{t-1}^e + w_o O_t)$$

Policy  $\pi$  is the agent behaviour. It's basically a distribution given the state  $s$ . Policy are stationary which mean they are time-invariant, deterministic  $a = \pi(s)$  uniform dist.

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

Value function  $v$  is a predictor 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  $\pi$  from state  $s$ .  $\gamma$  scalar between  $[0, 1]$ . weight the reward in the future

Model predict what the environment will do next

$$P(S_{t+1}, R_{t+1} | S_t, A_t) = P(S_{t+1} | S_t, A_t) \cdot P(R_{t+1} | S_t, A_t)$$

predict next state    predict next reward

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

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

**Markov Decision Process**: describe formally an environment for R.L.  
 Environment is fully observable, as 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.  $P_{SS'}^{\alpha} = P(S_{t+1} = S' | S_t = S, A_t = \alpha)$
- $R$  is a reward function s.t.  $R_S^{\alpha} = E[R_{t+1} | S_t = S, A_t = \alpha]$
- $\gamma$  is a discount factor,  $\gamma \in [0, 1]$

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

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

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

$$v(s) = E[G_t | S_t = s]$$

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  $v(S_{t+1})$

$$v(s) = E[G_t | S_t = s] = E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t\right] =$$

$\underbrace{\quad}_{\text{def of } G_t}$

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

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

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

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

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

expected value  
of the immediate reward

expected value of  
the future reward, which  
can be seen as the sum  
over all the value functions

of all the possible future  
state  $s'$ , multiplied by the probability of  
pass from the actual state  $s$  into that specific  $s'$

$$= E[R_{t+1} | S_t = s] + \gamma E[v(S_{t+1}) | S_t = s]$$

$E[R_{t+2} + \gamma R_{t+3}, \dots | S_{t+1}]$

The expected state-value  
of being in any state  
reachable from  $s$

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

↓  
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  $\pi$ .

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

**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  $\pi$

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

**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  $q_\pi(s, a)$ .

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

immediate reward

expected return of being the future state  $s'$  (the value function) weighted by the probability of achieving 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  $\pi$  (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  $\pi$ )

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

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

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

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

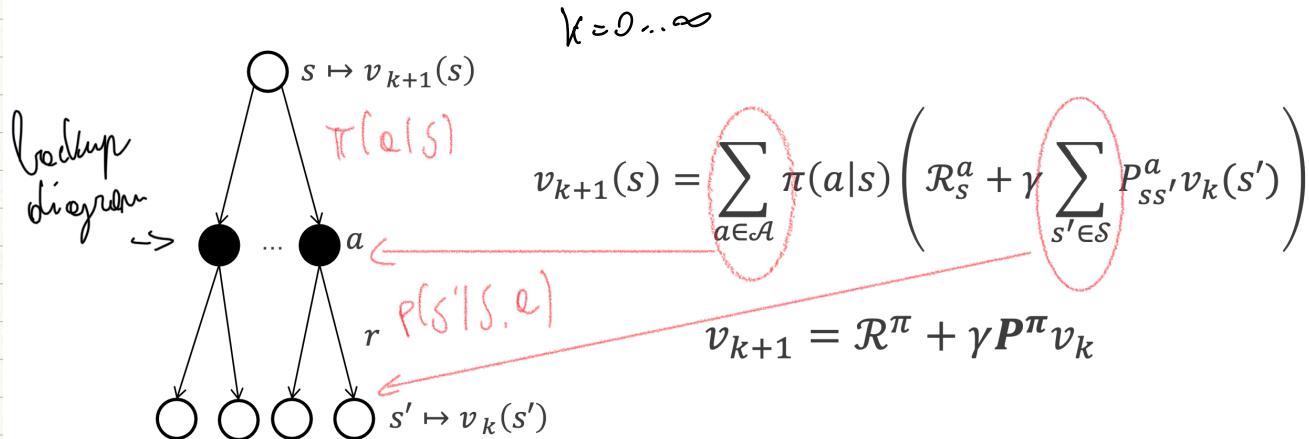
Model-based RL

Policy evaluation: evaluate a given policy  $\pi$ , using at each iteration the Bellman expectation.  $V_1 \rightarrow V_2 \rightarrow \dots \rightarrow V_\pi$  are used to estimate  $V_2$

Asynchronous 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)



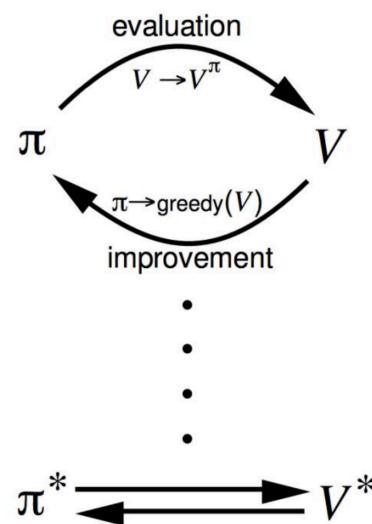
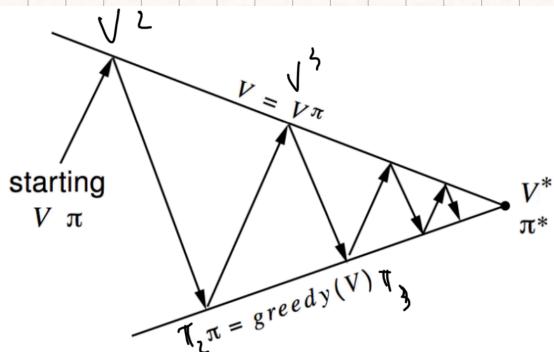
Improve a policy  $\rightarrow$  Given an initial policy  $\pi$ :

1) Evaluate the policy  $V_\pi(s) = E[R_{t+1} + \gamma R_{t+2} + \dots | S_t = s]$

2) Improve the policy by acting greedily with respect to  $V_\pi$

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

**Policy Iteration:** given a policy  $\pi$ , we can iterate two steps in order to improve our policy



✓ Policy evaluation - Estimate  $v_\pi$   
✓ Iterative policy evaluation

✓ Policy improvement - Generate  $\pi' \geq \pi$   
✓ Greedy policy improvement

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

Observe the Bellman optimality, so  $v_{\pi}(s) = v_{\pi^*}(s), \forall s$

Value iteration: find optimal policy  $\pi$ , applying iteratively

Bellman optimality

$$v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_{\pi}$$

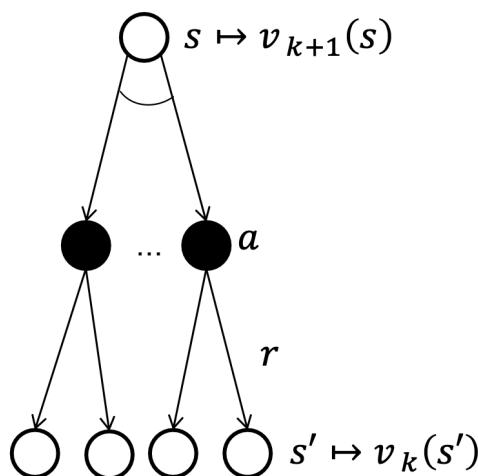
ynchronous backups

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 intermediate value functions may not correspond to our policy.



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

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

# Model-based Reinforcement Learning

evaluate a policy, learn value function, action value function

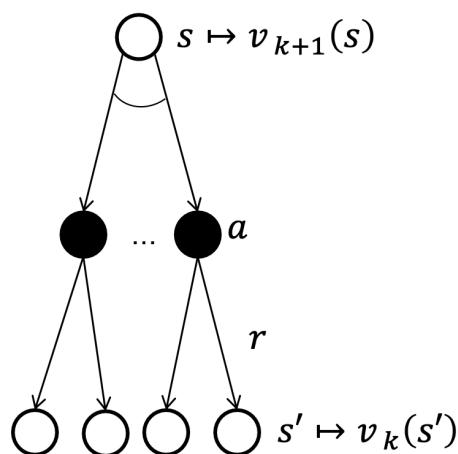
Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

learn an optimal policy, given no one

- why  
join one
- 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 does not knowlede of MDP transitions/rewards. MC learns from complete episodes (no bootstrapping). Value corresponds to the mean return over different episodes

$$S \xrightarrow{G_1} V(S) = \text{AUG}(G_i)$$

$$S \xrightarrow{G_2}$$

$$S \xrightarrow{G_3}$$

MC policy evaluation: learn  $V_\pi$  from episodes of experience under policy  $\pi$   $S_1, A_1, R_2, \dots, R_n \sim \pi$ .

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

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

For each state  $S_t$  with return  $G_t$ :

- 1) Increment counter  $N(S) \leftarrow N(S) + 1$   $S_t = S$
- 2) Update value function (with incremental mean)  

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

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

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

↑  
"learning rate"

## Temporal-Difference (TD) learning

TD is model-free and learns from incomplete episodes of experience, by bootstrapping. TD update  $\pi$  towards  $\pi$ .

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

$$V(S_t) \leftarrow V(S_t) + \alpha (R_t + \gamma V(S_{t+1}) - V(S_t))$$

$\xrightarrow{\text{TD-target}}$

$\xrightarrow{\text{TD-error}}$

old value

## 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  $V_\pi(S_t)$ .

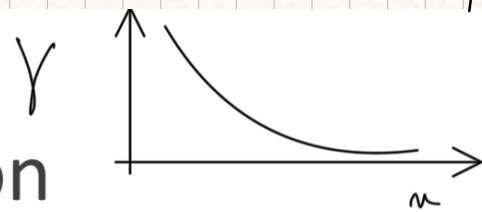
TD target  $R_{t+1} + \gamma V(S_{t+1})$  is biased estimate of  $v_\pi(S_t)$

In MC you learns after having taken different actions in a single episode (high variance)  $\rightarrow$  (less bias)

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

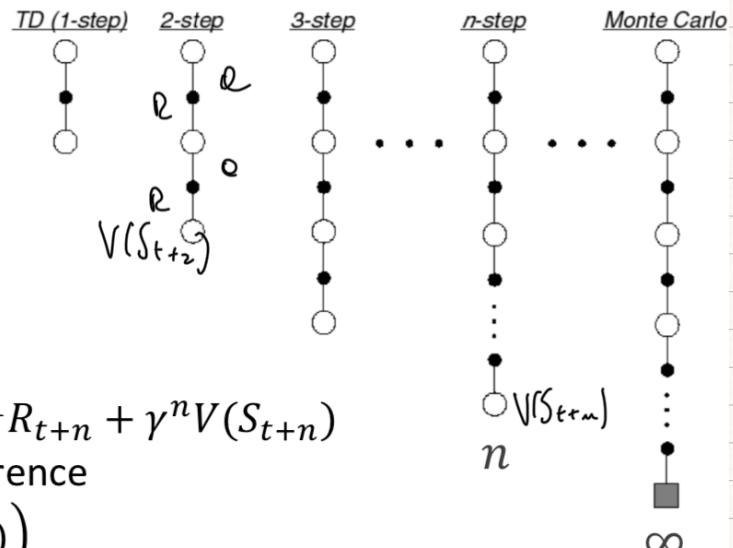
TD target learns from a single action (low variance)  $\downarrow$   
 (high bias)

Is more sample efficient than MC.  $TD(0)$  converges to  $V_\pi(s)$  (but not always with function approximation), and is more resilient to initial value (due to bootstrapping  $\rightarrow$  high bias)



## n-step Prediction

Have TD look and target  $n$  steps in the future



✓ Define the  $n$ -step return

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

✓ Learn based on the  $n$ -step difference

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

## $\lambda$ -returns (Forward View)

$$\lambda \in [0, 1]$$

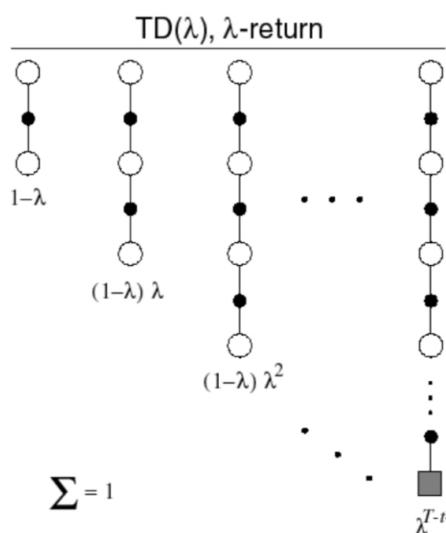
- The  $\lambda$ -return  $G_t^\lambda$  combines all  $n$ -step returns  $G_t^{(n)}$

- Using weight  $(1 - \lambda)\lambda^{n-1}$

We don't have access to  $\rightarrow G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$   
 in practice

- Update as appropriate ( $TD(\lambda)$ )

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



$$\sum = 1$$

Forward view doesn't give mechanism. Backward view does it.  
 Update online, every step, from incomplete sequences. It uses  
 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).  $\lambda$  is the parameter that controls the eligibility  
 trace in the algorithm.

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

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

$$V(s) = V(s) + \alpha f_t E_t(s),$$

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

## Model-free Q Control

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

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

## $\epsilon$ -greedy exploration

We add some randomness to ensure continual state exploration

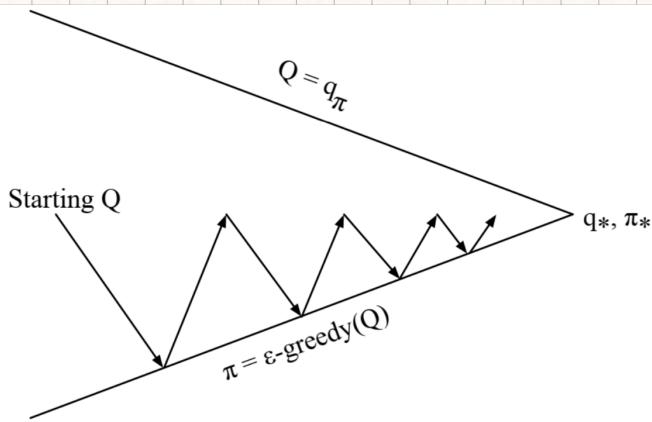
All actions are tried with non-zero probability

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

$\epsilon$  probability of choosing an action at random

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

## On-Policy control with SARSA



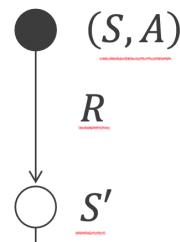
every time-step:

- do policy evaluation - TD,  $Q \approx \varphi \pi$
- do policy improvement -  $\epsilon$ -greedy policy

TD learning: take one action, arrive at the final state  $\rightarrow$  generate the error

$$Q = \boxed{\quad \quad \quad}$$

## Updating Action-Value Functions with SARSA



Expected SARSA

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

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  $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

    Initialize  $S$

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

    Loop for each step of episode:

        Take action  $A$ , observe  $R, S'$

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

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

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

    until  $S$  is terminal

$V \rightarrow E \leftarrow E$

$E \xrightarrow{\text{action}} \boxed{\phantom{\cdot}}$  state eligibility trace

$Q \rightarrow \boxed{\phantom{\cdot}}$

## SARSA( $\lambda$ ) 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.,  $\varepsilon$ -greedy)

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

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

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

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

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

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

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

    until  $S$  is terminal



UNIVERSITÀ DI

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

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

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

The target policy  $\pi$  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  $\mu$  is  $\epsilon$ -greedy w.r.t.  $Q(S, a)$

The Q-learning target then simplifies to

$$\begin{aligned} R_{t+1} + \gamma Q(S_{t+1}, A') &= R_{t+1} + \gamma Q(S_{t+1}, \arg \max_{a'} Q(S_{t+1}, a')) \\ &= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \end{aligned}$$

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

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

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

Repeat (for each episode):

    Initialize  $S$

    Repeat (for each step of episode):

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

        Take action  $A$ , observe  $R, S'$

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

$S \leftarrow S'$ ;

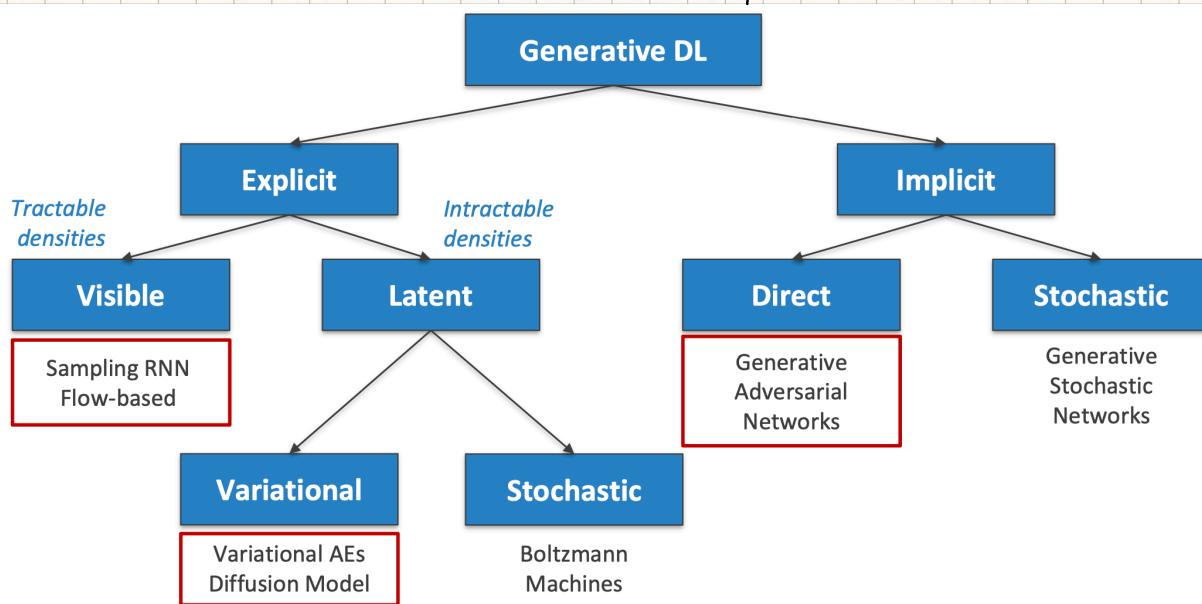
    until  $S$  is terminal

DR FINIRE

# Generative learning:

**Explicit:** it learns a model density  $P_\theta(x)$ . If it's tractable, we can use all the visible lets in a simple 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  $P_\theta(x) \approx P(x)$ . 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.

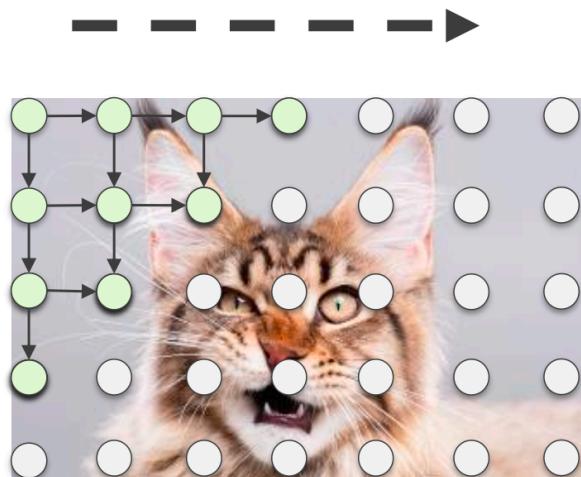
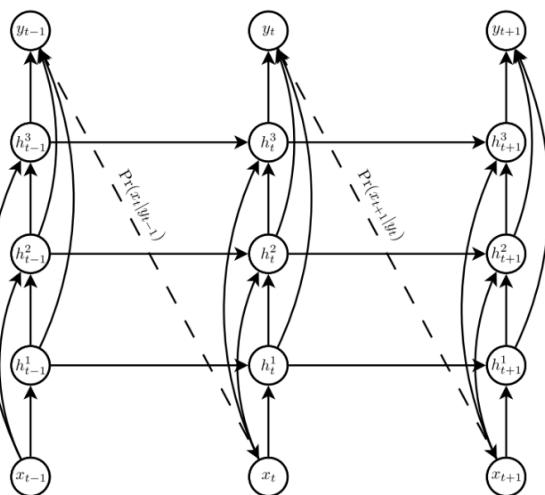
$$\text{Bayesian network} \rightarrow P(x) = \prod_i^N P(x_i | x_1, \dots, x_{i-1})$$

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 precursor.

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.



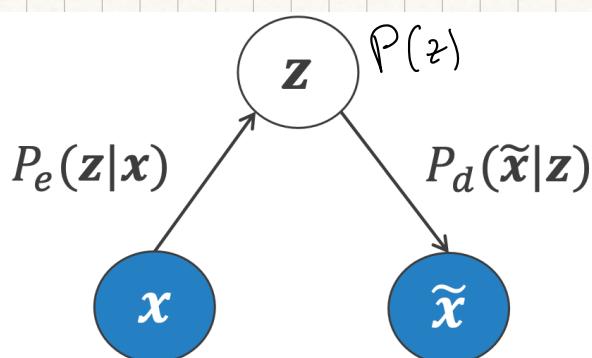
Generating pixels from an image (even small  $64 \times 64$ ) can be difficult due to the gradient vanishing prob. (we have long term dependencies from the first and last pixel).

We can generalize the parameter model  $P_\theta(x) = \prod_i P_\theta(x_i | x_1, x_2, \dots, x_{i-1})$  in a latent space regulated by unobserved variable  $z$ .

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

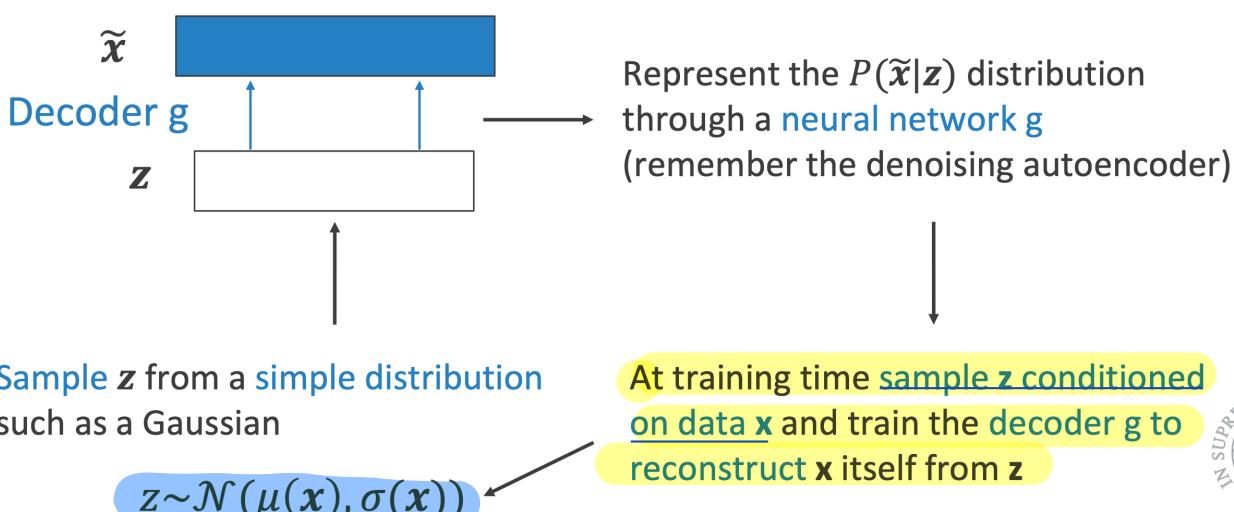
The computation of the integral is not trivial.

**Variational Autoencoders:** it's a NN that, in terms of probabilistic terms, generates  $\tilde{x}$  sampling  $z$  during the encoder 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  $P(\tilde{x}|z)$



distributions  $P(z)$  and  $P(\tilde{x}|z)$

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



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

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

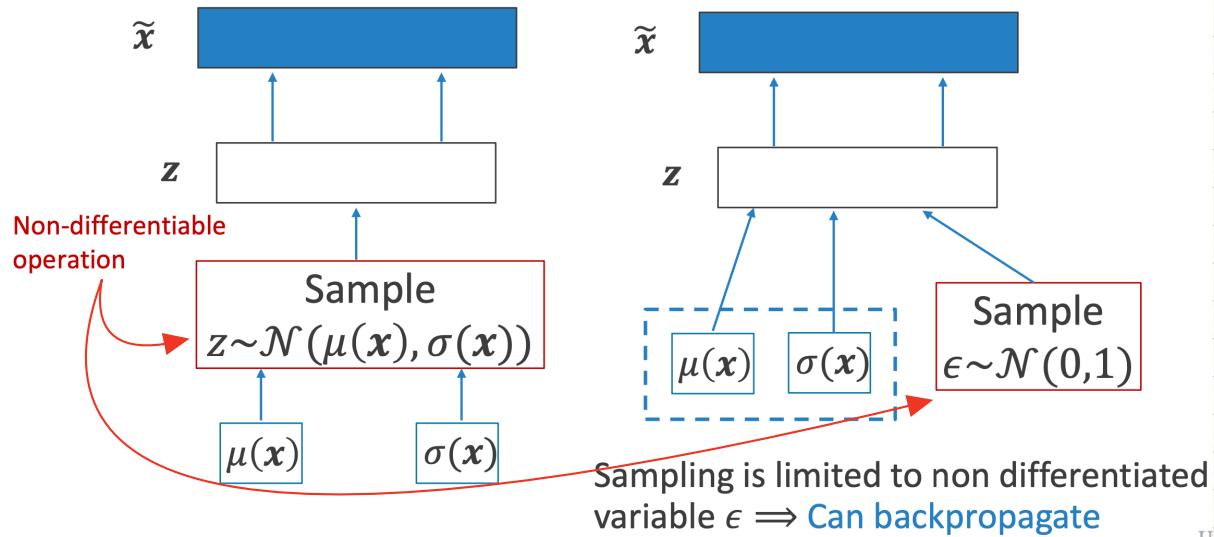
intractable

non differentiable

solved by: variational approximation

reparameterization

Reparameterization trick:



we can rescale white noise with  $\mu$  and  $\sigma$

Variational approximation: in practice we maximize the

$$\text{ELBO} \rightarrow \log P(x|\theta) \geq \mathbb{E}_Q [\log P(x,z)] - \mathbb{E}_Q [\log Q(z)] = \mathcal{L}(x,\theta,\phi)$$

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

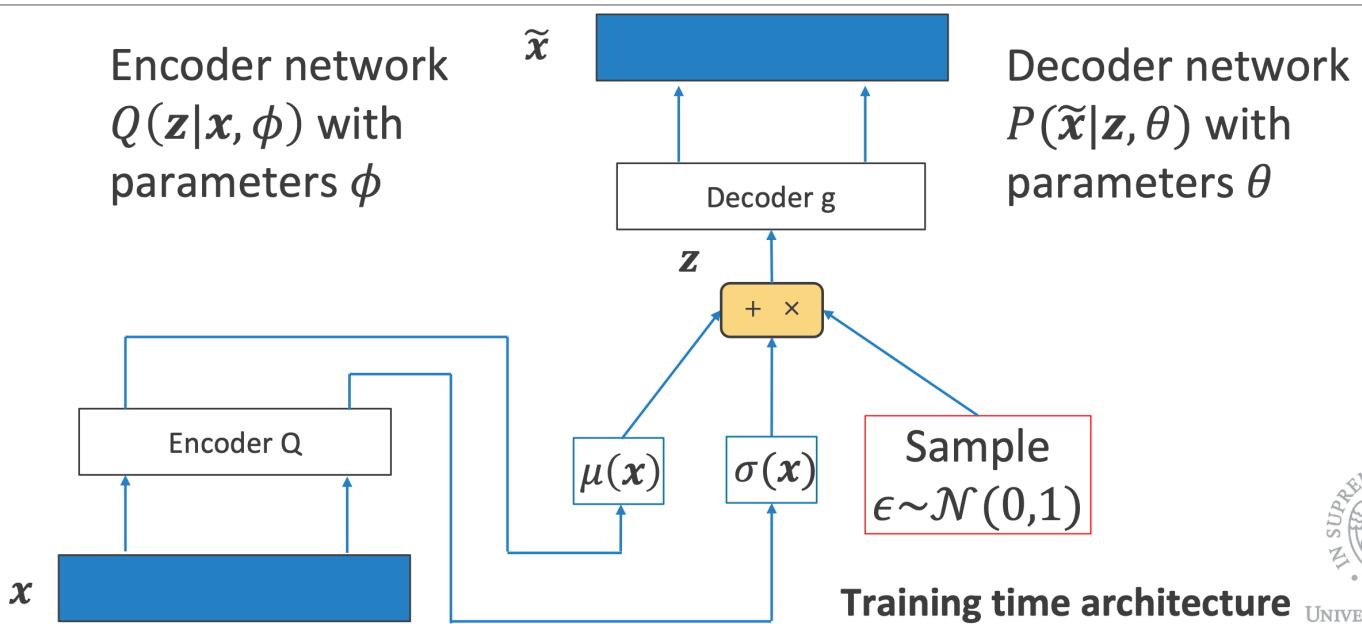
$$\mathcal{L}(x,\theta,\phi) = \mathbb{E}_Q [\log P(x|z)] + \mathbb{E}_Q [\log P(z)] - \mathbb{E}_Q [\log Q(z)]$$

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

$$KL(Q(z|\phi) || P(z|\theta))$$

need  $\circ Q(z)$  function to  
approximate  $P(z)$

## Variational Autoencoder – The Full Picture



VAE Training : training is performed by backpropagation on  $\theta$  and  $\phi$   
in order to maximize the ELBO

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

reconstruction (decoder)

Can be computed in closed form when

both  $Q(z)$  and  $P(z)$  are Gaussians

$$KL(N(\mu(x), \sigma(x)) || N(0, 1))$$

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

VAE final loss:

$$E_{x \sim p} [ E_{\epsilon \sim N(0,1)} [ \log P(x | z = \mu(x) + \sigma(x)^{1/2} \epsilon, \theta) ] - KL(Q(z|x, \phi) || P(z)) ]$$

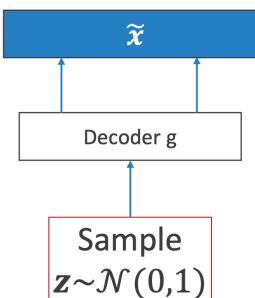
After the reparametrization trick the expectations don't depend on distributions of model parameter so we can perform SGD

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

$$E_{z \sim q} [ \log P(x | z) ]$$

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

VAE testing:

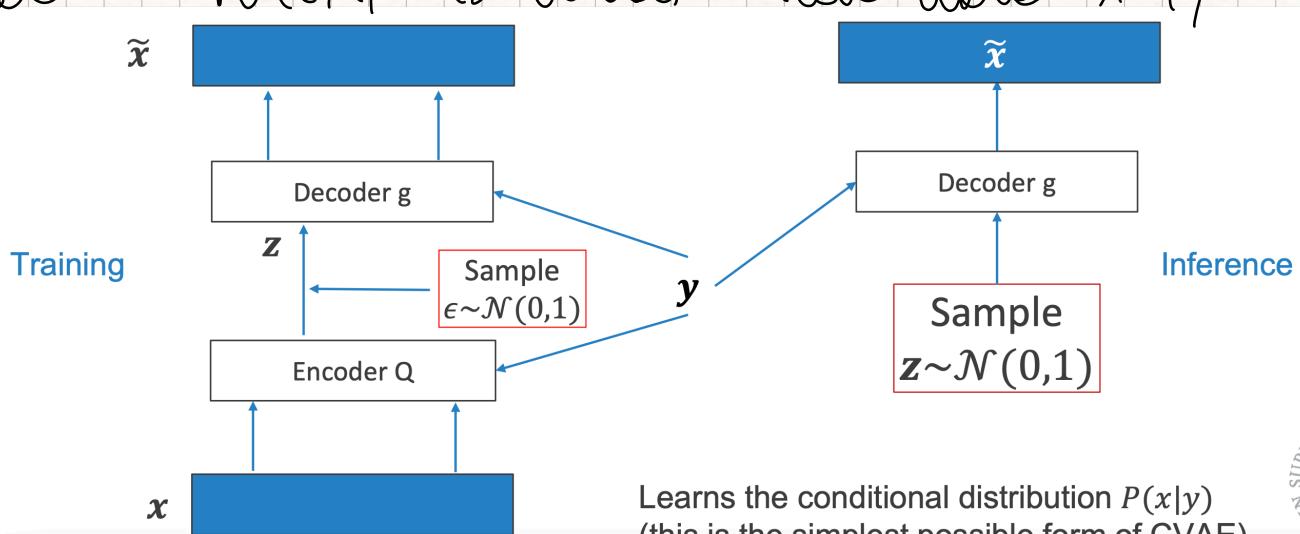


At test time detach the encoder, sample a random encoding and generate the sample or the corresponding reconstruction

**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.

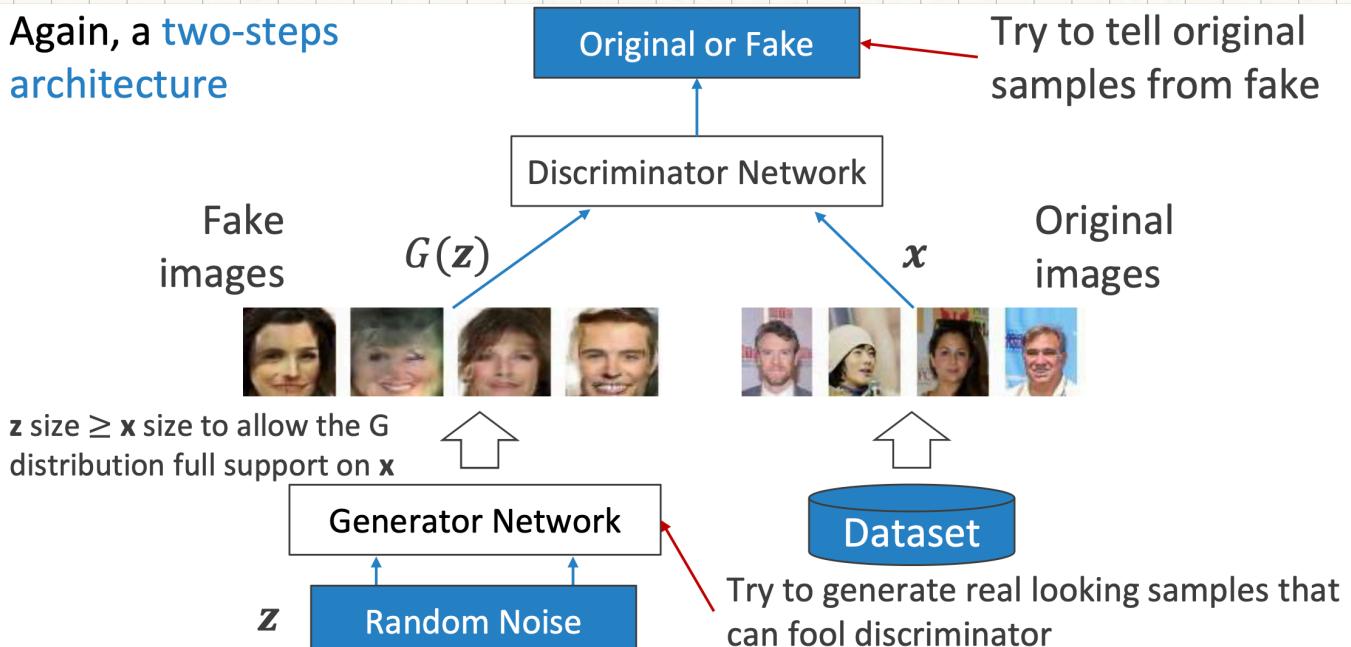
VAE, instead, tells us that given an input  $x$ , we can obtain the distribution of the latent space. This distribution can be sampled into the original space in order to obtain the reconstructed input

**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  $\epsilon \sim \mathcal{N}(0,1)$  to obtain new data  $\tilde{x}|y$



# GAN

Again, a two-steps architecture



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  $D(G(z))$ .

$$C = \min_{\Theta_G} \max_{\Theta_D} \left[ E_{x \sim p} [\log D_{\Theta_D}(x)] - E_{z \sim p} [\log (1 - D_{\Theta_D}(G_{\Theta_G}(z)))] \right]$$

Annotations for the loss function:

- A green arrow labeled "gradient descent" points to the term  $E_{z \sim p} [\log (1 - D_{\Theta_D}(G_{\Theta_G}(z)))]$ .
- A green arrow labeled "gradient ascent" points to the term  $E_{x \sim p} [\log D_{\Theta_D}(x)]$ .
- A red bracket labeled "discriminator output for real data  $x$ " spans the term  $E_{x \sim p} [\log D_{\Theta_D}(x)]$ .
- A blue bracket labeled "discriminator output for fake data  $G(z)$ " spans the term  $E_{z \sim p} [\log (1 - D_{\Theta_D}(G_{\Theta_G}(z)))]$ .

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  $C_G = \max_{\theta_G} \mathbb{E}_{\mathbf{z}} [\log D_{\theta_d}(G_{\theta_G}(\mathbf{z}))]$  (maximize the likelihood of olive being wrong)

## GAN Training pseudo-code

```

for number of training iterations do
    for k steps do
        • Sample minibatch of m noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
        • Sample minibatch of m examples  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  from data generating distribution  $p_{\text{data}}(\mathbf{x})$ .
        • Update the discriminator by ascending its stochastic gradient:
            
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m [\log D_{\theta_d}(\mathbf{x}^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}^{(i)})))]$$

    end for
    • Sample minibatch of m noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
    • Update the generator by ascending its stochastic gradient (improved objective):
        
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D_{\theta_d}(G_{\theta_g}(\mathbf{z}^{(i)})))$$

end for

```

discriminator

generator

Stability trick but difficult to choose k

Expectation

The optimal solution 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.

$$G^* = \underset{G}{\operatorname{arg\min}} \underset{D}{W}(\mu_1, \mu_G) = \text{generator dist.} \rightarrow \text{real data dist.}$$

$$= \underset{G}{\operatorname{arg\min}} \sup_D \left[ \mathbb{E}_{x \sim \mu} [D(x)] - \mathbb{E}_{x \sim \mu_G} [D(x)] \right] \quad \|D\|_L \leq 1$$

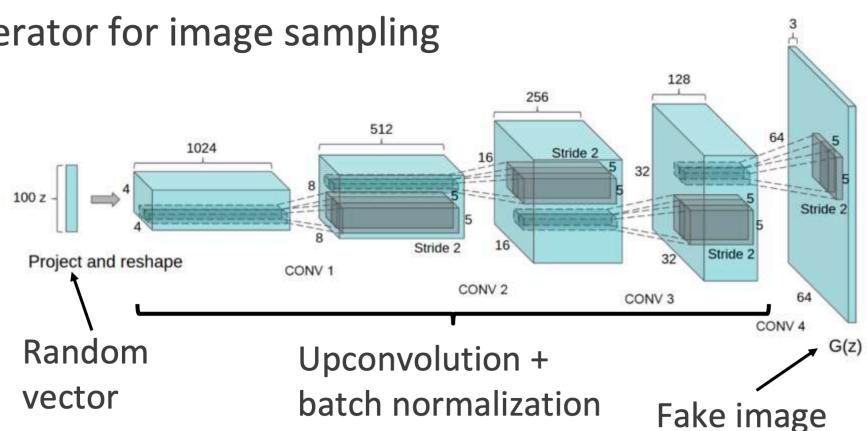
clipping Discriminator weights  
(slow to converge)

the discriminator is contractive because the norm is  $\leq d$  e constant

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

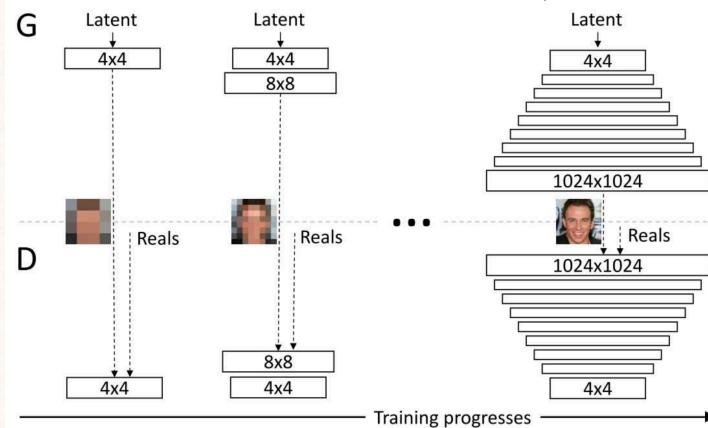
## DCGAN Architecture

Generator for image sampling

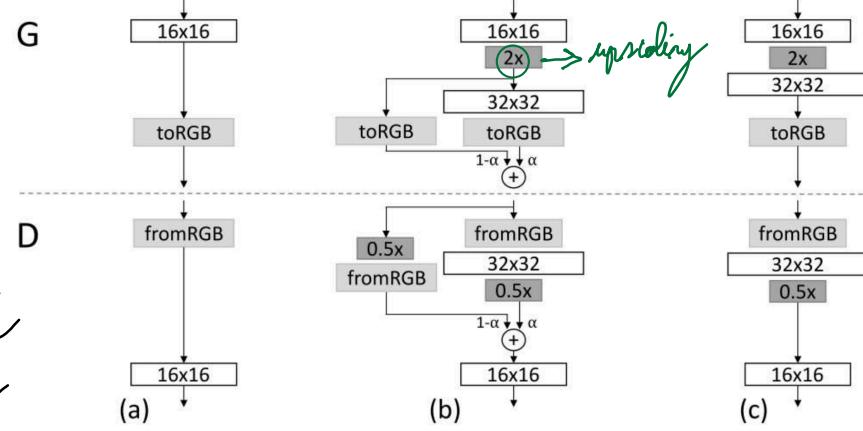


Discriminator is just a convolution network that takes fake images or input

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  $\alpha$  to trade between the upsampled smaller residual image  $D$  and the bigger one, slowly giving

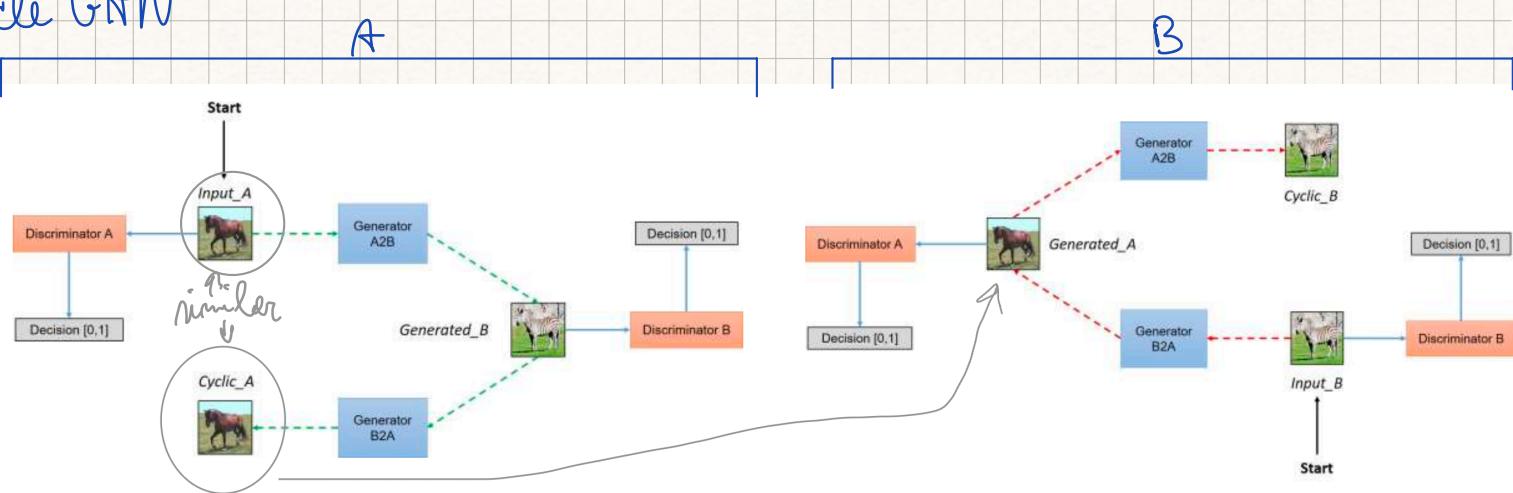
more weight to the larger image as we proceed in the training

Conditional generation: learn a mapping from an observed side information  $x$  and a random noise vector  $z$  to the fooling samples  $y$

$$g: \{x, z\} \rightarrow y$$

Auxiliary Information

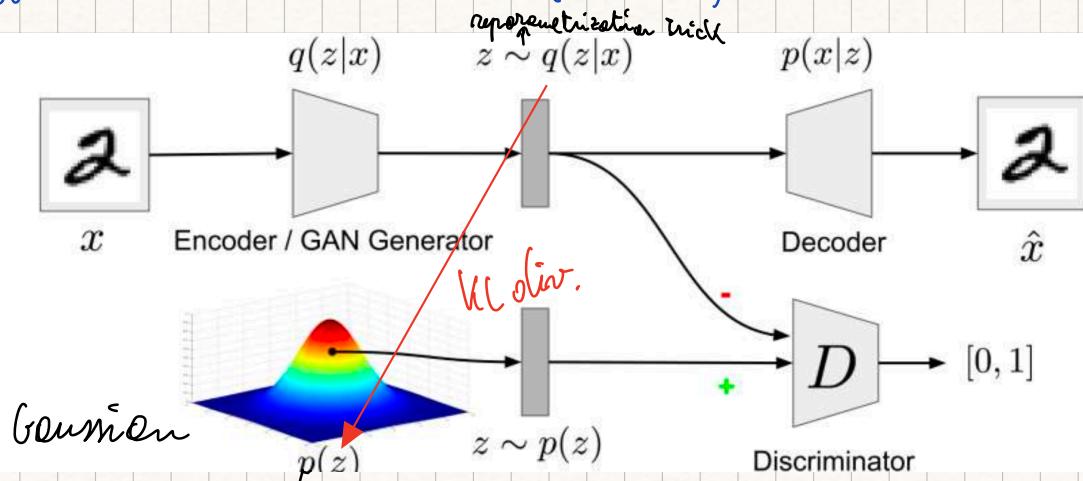
Cycle GAN



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 priori distribution. The discriminator discriminates the  $z$  from the encoder from the  $z$  from the distribution.

prior distribution. The discriminator discriminates the  $z$  from the encoder from the  $z$  from the distribution.

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

0.1 (WASSERSTEIN)  
replaced by an adversarial loss

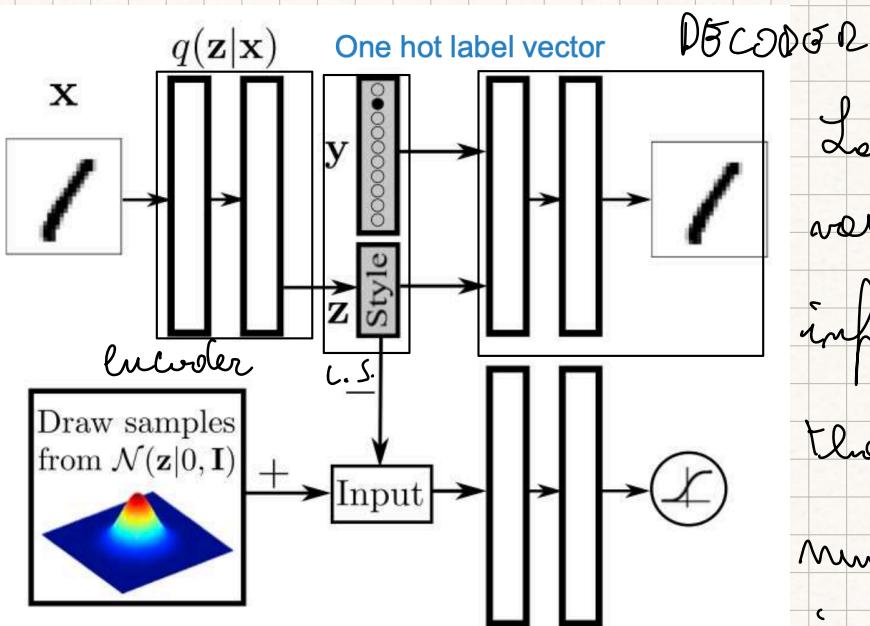
Reconstruction phase : update the encoder and decoder to minimize reconstruction error

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

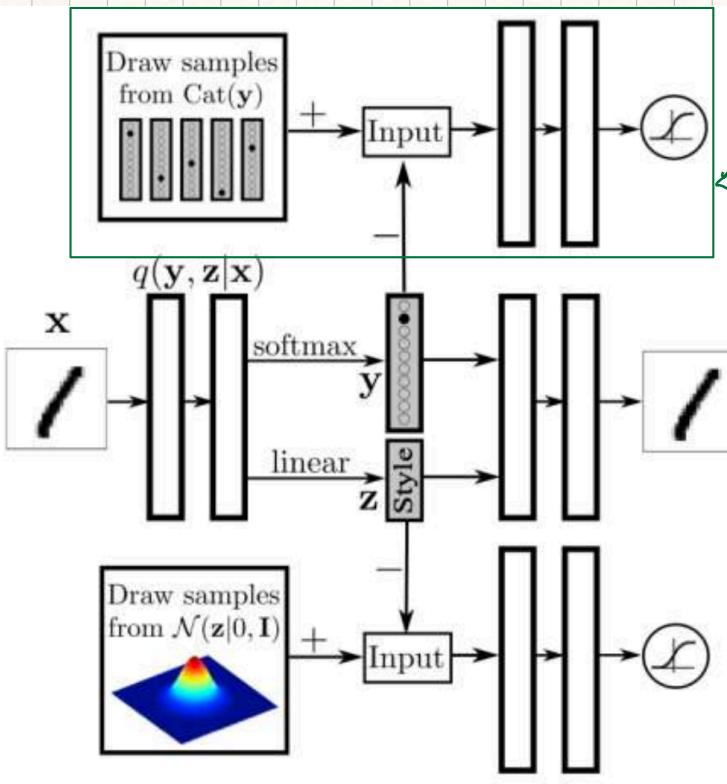
Adversarial regularization allows to impose priors for which we cannot compute KL divergence.

AAE yields a smoother coverage of the latent space.

## Style transfer

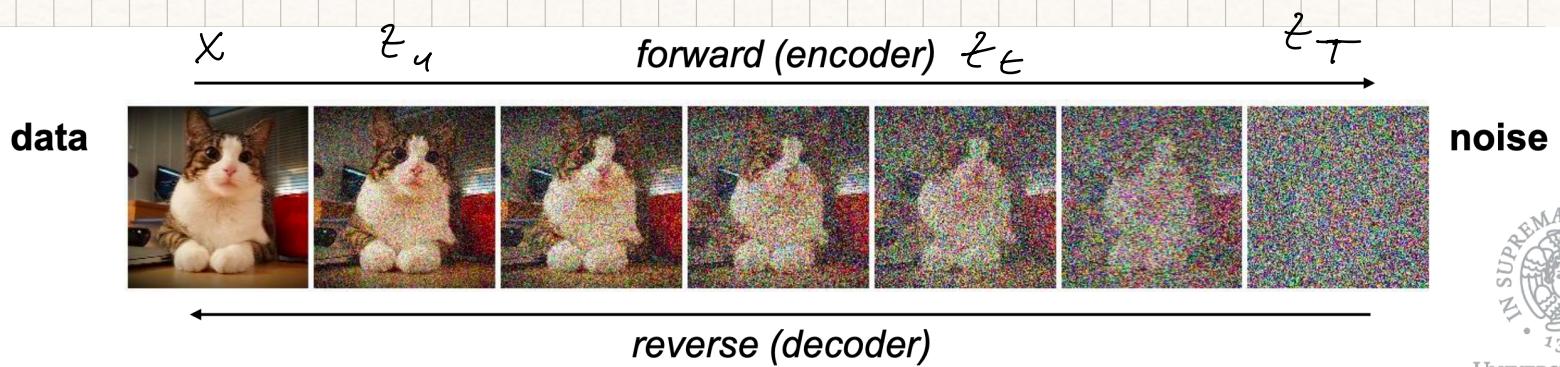


Latent space is formed by a variable that encodes the style information and a categorical variable that encodes the identity of the number. The decoder uses both information to create a number for a given style. We change the value of  $\epsilon$  in order to change the style.



If I don't know  $y$ , 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 here 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  $\epsilon_t \sim N(0, 1)$  at time  $t$  to the original data  $x$ , obtaining at the end a noise - size latent variable  $z_t$ .

$$z_1 = \sqrt{1 - \beta_1} x + \sqrt{\beta_1} \epsilon_1 \quad z_t = \sqrt{1 - \beta_t} z_{t-1} + \sqrt{\beta_t} \epsilon_t$$

$\beta_t \in [0, 1]$  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  $q(z_t | \cdot)$

$$q(z_t | x) = N(\sqrt{2t}x, (1 - \alpha_t)\mathbb{I})$$

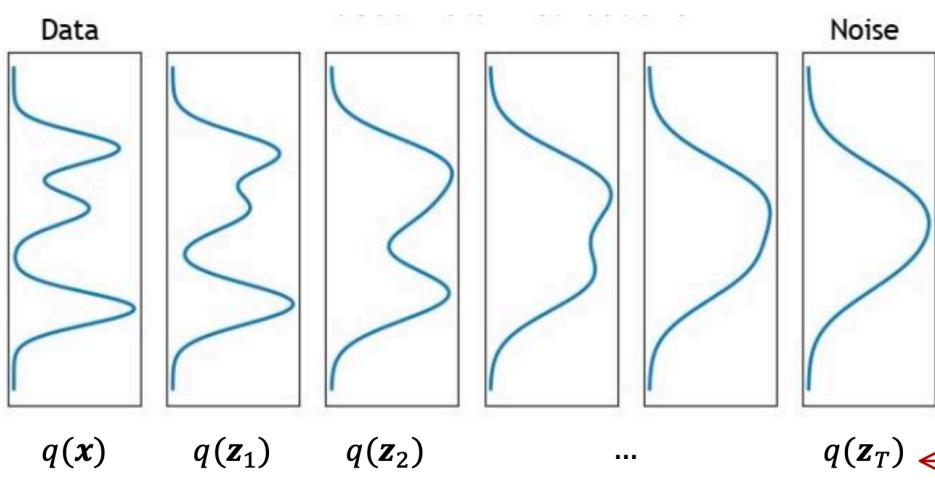
$$\alpha_t = \prod_{s=1}^t (1 - \beta_s)$$

We drew a sample from  $q(z_t | x)$  to obtain  $z$  at time  $t$  in one shot.

I can write the marginal as:  $q(z_t) = \int q(z_t, x) dx = \int q(x) q(z_t | x) dx$

$$q(z_t) = \int q(z_t, x) dx = \int q(x) q(z_t | x) dx$$

no parameter  
in the forward  
process.

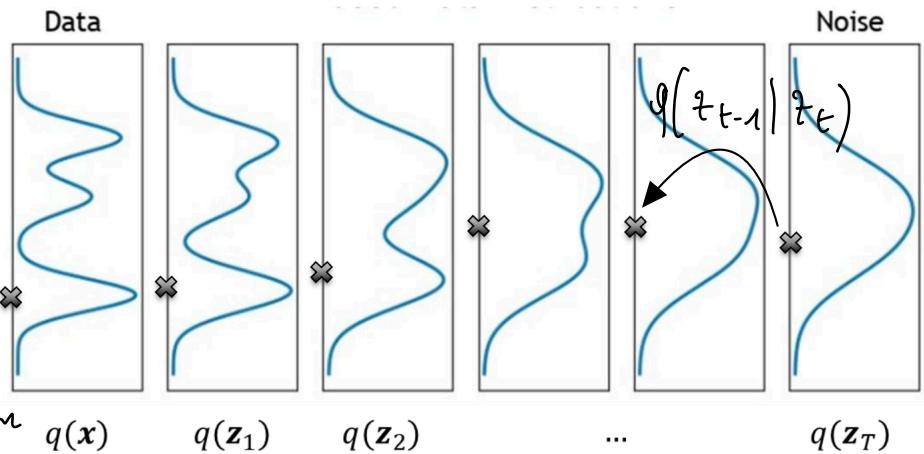


Denoising

sample  $z_t \sim N(0, 1)$

iterate  $z_{t-1} \sim q(z_{t-1} | z_t)$

true denoising distribution



$$\text{is intractable } q(z_{t-1}|z_t) = \frac{q(z_{t-1})q(z_t|z_{t-1})}{q(z_t)}$$

If we know  $x$  we can demonstrate that  $q(z_{t-1}|z_t, x)$  is Normal

The reverse process learns an approximated density dist.  
(observer), assuming reverse distributions are approximately Normal.

Basically we have  $Q NN$  that outputs ~~the mean and the variance~~

$$P(z_t) = N(0, t)$$

$$P_\theta(z_{t-1}|z_t) = N(\mu_\theta(z_t, t), \sigma_t^2 I)$$

actually the variance  
is like the noise schedule,  
is not a hyperparameter.

mean of the denoised image  $z_{t-1}$   
predicted by the  $\theta$ -parametrized model  
given  $z_t$  (and time encoding)

Training follows the classical log-likelihood maximization view

$$\begin{aligned} \log P_\theta(x) &= \log \int P_\theta(z_1, \dots, z_T, x) dz_1, \dots, z_T = \\ &= \log \int P_\theta(x|z_1) \prod_{t=2}^T P_\theta(z_{t-1}|z_t) P_\theta(z_t) dz_1, \dots, z_T \end{aligned}$$

UNTRACTABLE

We use ELBO to 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

$$E_{q(z_t|x)} [\log P_\theta(x|z_t)] - \sum_{t=2}^T KL(P_\theta(z_{t-1}|z_t) || q(z_{t-1}|z_t, x))$$

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 TQ.

how good the learned  
density est.  $P_\theta$  is w.r.t.  $q$

Both dist. will be normal w

we can solve the KL divergence  
in closed form.

$$\sum_x \left( \underbrace{(-\log N(\mu_\theta(z_1, t), \sigma_1^2 I))}_{\text{Reconstruction}} + \sum_{t=2}^T \frac{1}{2\sigma_t^2} \left\| \underbrace{\left( \frac{(1-\alpha_{t-1})}{(1-\alpha_t)} \sqrt{1-\beta_t} z_t + \frac{\sqrt{\alpha_{t-1}\beta_t}}{(1-\alpha_t)} x \right)}_{\text{Target mean of } q(z_{t-1}|z_t, x)} - \underbrace{\mu_\theta(z_t, t), \sigma_t^2 I}_{\text{predicted } z_{t-1}} \right\|^2 \right)$$

Minimize the difference between estimate of  $z_{t-1}$  and the most likely  
value from ground truth observed 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  $\epsilon_\theta$ .

$$\text{we can rewrite as: } x = \frac{1}{\sqrt{1-t}} z_t - \frac{\sqrt{1-2t}}{\sqrt{2t}} \epsilon$$

inserting  $x$  in the B(B) yields

$$\text{Loss}(\theta) = \sum_x \sum_{t=1}^T \| \epsilon_\theta (\underbrace{\sqrt{dt} x + \sqrt{1-dt} \epsilon_t, t}_{z_t}) - \epsilon_t \|^2$$

a network which

add in the  
forward phase

predict the next noise

given current noised input

$z_t$  and an embedding of the

time

#### Algorithm 18.1: Diffusion model training

```

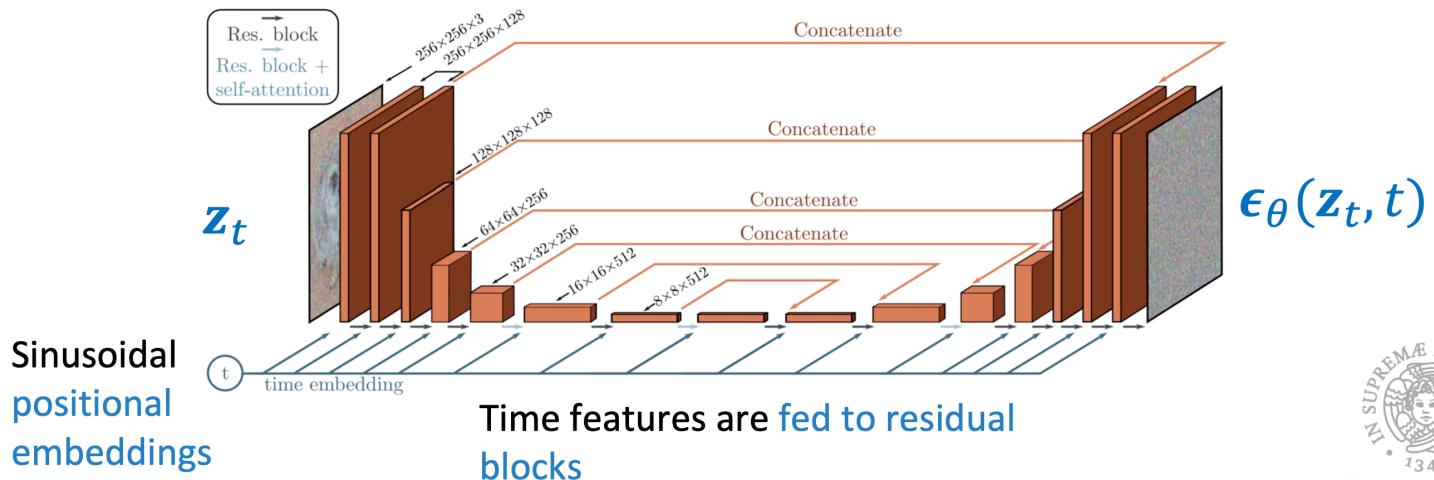
Input: Training data  $x$ 
Output: Model parameters  $\theta$ 
repeat
  for  $i \in \mathcal{B}$  do
     $t \sim \text{Uniform}[1, \dots T]$  // For every training example index in batch
     $\epsilon \sim \text{Norm}[\mathbf{0}, \mathbf{I}]$  // Sample random timestep
     $\ell_i = \| \epsilon_\theta (\sqrt{a_t} x_i + \sqrt{1-a_t} \epsilon, t) - \epsilon \|^2$  // 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 e  
NN and then subtract  
it from the image  
to denoise it.

## U-Net architectures with ResNet blocks and self-attention layers



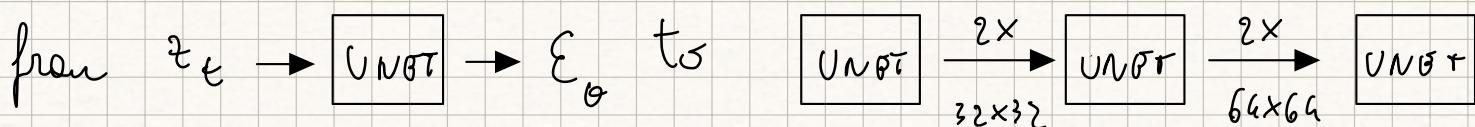
TATUM CTP

Terms  $\beta_t$  and  $\tau_t$  control variance of forward diffusion and reverse denoising, respectively.  $\beta_t$  linear schedule and  $\tau_t^2 = \beta_t$ . Slowly increase the amount of added noise (or high-res information is corrupted first).

$\tau_t$  can be learned by minimizing the bound.

$\beta_t$  can be learned by minimizing the variance of the training objective

High res-imagers  $\rightarrow$  Concatenation of UNet



Guided generation: guide diffusion process using auxiliary detector, 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.

$$z_{t-1} = \underbrace{z_{t-1} + \sigma_t E}_{\text{reversed diffusion}} + \sigma_t^2 \frac{\partial P(c|z_t)}{\partial z_t} \underbrace{\frac{\partial P(c|z_t)}{\partial z_t}}_{\text{classifier guidance}}$$

classifier guidance comes from mixing the predicted score function  
 of the unconditioned diffusion model with the classifier gradient

Given me a  
 (mixed) version  
 of a image with  
 label  $c$

$$\frac{\partial \log P_j(z_t|c)}{\partial z_t} = \frac{\partial \log P(z_t)}{\partial z_t} + \gamma \frac{\partial \log P(c|z_t)}{\partial z_t}$$

guidance scale, higher  $\gamma$  same  
 label  $c$  image, low  $\gamma$  not so  
 clear image

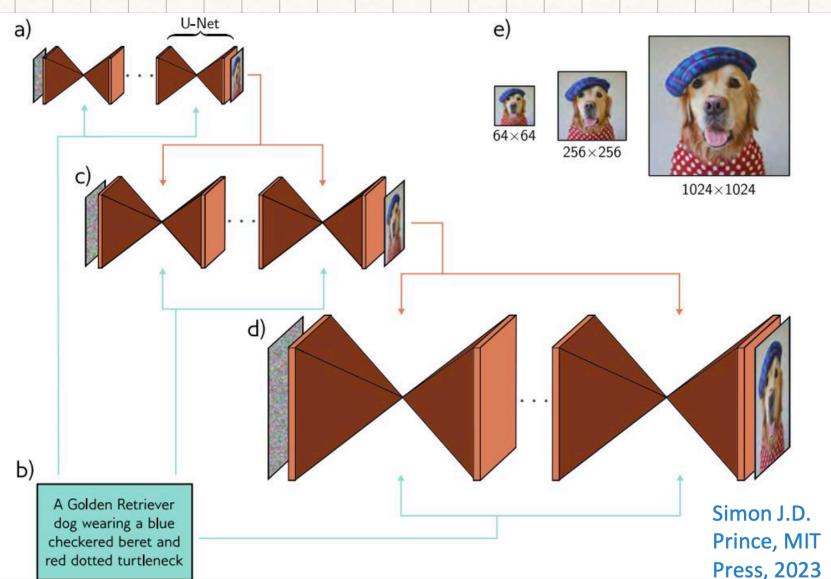
We can obtain a guided version of diff. model mixing the unconditioned  
 one with the conditioned one.

$$\frac{\partial \log P_j(z_t|c)}{\partial z_t} = (\alpha - \gamma) \underbrace{\frac{\partial \log P(z_t)}{\partial z_t}}_{\text{unconditioned}} + \gamma \underbrace{\frac{\partial \log P(z_t|c)}{\partial z_t}}_{\text{conditional}}$$

We don't need to train two different models. We train conditional diffusion model with dropout (randomly removing conditioning). Conditioning replaced by flag input (presence / absence of conditioning). A label is given in input with the time embedding.

Diffusion models can be used to do image representation, denoising back to the image representation space

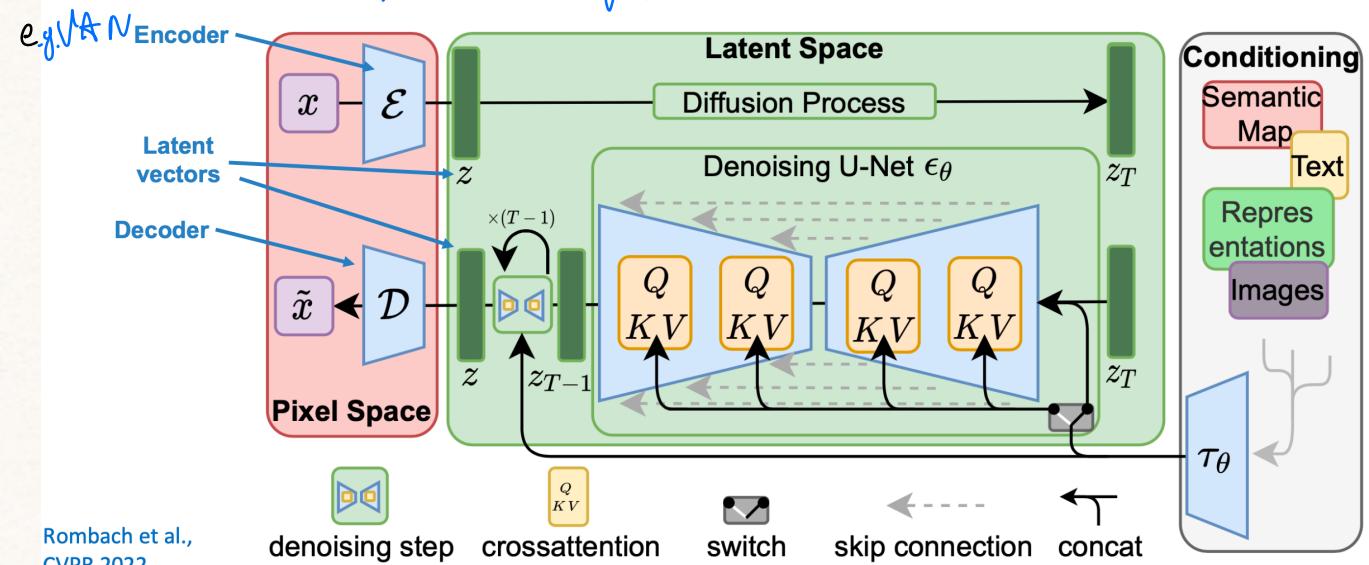
### Concate conditional generation



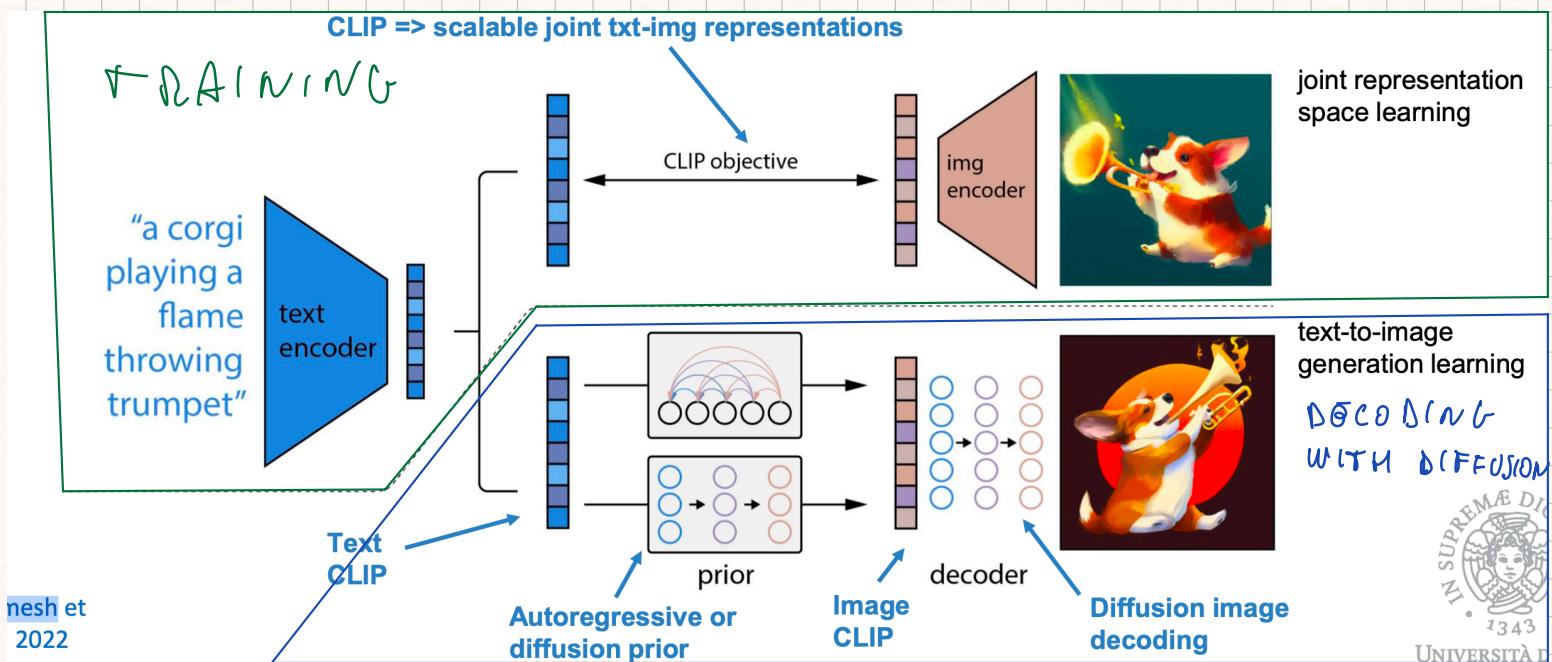
- **Scalar** – vector embedding + spatial addition (or adaptive group normalization)
- **Image** - channel-wise concatenation of the conditional image
- **Text** - vector embedding + spatial addition or cross-attention



### Latent space diffusion



# 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.