Deep Learning II

M. Vazirgiannis

Data Science and Mining Team (DASCIM), LIX
École Polytechnique
http://www.lix.polytechnique.fr/dascim
Google Scholar: https://bit.ly/2rwmvQU
Twitter: @mvazirg

November 2022

DL Architectures overview

Data\Architecture	MLP	CNN	RNN	GNN	Set	Attention
Matrix	X					
Text		X	X	+		+
Sequences/ Time series			X	+		+
Images		X		+		+
Graphs				X		+
Sets				+	X	+

X: Native, +: Transformed

Outline

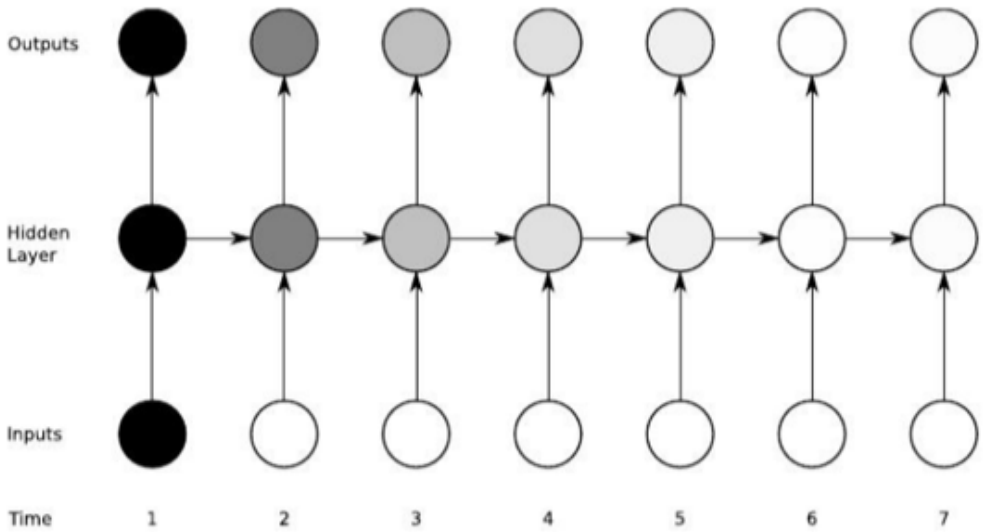
Recurrent Neural Networks

- CNNs are good at image classification
  - Input: an image
  - Output: probability of the class
    - $p(\textit{Red Panda}|\textit{image}) = 0.9$
    - $p(\textit{Cat}|\textit{image}) = 0.1$

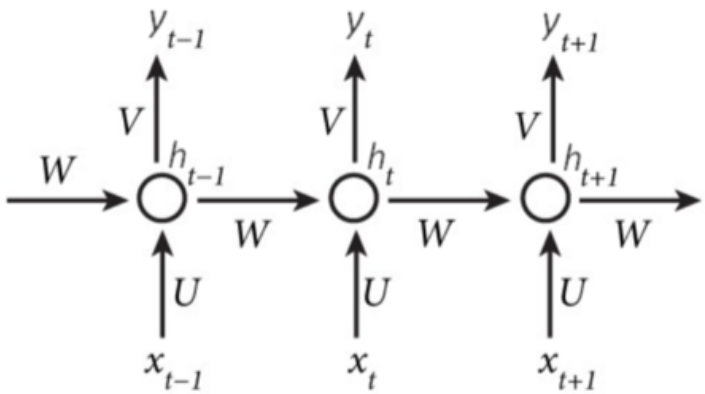
- Sequence learning: study of machine learning algorithms designed for sequential data (time series, language...)
  - Translate: ”machine learning is a challenging topic” to French: ”L’apprentissage automatique est un sujet difficile”
  - Predict the next word: ”John visited Paris, the capital of... ”
  - Input and output strongly correlated within the sequence.

Recurrent Neural Networks

- Update the hidden state in a deterministic nonlinear way.
- RNNs are powerful,
- Distributed hidden state that allows them to store a lot of information about the past efficiently.
- Non-linear dynamics that allows them to update their hidden state in complicated ways.
- No need to infer hidden state, pure deterministic.
- Weight sharing

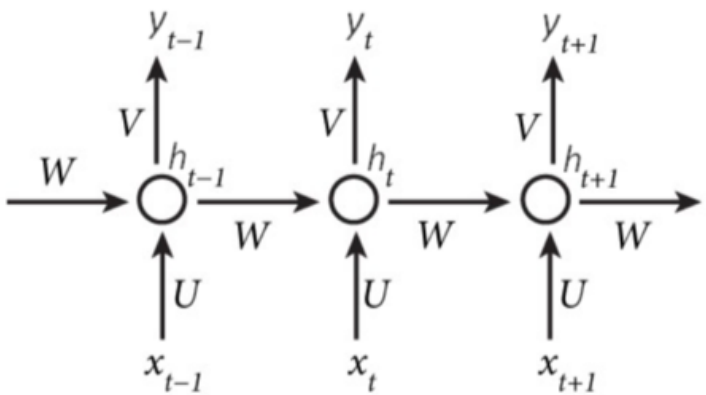


Recurrent Neural Networks



- input: ordered list of input vectors $x_1, \ldots, x_T$ initial hidden state h_0 initialized to all zeros
- output
  - ordered list of hidden states $h_1, \ldots, h_T$
  - ordered list of output vectors $y_1, \ldots, y_T$
  - The output vectors may serve as input for other RNN units, when considering deep architectures
  - The hidden states correspond to the "short-term" memory of the network.
- The last hidden state represents the encoding (embedding) of the time series

Recurrent Neural Networks



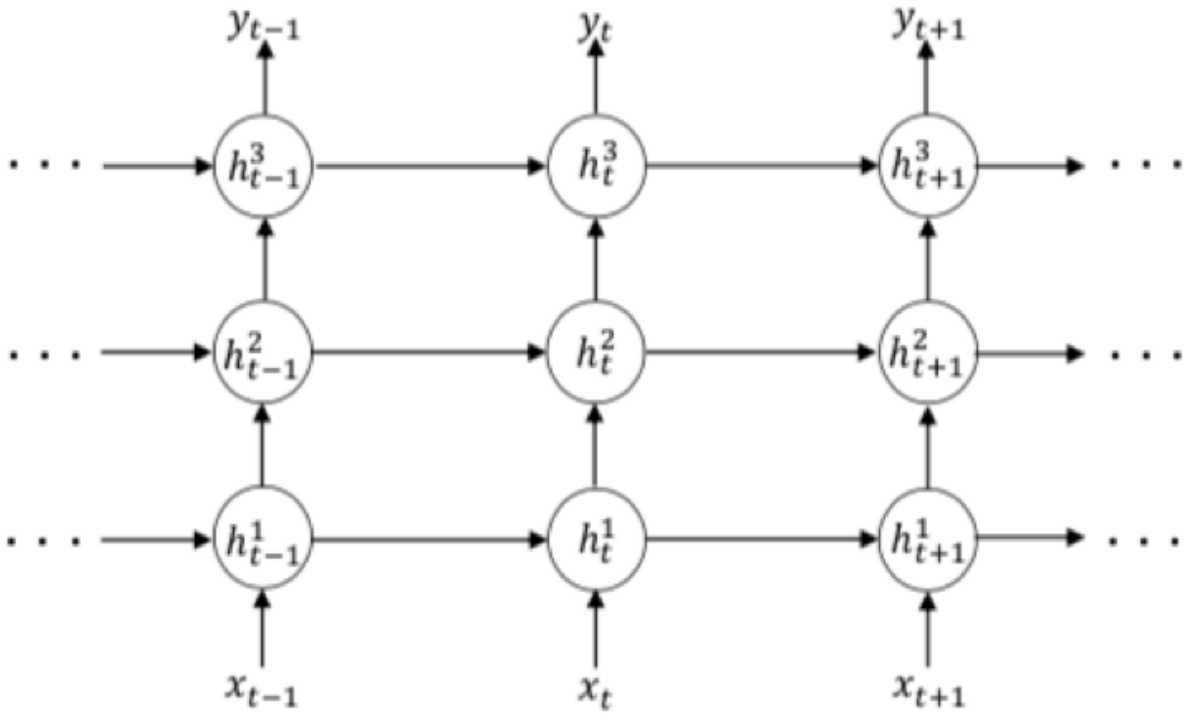
$$h_t = f(Ux_t + Wh_{t-1} + b)$$

- f: a nonlinear function
- $x_t \in \mathbb{R}^{d_{in}}$, $U \in \mathbb{R}^{H \times d_{in}}$, $W \in \mathbb{R}^{H \times H}$,

Parameter matrices

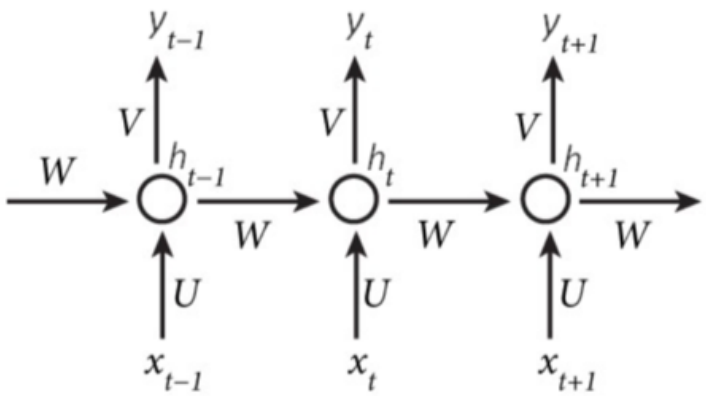
- d_{in}: size of the vocabulary
- H: dimension of the hidden layer ($H \sim 100$)
- $y_t \in \mathbb{R}^{d_{out}}$: transforms the current hidden state h_t to depend on the final task: i.e. for classification $y_t = softmax(Vh_t)$
- $V \in \mathbb{R}^{d_{out} \times H}$: parameter matrix shared across all steps (i.e. for word level language model $d_{out} = |V|$)

Deep RNNs



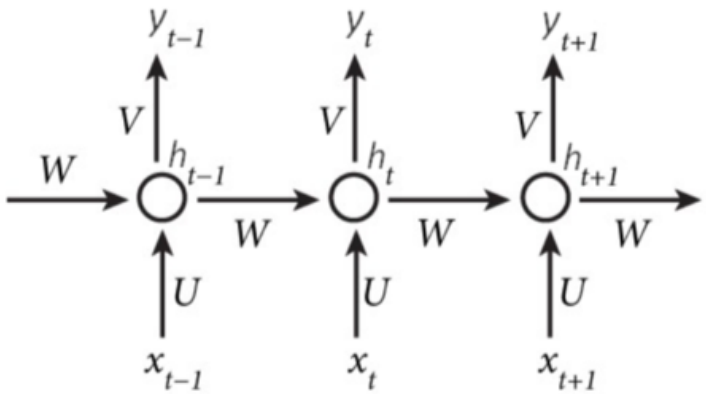
3 steps of an unrooled deep RNN. Each circle represents a RNN unit. The hidden state of each unit in the inner layer (1 & 2) serves as input to the corresponding unit in the layer above.

Recurrent Neural Networks



- CNNs are naturally efficient with *grids*,
- RNNs were specifically developed to be used with sequences
  - time series, or, in NLP, words (sequences of letters) or sentences (sequences of words).
  - language modeling $P[w_n|w_1, \ldots, w_{n-1}]$.
  - RNNs trained with such objectives can be used to generate new and quite convincing sentences from scratch
  - RNN can be considered as a chain of simple neural layers that share the same parameters

Learning in RNNs



- Assuming input
  $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$
- As a single time:

$$h_t = \sigma(Wh_{t-1} + Ux_t)$$

$$y_t = softmax(Vh_t)$$

Learning in RNNs

$$h_t = \sigma(Wh_{t-1} + Ux_t)$$
$$y_t = softmax(Vh_t)$$

- Main idea: we use the same set of W,U,V weights at all time steps!
- $h_0 \in \mathbb{R}^{H \times H}$ initialization vector for the hidden layer at time step 0
- $\hat{y} \in \mathbb{R}^{|V|}$ is a probability distribution over the vocabulary
- Loss function (@ time t): cross entropy predicting words instead of classes

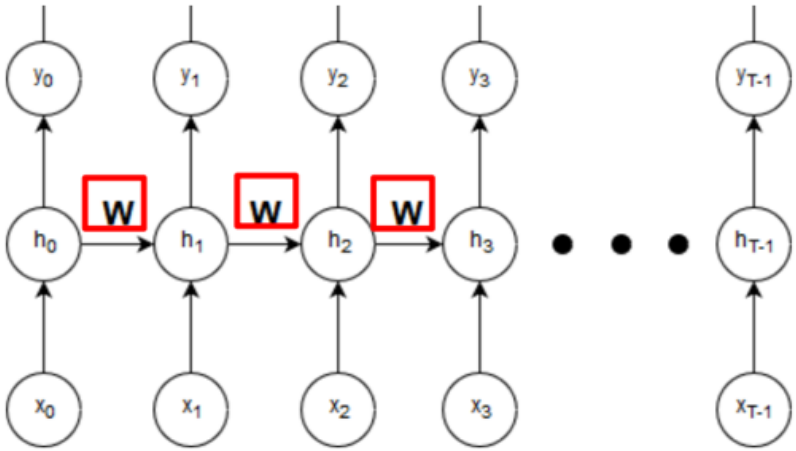
$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} log \hat{y}_{t,j}$$

Learning in RNNs -backpropagation in time

- learning the weigths of **W**:

$$\mathbf{W} \rightarrow \mathbf{W} - \alpha \frac{\partial \mathbf{y}}{\partial \mathbf{W}}$$

- Computation involves summation over all paths regarding
  - i. time $\frac{\partial \mathbf{y}}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \frac{\partial \mathbf{y}_j}{\partial \mathbf{W}}$
  - ii. Levels of hidden layers $\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$



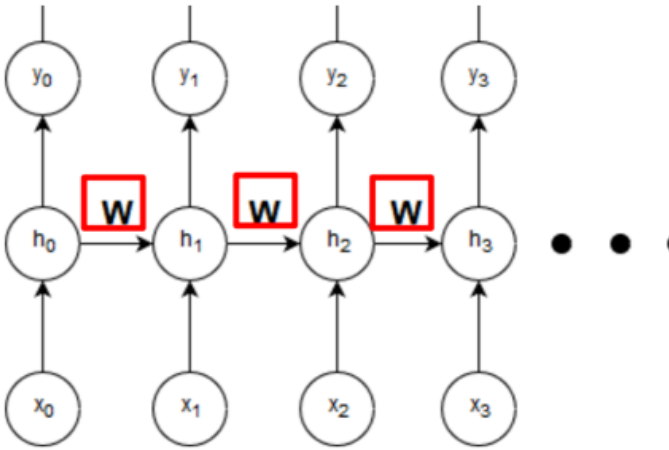
Learning in RNNs - backpropagation in time

- Therefore:

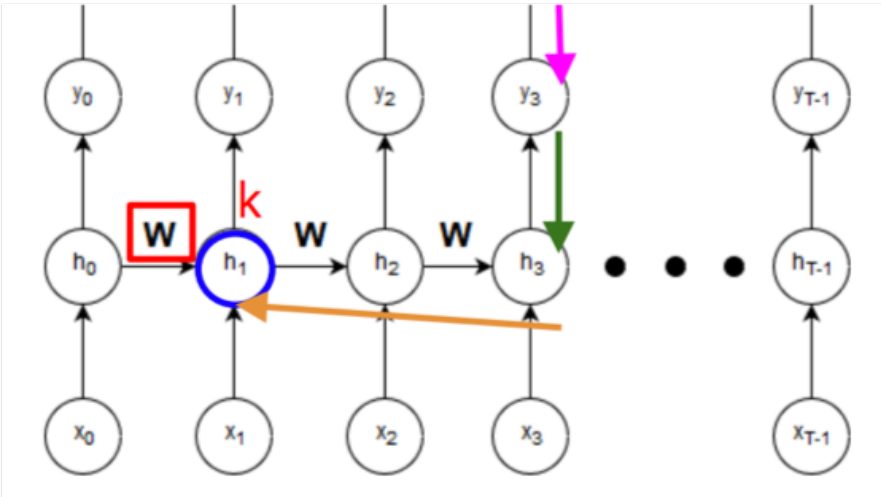
$$\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

- Indirect dependency - use of the chain rule:

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}}$$



Learning in RNNs - back propagation in time



$$\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W}}$$

Learning in RNNs - vanishing/exploding gradients

$$\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W}}$$

$$h_m = f(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)$$

$$\frac{\partial h_m}{\partial h_{m-1}} = \mathbf{W}_h^T \, diag(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m))$$

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \mathbf{W}_h^T \, \mathrm{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

Weight Matrix

Derivative of activation function

Repeated matrix multiplications leads to vanishing and exploding gradients

Learning in RNNs - vanishing/exploding gradients

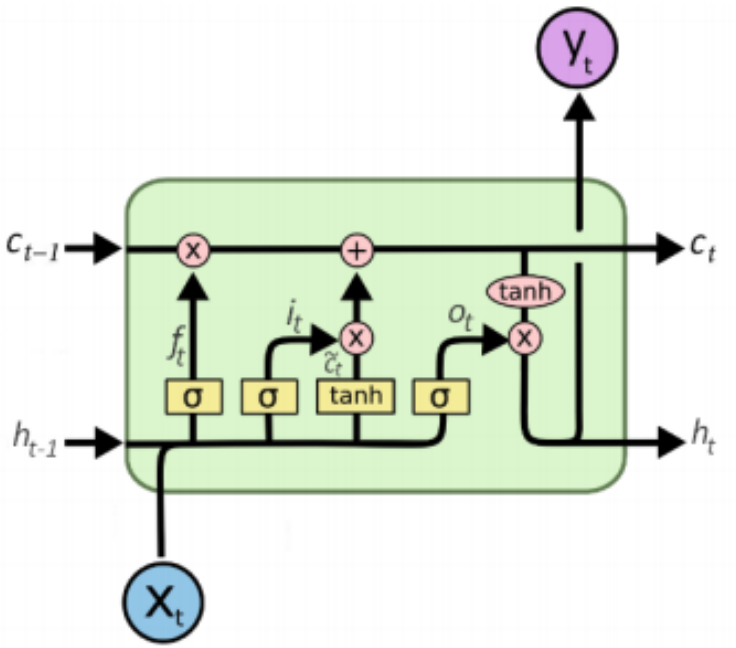
- Initialization of W with 1s
- Using relu as activation function f(z)= rect(z) = max(z, 0)
- Using - clip gradients to a maximum value [Mikolov]

Algorithm 1: Pseudo-code for norm clipping the gradients whenever they explode

---

$\hat{\mathbf{g}} \leftarrow \frac{\partial \xi}{\partial \theta}$;
if $||\hat{\boldsymbol{g}}|| \geq$ *threshold* **then**
  $\hat{\mathbf{g}} \leftarrow \frac{\textit{threshold}}{||\hat{\mathbf{g}}||}\hat{\mathbf{g}}$;
end

LSTM units

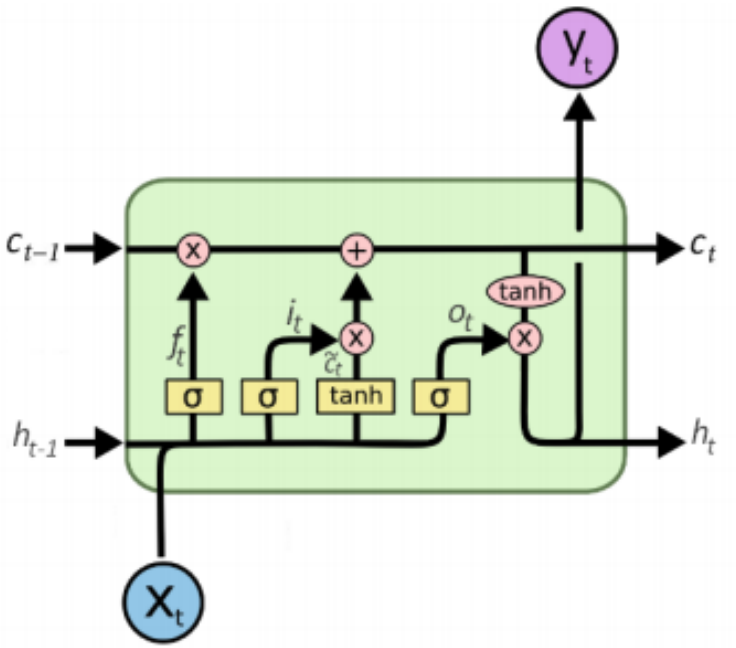


The LSTM unit. Adapted from Chris Colah's blog

1. forget gate layer: $f_t = \sigma(U_f x_t + W_f h_{t-1} + b_f)$
2. input gate layer: $i_t = \sigma(U_i x_t + W_i h_{t-1} + b_i)$
3. candidate values computation layer: $\tilde{c}_t = tanh(U_c x_t + W_c h_{t-1} + b_c)$
4. output gate layer: $o_t = \sigma(U_o x_t + W_o h_{t-1} + b_o)$

- To tackle RNN problems of i. vanishing gradients and ii. Keep track of information for longer term.
- There is a "memory bus" C_t on which we chose how much to write and read

LSTM units



The LSTM unit. Adapted from Chris Colah's blog

1. forget gate layer: $f_t = \sigma(U_f x_t + W_f h_{t-1} + b_f)$
2. input gate layer: $i_t = \sigma(U_i x_t + W_i h_{t-1} + b_i)$
3. candidate values computation layer: $\tilde{c}_t = tanh(U_c x_t + W_c h_{t-1} + b_c)$
4. output gate layer: $o_t = \sigma(U_o x_t + W_o h_{t-1} + b_o)$

Assume a new training example x_t and the current hidden state h_{t-1},

- forget gate layer ft determines how much of the previous cell state c_{t-1} should be forgotten (what fraction of the memory should be freed up),
- input gate layer decides how much of the candidate values $\tilde{c}_t$ should be written to the memory: how much of the new information should be learned. Combining the output of the two filters updates the cell state:
  - $c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$
  - $h_t = tanh(c_t) \circ o_t$
  - $y_t = softmax(Vh_t)$

Outline

Autoencoders - the concept

- introduced by G. Hinton
  - address the problem of “backpropagation without a teacher”
  - Need to formulate an error function – using input data as the teacher [RUM1986]
- more recently, “deep architecture” approach (Hinton et al., 2006; Hinton and Salakhutdinov, 2006; Bengio and LeCun, 2007;Erhan et al., 2010)
  - Restricted Boltzmann Machines (RBMS), stacked, trained bottom up in unsupervised fashion.

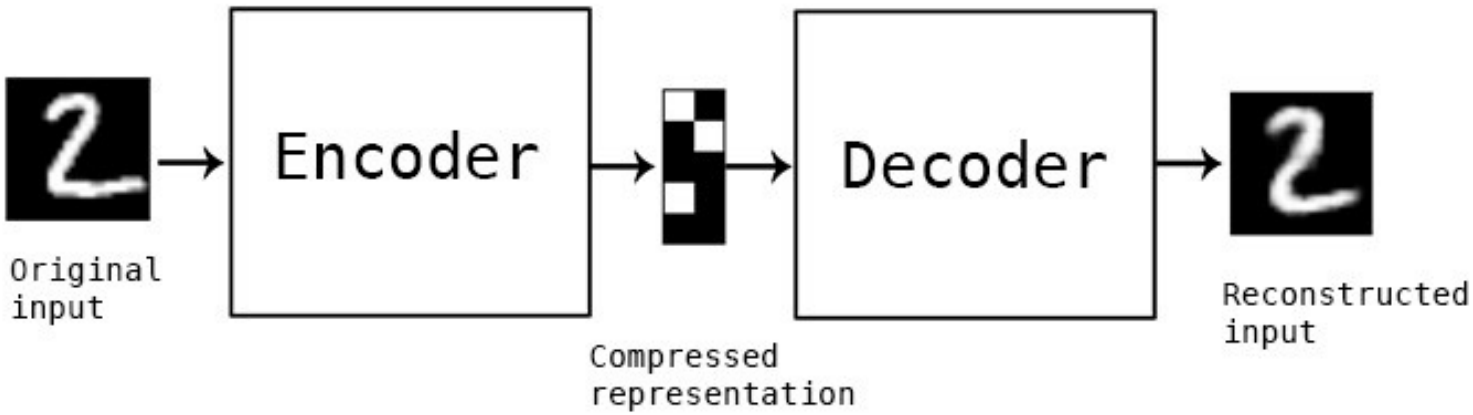
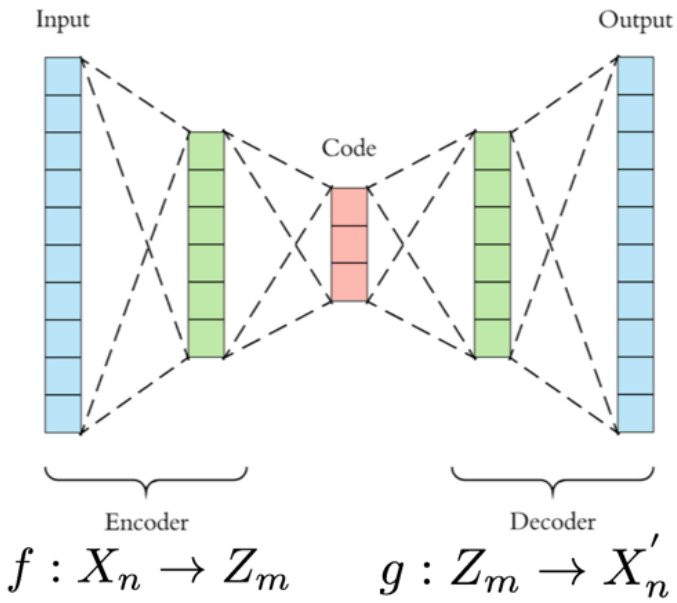


image from: https://medium.com/@syoya/what-happens-in-sparse-autencoder-b9a5a69da5c6

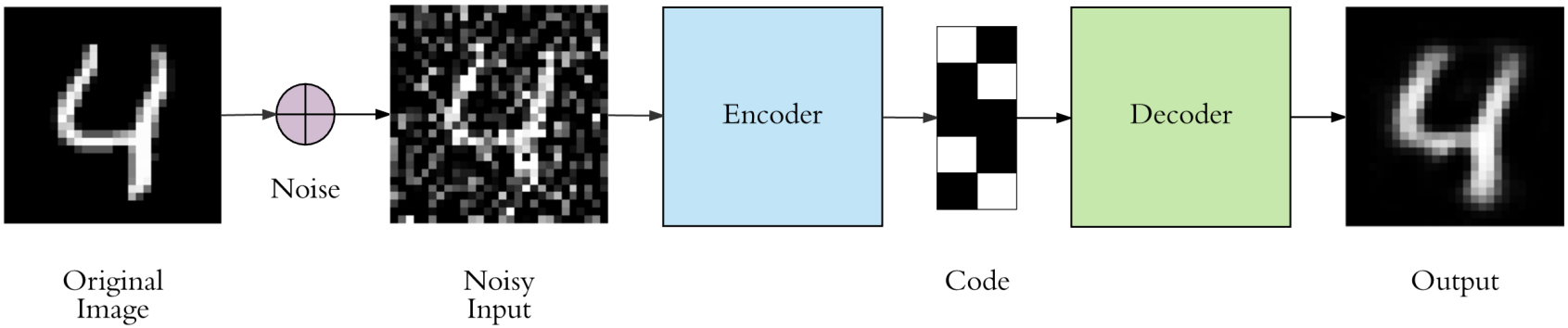
Autoencoders

- Assume a set of n-dim data X_n - we assume transformations f, g
  - f : mapping data to a lower dim space Z_m (code)
  - g: mapping Z_m to the an original dim data $X'_n \sim X_n$
- Learning: minimize the loss $L(x(f(g(x)))$
  - Sparse autoencoders: $L(x(f(g(x)) + \Omega(Z_m)$



Denoising Autoencoders

- input data $\longrightarrow$ corrupted (e.g. noisy) version
- still train to reconstruct original data



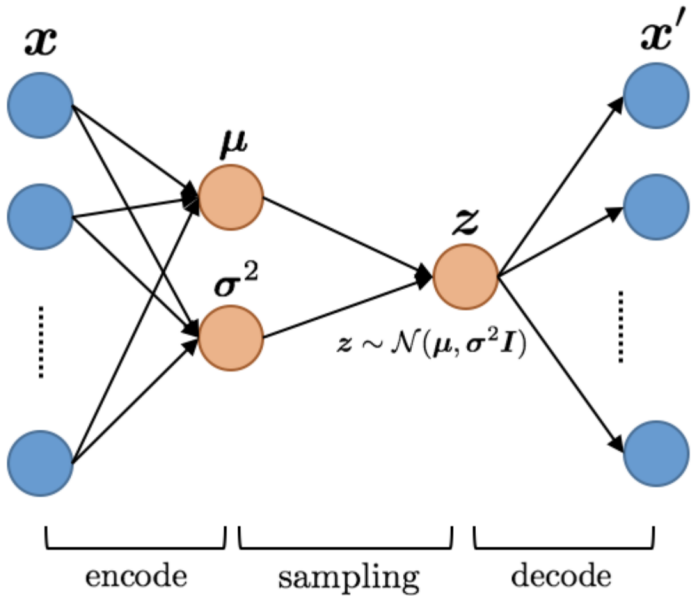
Denoising Autoencoders on noisy handwritten digits [1]

[1] Image from [Dertat, 2017]

Variational Autoencoders (VAE)

Variational Autoencoders (VAE) [kingma2013vae]:

- **VAE assume that the latent code z is a sample drawn from some distribution**
- Usually, $z \sim \mathcal{N}(\mu, \sigma I_d)$, a multidimensional **Gaussian** distribution
- Instead of learning z, the NN encoder aims at learning μ and Σ



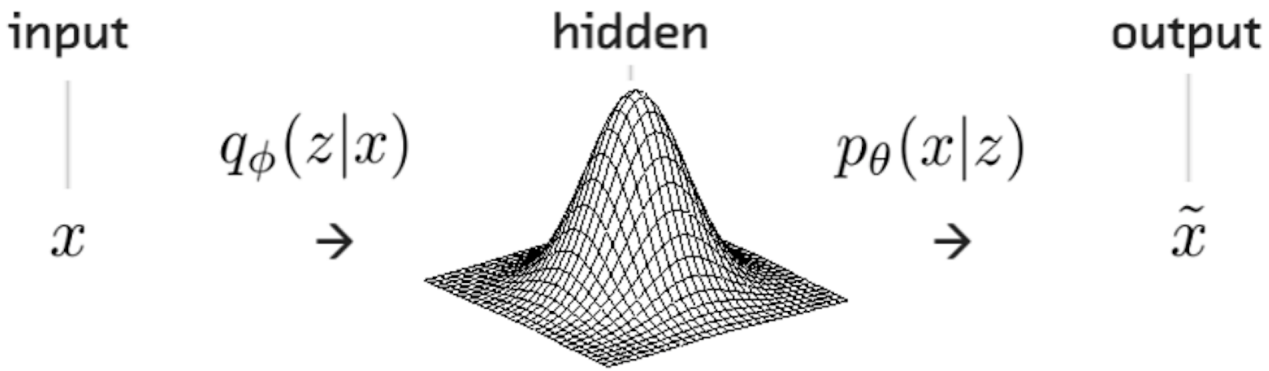
Variational Autoencoders with Gaussian hypotheses [2]

[2] Image from [Dehaene, 2018]

Variational Autoencoders (VAE)

In addition to learning embedding representation, VAE are **generative models**

- Idea: sample new data from the previously learned distributions



Learning: to tune the encoder and decoder weights:

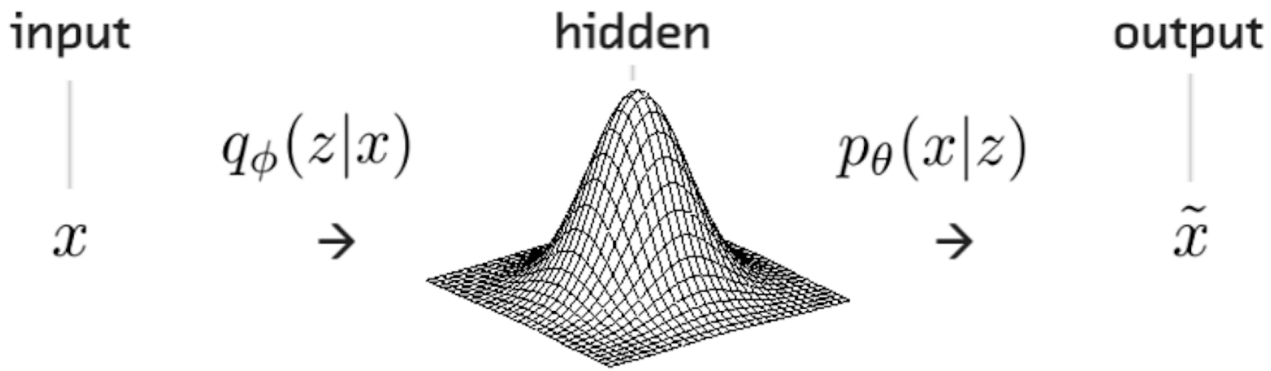
- we would like to maximize the likelihood $p(x)$ of the model, but it is usually untractable [**?**], [**?**]
- instead, we iteratively maximize a **lower bound of the model's likelihood** (called the Evidence Lower Bound or **ELBO**), by gradient descent

$$\log p(x) \geq \mathbb{E}_{z\sim q}\big[\log p(x|z)\big] - \mathcal{D}_{KL}\big[q(z|x) \parallel p(z)\big] = ELBO$$

Variational Autoencoders (VAE)

In addition to learning embedding representation, VAE are **generative models**

- Idea: sample new data from the previously learned distributions

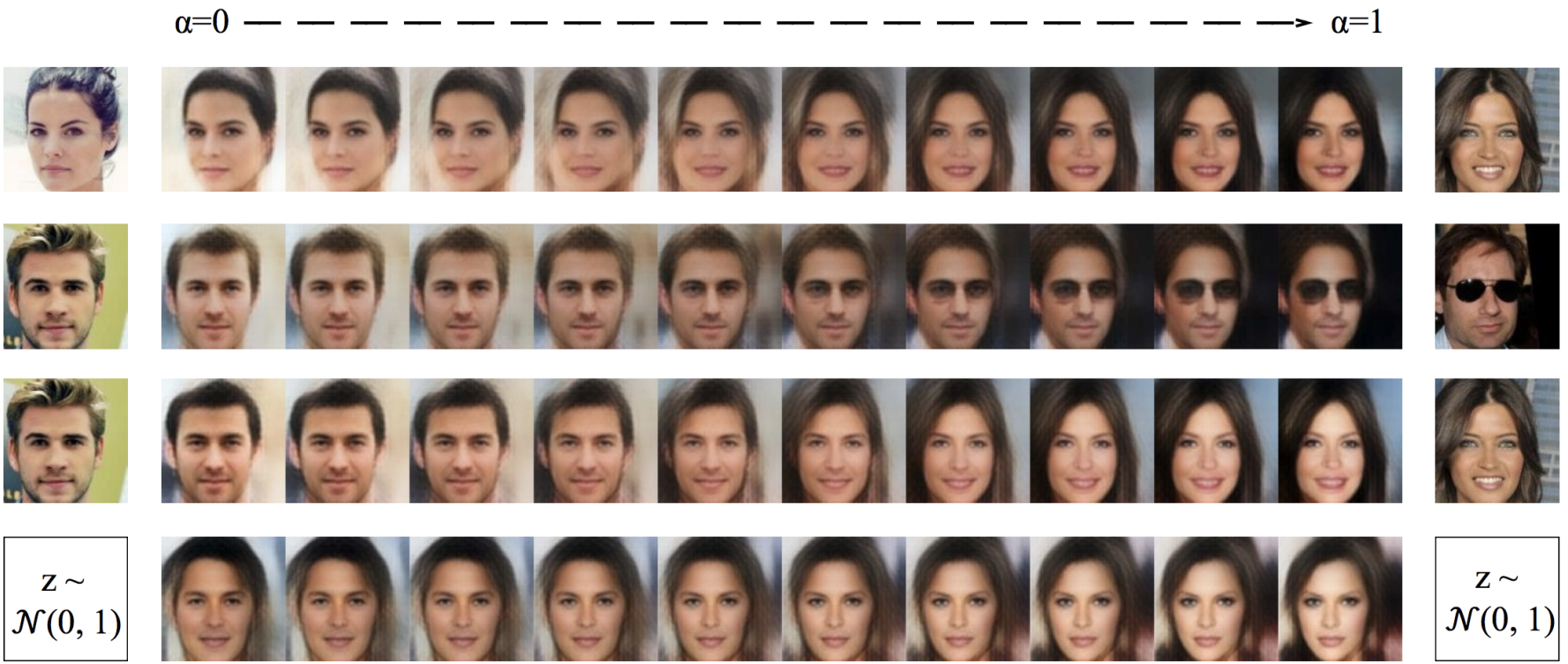


Learning: to tune the encoder and decoder weights:

- we would like to maximize the likelihood $p(x)$ of the model, but it is usually untractable [**?**], [**?**]
- instead, we iteratively maximize a **lower bound of the model's likelihood** (called the Evidence Lower Bound or **ELBO**), by gradient descent

$$\log p(x) \geq \mathbb{E}_{z\sim q}\big[\log p(x|z)\big] - \mathcal{D}_{KL}\big[q(z|x) \parallel p(z)\big] = ELBO$$

VAE - Applications



Embedding faces with VAE [**?**] and linear interpolation from left latent vector z_{left} to right vector z_{right} i.e. $(1-\alpha)z_{left} + \alpha z_{right}$.

Applications of AEs

- Learn a low dimensional Z_m representation of the data X
  - learning low dimensional space with less reconstruction error than PCA (Hinton 2006)
  - Features used in classification and improve generalization (Hinton 2007)
- Perform clustering – unsupervised learning
- Information Retrieval
  - Semantic hashing for images and NLP (Hinton 2007, Torralba 2007)
- Learn data features in the absence of a supervised task

Auto-encoders for clustering [Xie et al 2016]

- Student's t-distribution used as a kernel to measure the similarity (α : degrees of freedom) between embedded point z_i and centroid μ_j

$$q_{ij} = \frac{(1 + ||z_i - \mu_j||^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j'} (1 + ||z_i - \mu_{j'}||^2/\alpha)^{-\frac{\alpha+1}{2}}}$$

- iteratively refine clusters by learning from their high confidence assignments with the help of an auxiliary target distribution

$$p_{ij} = \frac{q_{ij}^2/f_j}{\sum_{j'} q_{ij'}^2/f_{j'}}$$

where $f_j = \sum_i q_{ij}$ are the soft cluster frequencies.

- Error function: $L = KL(P||Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}}$

Auto-encoders for clustering [Xie te al 2016]

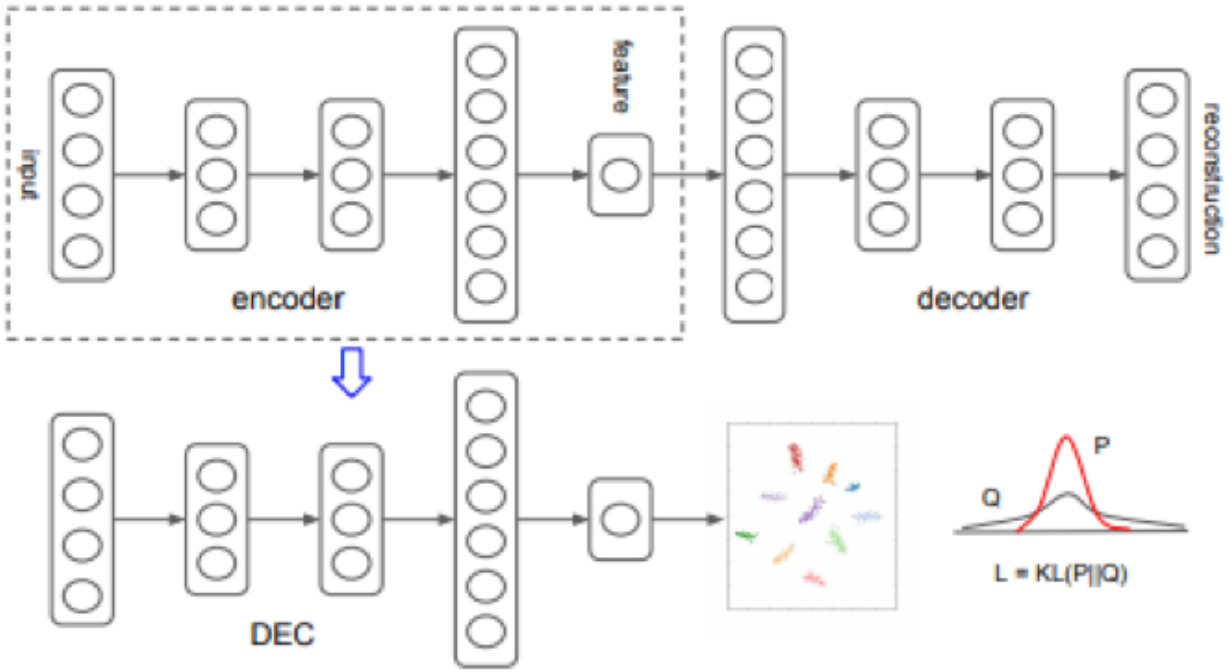
- jointly optimize cluster centers $\{\mu_j\}$ and DNN parameters θ using Stochastic Gradient Descent (SGD):

$$\frac{\partial L}{\partial z_i} = \frac{\alpha+1}{\alpha} \sum_j (1 + \frac{||z_i - \mu_j||^2}{\alpha})^{-1} \times (p_{ij} - q_{ij})(z_i - \mu_j)$$

$$\frac{\partial L}{\partial \mu_j} = -\frac{\alpha+1}{\alpha} \sum_i (1 + \frac{||z_i - \mu_j||^2}{\alpha})^{-1} \times (p_{ij} - q_{ij})(z_i - \mu_j)$$

- gradients $\frac{\partial L}{\partial z_i}$ are passed down to the DNN
- gstandard backpropagation compute DNN's parameter gradient $\frac{\partial L}{\partial \theta}$
- discovering cluster assignments, stop when less than tol % of points change cluster assignment between consecutive iterations.

Auto-encoders for clustering [Xie et al 2016]



To initialize the cluster centers:

- Pass data through the initialized DNN to get embedded data points and then perform standard k-means clustering in the feature space Z to obtain k initial centroids $\{\mu_j\}_{j=1}^{k}$.

Auto-encoders for clustering [Xie et al 2016]

Dataset	# Points	# classes	Dimension	% of largest class
MNIST (LeCun et al., 1998)	70000	10	784	11%
STL-10 (Coates et al., 2011)	13000	10	1428	10%
REUTERS-10K	10000	4	2000	43%
REUTERS (Lewis et al., 2004)	685071	4	2000	43%

Data Set features

Method	MNIST	STL-HOG	REUTERS-10k	REUTERS
k-means	53.49%	28.39%	52.42%	53.29%
LDMGI	84.09%	33.08%	43.84%	N/A
SEC	80.37%	30.75%	60.08%	N/A
DEC w/o backprop	79.82%	34.06%	70.05%	69.62%
DEC (ours)	**84.30%**	**35.90%**	**72.17%**	**75.63%**

Clustering Accuracy

Auto-encoders for clustering [Xie et al 2016]

(a) MNIST

(b) STL-10

Top 10 results for each of the 10 clusers for the MNIST and STL data sets

Why is Unsupervised learning important

- Machine learning: learning representations (features, latent variables) to capture the statistical dependencies
- Supervised learning: from input variables to output variables, – i.e. categories
  - supervised learning require ”teaching” computer concepts that matter to humans
  - supervised deep learning discover meaningful intermediate representations in their layers.
- Unsupervised learning
  - capture all possible dependencies between any subset of variables
  - human learning: based on observation and unsupervised interaction (action-perception loop)
- Issues
  - Training sets increasingly difficult to keep the pace with data production
  - Can we trust humans ?

“. . . hope is that deep unsupervised learning will be able to discover (possibly with a little bit of help from the few labeled examples we can provide) all of the concepts and underlying causes that matter (some being explicitly labeled, some remaining unnamed) to explain what we see around us. So I believe it is essential for approaching AI to make progress in this direction. And we are ;-)” Y. Bengio