

# Super VIP Cheatsheet: Deep Learning

Afshine AMIDI and Shervine AMIDI

November 25, 2018

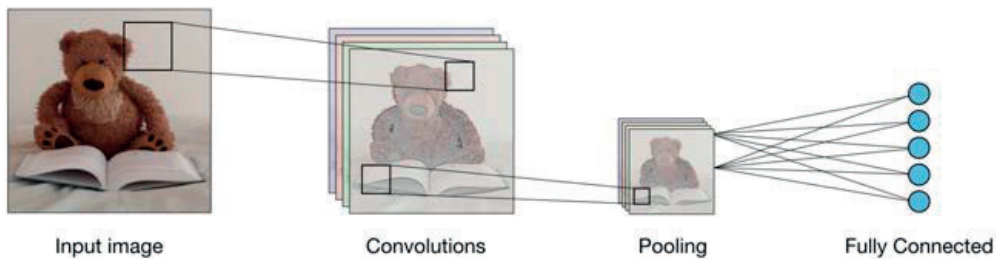
## Contents

<b>1</b>	<b>Convolutional Neural Networks</b>	<b>2</b>
1.1	Overview . . . . .	
1.2	Types of layer . . . . .	
1.3	Filter hyperparameters . . . . .	
1.4	Tuning hyperparameters . . . . .	
1.5	Commonly used activation functions . . . . .	
1.6	Object detection . . . . .	
1.6.1	Face verification and recognition . . . . .	
1.6.2	Neural style transfer . . . . .	
1.6.3	Architectures using computational tricks . . . . .	
<b>2</b>	<b>Recurrent Neural Networks</b>	<b>12</b>
2.1	Overview . . . . .	
2.2	Handling long term dependencies . . . . .	
2.3	Learning word representation . . . . .	
2.3.1	Motivation and notations . . . . .	
2.3.2	Word embeddings . . . . .	
2.4	Comparing words . . . . .	
2.5	Language model . . . . .	
2.6	Machine translation . . . . .	
2.7	Attention . . . . .	
<b>3</b>	<b>Deep Learning Tips and Tricks</b>	<b>21</b>
3.1	Data processing . . . . .	
3.2	Training a neural network . . . . .	
3.2.1	Definitions . . . . .	
3.2.2	Finding optimal weights . . . . .	
3.3	Parameter tuning . . . . .	
3.3.1	Weights initialization . . . . .	
3.3.2	Optimizing convergence . . . . .	
3.4	Regularization . . . . .	
3.5	Good practices . . . . .	

## 1 Convolutional Neural Networks

### 1.1 Overview

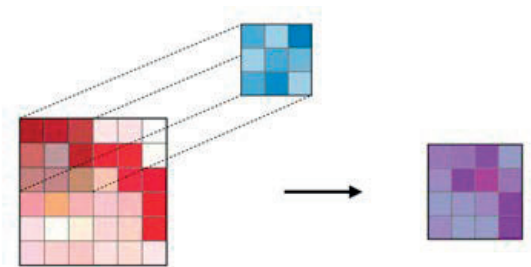
□ **Architecture of a traditional CNN** – Convolutional neural networks, also known as CNNs, are a specific type of neural networks that are generally composed of the following layers:



The convolution layer and the pooling layer can be fine-tuned with respect to hyperparameters that are described in the next sections.

### 1.2 Types of layer

□ **Convolutional layer (CONV)** – The convolution layer (CONV) uses filters that perform convolution operations as it is scanning the input  $I$  with respect to its dimensions. Its hyperparameters include the filter size  $F$  and stride  $S$ . The resulting output  $O$  is called *feature map* or *activation map*.

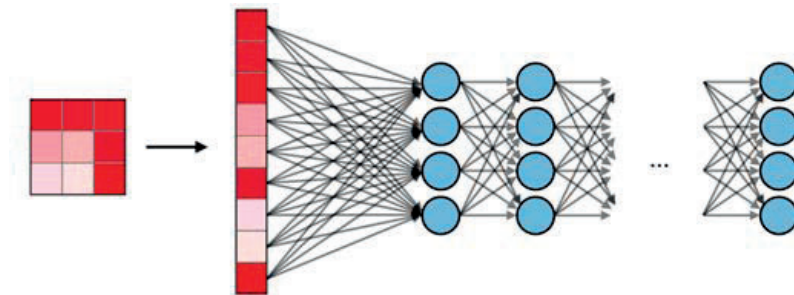


*Remark: the convolution step can be generalized to the 1D and 3D cases as well.*

□ **Pooling (POOL)** – The pooling layer (POOL) is a downsampling operation, typically applied after a convolution layer, which does some spatial invariance. In particular, max and average pooling are special kinds of pooling where the maximum and average value is taken, respectively.

	Max pooling	Average pooling
<b>Purpose</b>	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
<b>Illustration</b>		
<b>Comments</b>	<ul style="list-style-type: none"> <li>- Preserves detected features</li> <li>- Most commonly used</li> </ul>	<ul style="list-style-type: none"> <li>- Downsamples feature map</li> <li>- Used in LeNet</li> </ul>

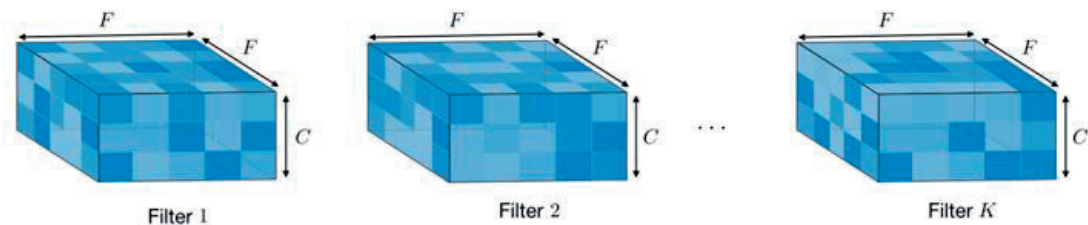
□ **Fully Connected (FC)** – The fully connected layer (FC) operates on a flattened input where each input is connected to all neurons. If present, FC layers are usually found towards the end of CNN architectures and can be used to optimize objectives such as class scores.



### 1.3 Filter hyperparameters

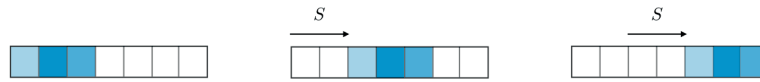
The convolution layer contains filters for which it is important to know the meaning behind its hyperparameters.

□ **Dimensions of a filter** – A filter of size  $F \times F$  applied to an input containing  $C$  channels is a  $F \times F \times C$  volume that performs convolutions on an input of size  $I \times I \times C$  and produces an output feature map (also called activation map) of size  $O \times O \times 1$ .



*Remark: the application of  $K$  filters of size  $F \times F$  results in an output feature map of size  $O \times O \times K$ .*

□ **Stride** – For a convolutional or a pooling operation, the stride  $S$  denotes the number of pixels by which the window moves after each operation.



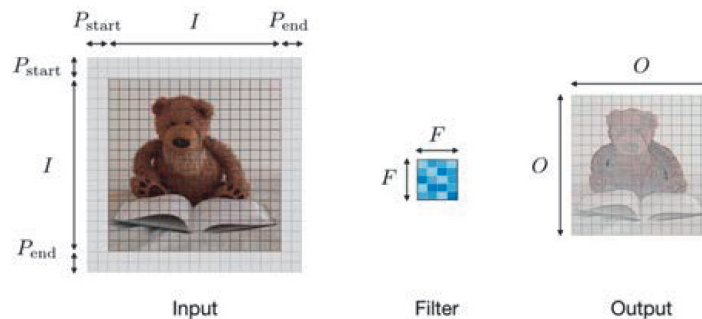
□ **Zero-padding** – Zero-padding denotes the process of adding  $P$  zeroes to each side of the boundaries of the input. This value can either be manually specified or automatically set through one of the three modes detailed below:

	Valid	Same	Full
Value	$P = 0$	$P_{\text{start}} = \left\lfloor \frac{S \lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rfloor$ $P_{\text{end}} = \left\lceil \frac{S \lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rceil$	$P_{\text{start}} \in \llbracket 0, F - 1 \rrbracket$ $P_{\text{end}} = F - 1$
Illustration			
Purpose	<ul style="list-style-type: none"> <li>- No padding</li> <li>- Drops last convolution if dimensions do not match</li> </ul>	<ul style="list-style-type: none"> <li>- Padding such that feature map size has size <math>\lceil \frac{I}{S} \rceil</math></li> <li>- Output size is mathematically convenient</li> <li>- Also called 'half' padding</li> </ul>	<ul style="list-style-type: none"> <li>- Maximum padding such that end convolutions are applied on the limits of the input</li> <li>- Filter 'sees' the input end-to-end</li> </ul>

### 1.4 Tuning hyperparameters

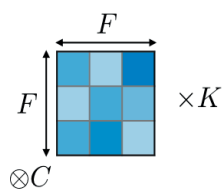
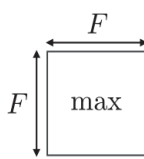
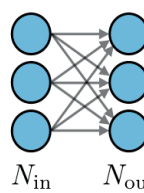
□ **Parameter compatibility in convolution layer** – By noting  $I$  the length of the input volume size,  $F$  the length of the filter,  $P$  the amount of zero padding,  $S$  the stride, then the output size  $O$  of the feature map along that dimension is given by:

$$O = \frac{I - F + P_{\text{start}} + P_{\text{end}}}{S} + 1$$



*Remark: often times,  $P_{\text{start}} = P_{\text{end}} \triangleq P$ , in which case we can replace  $P_{\text{start}} + P_{\text{end}}$  by  $2P$  in the formula above.*

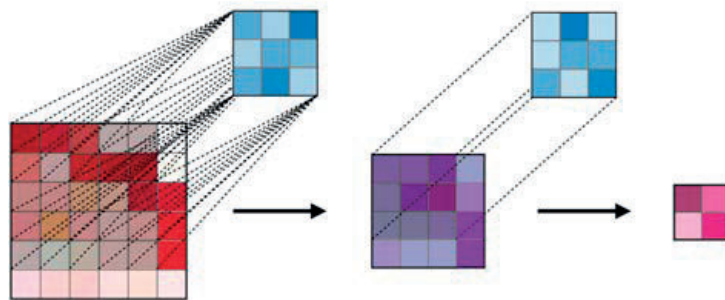
□ **Understanding the complexity of the model** – In order to assess the complexity of a model, it is often useful to determine the number of parameters that its architecture will have. In a given layer of a convolutional neural network, it is done as follows:

	CONV	POOL	FC
Illustration			
Input size	$I \times I \times C$	$I \times I \times C$	$N_{in}$
Output size	$O \times O \times K$	$O \times O \times C$	$N_{out}$
Number of parameters	$(F \times F \times C + 1) \cdot K$	0	$(N_{in} + 1) \times N_{out}$
Remarks	<ul style="list-style-type: none"> <li>- One bias parameter per filter</li> <li>- In most cases, <math>S &lt; F</math></li> <li>- A common choice for <math>K</math> is <math>2C</math></li> </ul>	<ul style="list-style-type: none"> <li>- Pooling operation done channel-wise</li> <li>- In most cases, <math>S = F</math></li> </ul>	<ul style="list-style-type: none"> <li>- Input is flattened</li> <li>- One bias parameter per neuron</li> <li>- The number of FC neurons is free of structural constraints</li> </ul>

□ **Receptive field** – The receptive field at layer  $k$  is the area denoted  $R_k \times R_k$  of the input that each pixel of the  $k$ -th activation map can 'see'. By calling  $F_j$  the filter size of layer  $j$  and  $S_i$  the stride value of layer  $i$  and with the convention  $S_0 = 1$ , the receptive field at layer  $k$  can be computed with the formula:

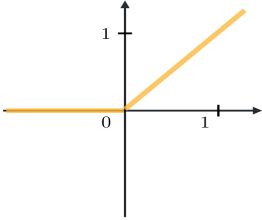
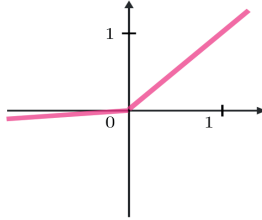
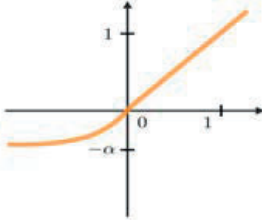
$$R_k = 1 + \sum_{j=1}^k (F_j - 1) \prod_{i=0}^{j-1} S_i$$

In the example below, we have  $F_1 = F_2 = 3$  and  $S_1 = S_2 = 1$ , which gives  $R_2 = 1+2 \cdot 1+2 \cdot 1 = 5$ .



### 1.5 Commonly used activation functions

□ **Rectified Linear Unit** – The rectified linear unit layer (ReLU) is an activation function  $g$  that is used on all elements of the volume. It aims at introducing non-linearities to the network. Its variants are summarized in the table below:


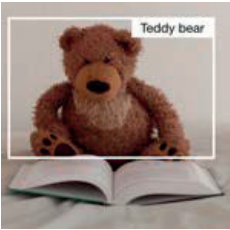
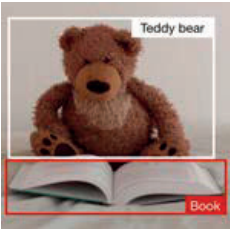
ReLU	Leaky ReLU	ELU
$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$	$g(z) = \max(\alpha(e^z - 1), z)$ with $\alpha \ll 1$
		
Non-linearity complexities biologically interpretable	Addresses dying ReLU issue for negative values	Differentiable everywhere

□ **Softmax** – The softmax step can be seen as a generalized logistic function that takes as input a vector of scores  $x \in \mathbb{R}^n$  and outputs a vector of output probability  $p \in \mathbb{R}^n$  through a softmax function at the end of the architecture. It is defined as follows:


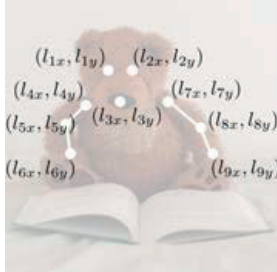
$$p = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \quad \text{where} \quad p_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

### 1.6 Object detection

□ **Types of models** – There are 3 main types of object recognition algorithms, for which the nature of what is predicted is different. They are described in the table below:

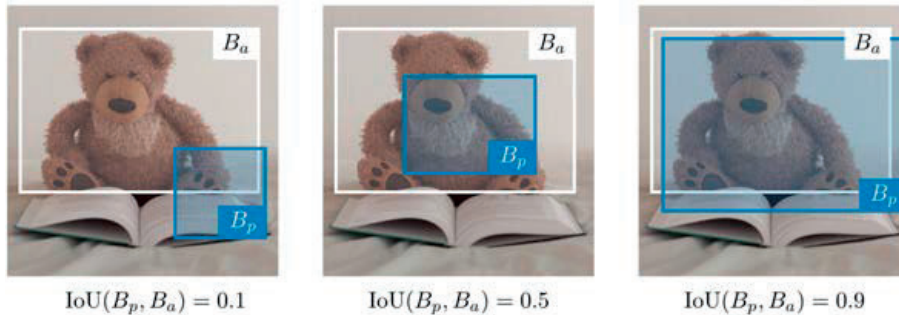
Image classification	Classification w. localization	Detection
		
<ul style="list-style-type: none"> <li>- Classifies a picture</li> <li>- Predicts probability of object</li> </ul>	<ul style="list-style-type: none"> <li>- Detects object in a picture</li> <li>- Predicts probability of object and where it is located</li> </ul>	<ul style="list-style-type: none"> <li>- Detects up to several objects in a picture</li> <li>- Predicts probabilities of objects and where they are located</li> </ul>
Traditional CNN	Simplified YOLO, R-CNN	YOLO, R-CNN

□ **Detection** – In the context of object detection, different methods are used depending on whether we just want to locate the object or detect a more complex shape in the image. The two main ones are summed up in the table below:

Bounding box detection	Landmark detection
Detects the part of the image where the object is located	- Detects a shape or characteristics of an object (e.g. eyes) - More granular
	
Box of center $(b_x, b_y)$ , height $b_h$ and width $b_w$	Reference points $(l_{1x}, l_{1y}), \dots, (l_{nx}, l_{ny})$

□ **Intersection over Union** – Intersection over Union, also known as IoU, is a function that quantifies how correctly positioned a predicted bounding box  $B_p$  is over the actual bounding box  $B_a$ . It is defined as:

$$\text{IoU}(B_p, B_a) = \frac{B_p \cap B_a}{B_p \cup B_a}$$

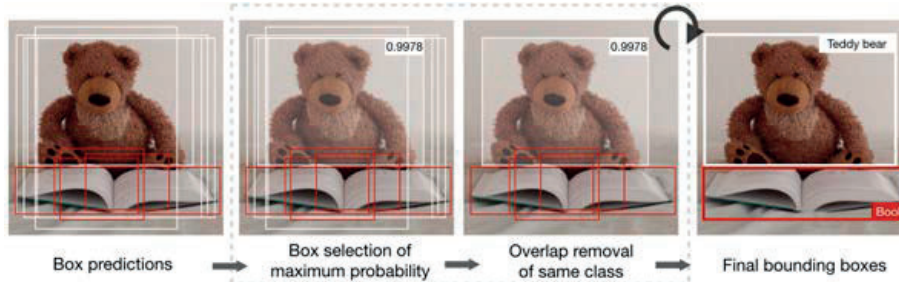


*Remark: we always have  $\text{IoU} \in [0,1]$ . By convention, a predicted bounding box  $B_p$  is considered as being reasonably good if  $\text{IoU}(B_p, B_a) \geq 0.5$ .*

□ **Anchor boxes** – Anchor boxing is a technique used to predict overlapping bounding boxes. In practice, the network is allowed to predict more than one box simultaneously, where each box prediction is constrained to have a given set of geometrical properties. For instance, the first prediction can potentially be a rectangular box of a given form, while the second will be another rectangular box of a different geometrical form.

□ **Non-max suppression** – The non-max suppression technique aims at removing duplicate overlapping bounding boxes of a same object by selecting the most representative ones. After having removed all boxes having a probability prediction lower than 0.6, the following steps are repeated while there are boxes remaining:

- Step 1: Pick the box with the largest prediction probability.
- Step 2: Discard any box having an  $\text{IoU} \geq 0.5$  with the previous box.



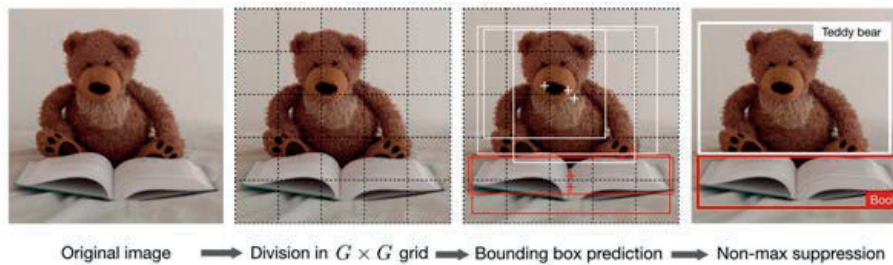
□ **YOLO** – You Only Look Once (YOLO) is an object detection algorithm that performs the following steps:

- Step 1: Divide the input image into a  $G \times G$  grid.
- Step 2: For each grid cell, run a CNN that predicts  $y$  of the following form:

$$y = \left[ \underbrace{p_c, b_x, b_y, b_h, b_w, c_1, c_2, \dots, c_p}_{\text{repeated } k \text{ times}}, \dots \right]^T \in \mathbb{R}^{G \times G \times k \times (5+p)}$$

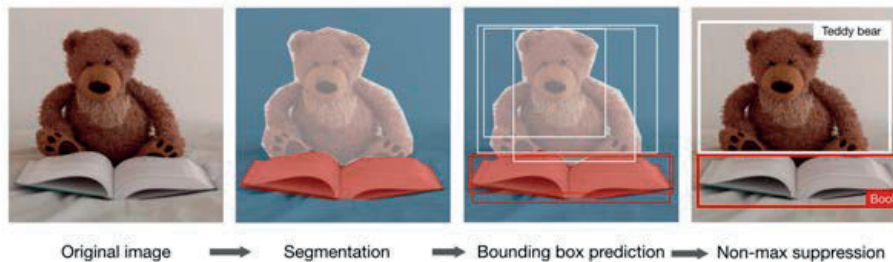
where  $p_c$  is the probability of detecting an object,  $b_x, b_y, b_h, b_w$  are the properties of the detected bounding box,  $c_1, \dots, c_p$  is a one-hot representation of which of the  $p$  classes were detected, and  $k$  is the number of anchor boxes.

- Step 3: Run the non-max suppression algorithm to remove any potential duplicate overlapping bounding boxes.



*Remark: when  $p_c = 0$ , then the network does not detect any object. In that case, the corresponding predictions  $b_x, \dots, c_p$  have to be ignored.*

□ **R-CNN** – Region with Convolutional Neural Networks (R-CNN) is an object detection algorithm that first segments the image to find potential relevant bounding boxes and then run the detection algorithm to find most probable objects in those bounding boxes.

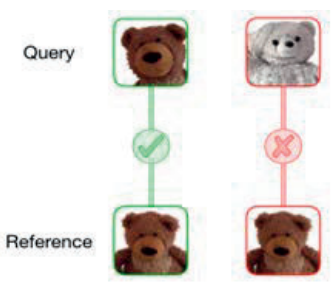
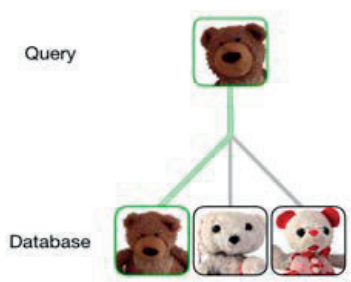


*Remark: although the original algorithm is computationally expensive and slow, newer architectures enabled the algorithm to run faster, such as Fast R-CNN and Faster R-CNN.*



### 1.6.1 Face verification and recognition

□ **Types of models** – Two main types of model are summed up in table below:

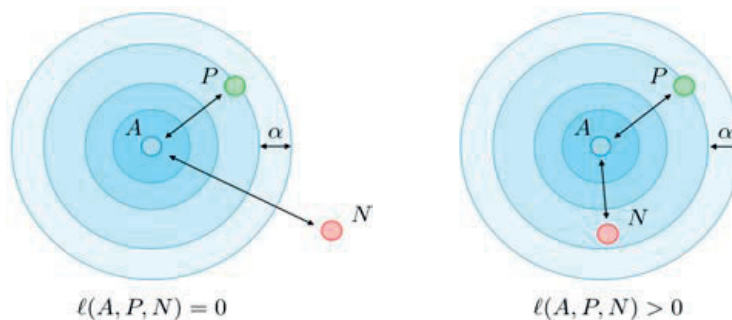
Face verification	Face recognition
- Is this the correct person? - One-to-one lookup	- Is this one of the $K$ persons in the database? - One-to-many lookup
	

□ **One Shot Learning** – One Shot Learning is a face verification algorithm that uses a limited training set to learn a similarity function that quantifies how different two given images are. The similarity function applied to two images is often noted  $d(\text{image 1}, \text{image 2})$ .

□ **Siamese Network** – Siamese Networks aim at learning how to encode images to then quantify how different two images are. For a given input image  $x^{(i)}$ , the encoded output is often noted as  $f(x^{(i)})$ .

□ **Triplet loss** – The triplet loss  $\ell$  is a loss function computed on the embedding representation of a triplet of images  $A$  (anchor),  $P$  (positive) and  $N$  (negative). The anchor and the positive example belong to a same class, while the negative example to another one. By calling  $\alpha \in \mathbb{R}^+$  the margin parameter, this loss is defined as follows:

$$\ell(A, P, N) = \max(d(A, P) - d(A, N) + \alpha, 0)$$



### 1.6.2 Neural style transfer

□ **Motivation** – The goal of neural style transfer is to generate an image  $G$  based on a given content  $C$  and a given style  $S$ .



□ **Activation** – In a given layer  $l$ , the activation is noted  $a^{[l]}$  and is of dimensions  $n_H \times n_w \times n_c$

□ **Content cost function** – The content cost function  $J_{\text{content}}(C, G)$  is used to determine how the generated image  $G$  differs from the original content image  $C$ . It is defined as follows:

$$J_{\text{content}}(C, G) = \frac{1}{2} \|a^{[l](C)} - a^{[l](G)}\|^2$$

□ **Style matrix** – The style matrix  $G^{[l]}$  of a given layer  $l$  is a Gram matrix where each of its elements  $G_{kk'}^{[l]}$  quantifies how correlated the channels  $k$  and  $k'$  are. It is defined with respect to activations  $a^{[l]}$  as follows:

$$G_{kk'}^{[l]} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_w^{[l]}} a_{ijk}^{[l]} a_{ijk'}^{[l]}$$

*Remark: the style matrix for the style image and the generated image are noted  $G^{[l](S)}$  and  $G^{[l](G)}$  respectively.*

□ **Style cost function** – The style cost function  $J_{\text{style}}(S, G)$  is used to determine how the generated image  $G$  differs from the style  $S$ . It is defined as follows:

$$J_{\text{style}}^{[l]}(S, G) = \frac{1}{(2n_H n_w n_c)^2} \|G^{[l](S)} - G^{[l](G)}\|_F^2 = \frac{1}{(2n_H n_w n_c)^2} \sum_{k, k'=1}^{n_c} \left( G_{kk'}^{[l](S)} - G_{kk'}^{[l](G)} \right)^2$$

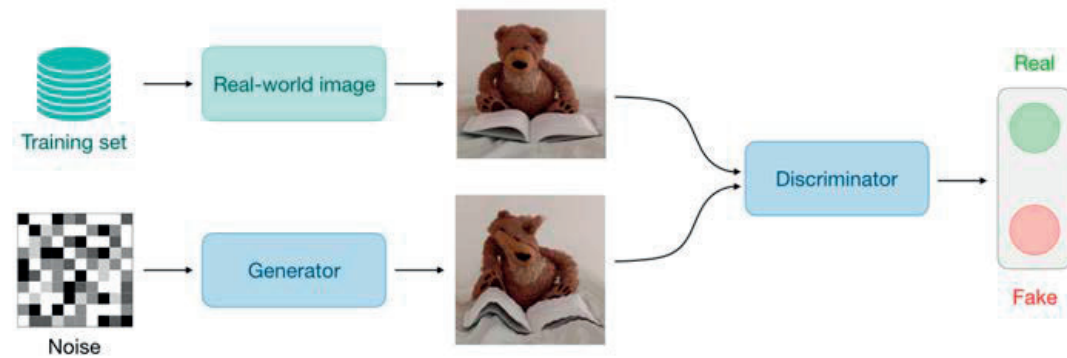
□ **Overall cost function** – The overall cost function is defined as being a combination of the content and style cost functions, weighted by parameters  $\alpha, \beta$ , as follows:

$$J(G) = \alpha J_{\text{content}}(C, G) + \beta J_{\text{style}}(S, G)$$

*Remark: a higher value of  $\alpha$  will make the model care more about the content while a higher value of  $\beta$  will make it care more about the style.*

### 1.6.3 Architectures using computational tricks

□ **Generative Adversarial Network** – Generative adversarial networks, also known as GANs, are composed of a generative and a discriminative model, where the generative model aims at generating the most truthful output that will be fed into the discriminative which aims at differentiating the generated and true image.



*Remark: use cases using variants of GANs include text to image, music generation and synthesis.*

□ **ResNet** – The Residual Network architecture (also called ResNet) uses residual blocks with a high number of layers meant to decrease the training error. The residual block has the following characterizing equation:

$$a^{[l+2]} = g(a^{[l]} + z^{[l+2]})$$

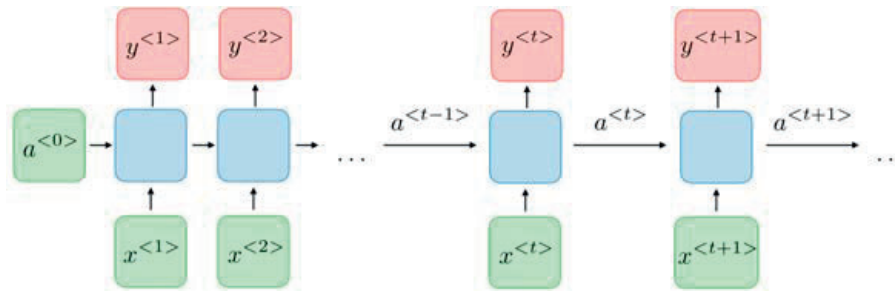
□ **Inception Network** – This architecture uses inception modules and aims at giving a try at different convolutions in order to increase its performance. In particular, it uses the  $1 \times 1$  convolution trick to lower the burden of computation.

\* \* \*

## 2 Recurrent Neural Networks

### 2.1 Overview

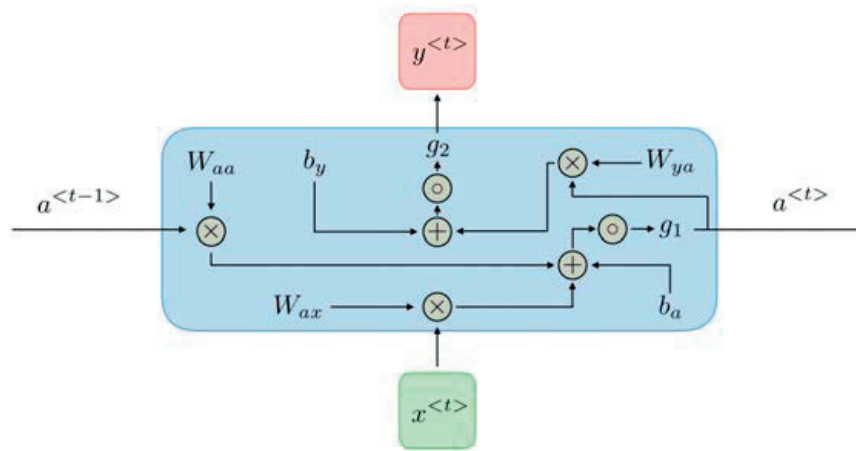
□ **Architecture of a traditional RNN** – Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. They are typically as follows:



For each timestep  $t$ , the activation  $a^{<t>}$  and the output  $y^{<t>}$  are expressed as follows:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad \text{and} \quad y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

where  $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$  are coefficients that are shared temporally and  $g_1, g_2$  activation functions



The pros and cons of a typical RNN architecture are summed up in the table below:

Advantages	Drawbacks
<ul style="list-style-type: none"> <li>- Possibility of processing input of any length</li> <li>- Model size not increasing with size of input</li> <li>- Computation takes into account historical information</li> <li>- Weights are shared across time</li> </ul>	<ul style="list-style-type: none"> <li>- Computation being slow</li> <li>- Difficulty of accessing information from a long time ago</li> <li>- Cannot consider any future input for the current state</li> </ul>

□ **Applications of RNNs** – RNN models are mostly used in the fields of natural language processing and speech recognition. The different applications are summed up in the table below:

Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification
Many-to-many $T_x = T_y$		Name entity recognition
Many-to-many $T_x \neq T_y$		Machine translation

□ **Loss function** – In the case of a recurrent neural network, the loss function  $\mathcal{L}$  of all time steps is defined based on the loss at every time step as follows:

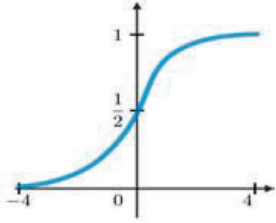
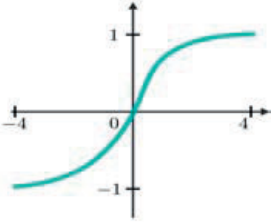
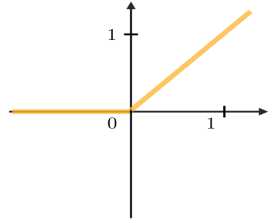
$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

□ **Backpropagation through time** – Backpropagation is done at each point in time. At timestep  $T$ , the derivative of the loss  $\mathcal{L}$  with respect to weight matrix  $W$  is expressed as follows:

$$\frac{\partial \mathcal{L}^{(T)}}{\partial W} = \sum_{t=1}^T \left. \frac{\partial \mathcal{L}^{(T)}}{\partial W} \right|_{(t)}$$

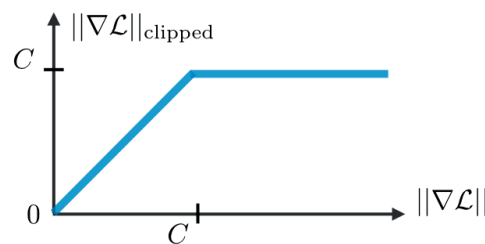
## 2.2 Handling long term dependencies

□ **Commonly used activation functions** – The most common activation functions used in RNN modules are described below:

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

□ **Vanishing/exploding gradient** – The vanishing and exploding gradient phenomena are often encountered in the context of RNNs. The reason why they happen is that it is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.

□ **Gradient clipping** – It is a technique used to cope with the exploding gradient problem sometimes encountered when performing backpropagation. By capping the maximum value for the gradient, this phenomenon is controlled in practice.



□ **Types of gates** – In order to remedy the vanishing gradient problem, specific gates are used in some types of RNNs and usually have a well-defined purpose. They are usually noted  $\Gamma$  and are equal to:

$$\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b)$$

where  $W, U, b$  are coefficients specific to the gate and  $\sigma$  is the sigmoid function. The main ones are summed up in the table below:

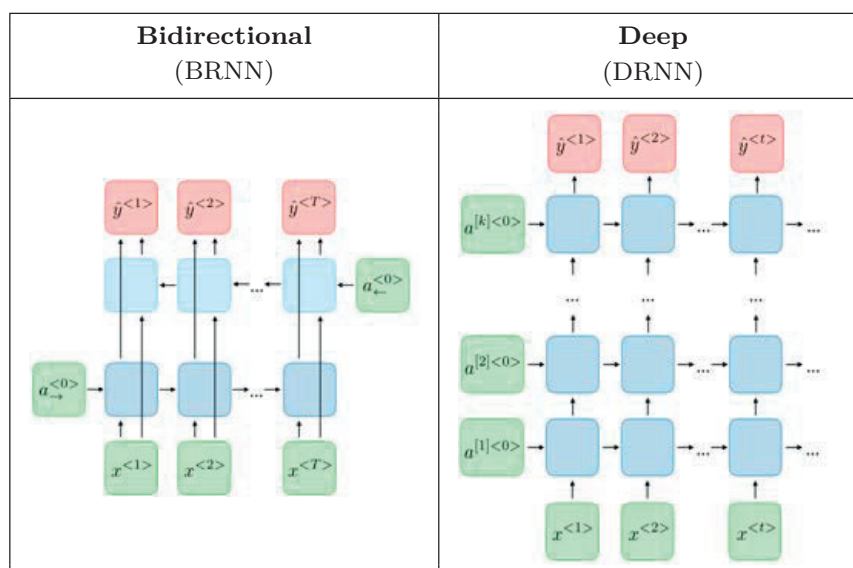
Type of gate	Role	Used in
Update gate $\Gamma_u$	How much past should matter now?	GRU, LSTM
Relevance gate $\Gamma_r$	Drop previous information?	GRU, LSTM
Forget gate $\Gamma_f$	Erase a cell or not?	LSTM
Output gate $\Gamma_o$	How much to reveal of a cell?	LSTM

□ **GRU/LSTM** – Gated Recurrent Unit (GRU) and Long Short-Term Memory units (LSTM) deal with the vanishing gradient problem encountered by traditional RNNs, with LSTM being a generalization of GRU. Below is a table summing up the characterizing equations of each architecture:

	Gated Recurrent Unit (GRU)	Long Short-Term Memory (LSTM)
$\tilde{c}^{<t>}$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$
$c^{<t>}$	$\Gamma_u \star \tilde{c}^{<t>} + (1 - \Gamma_u) \star c^{<t-1>}$	$\Gamma_u \star \tilde{c}^{<t>} + \Gamma_f \star c^{<t-1>}$
$a^{<t>}$	$c^{<t>}$	$\Gamma_o \star c^{<t>}$
Dependencies		

Remark: the sign  $\star$  denotes the element-wise multiplication between two vectors.

□ **Variants of RNNs** – The table below sums up the other commonly used RNN architectures:



### 2.3 Learning word representation

In this section, we note  $V$  the vocabulary and  $|V|$  its size.

#### 2.3.1 Motivation and notations

□ **Representation techniques** – The two main ways of representing words are summed up in the table below:

1-hot representation	Word embedding
<ul style="list-style-type: none"> <li>- Noted <math>o_w</math></li> <li>- Naive approach, no similarity information</li> </ul>	<ul style="list-style-type: none"> <li>- Noted <math>e_w</math></li> <li>- Takes into account words similarity</li> </ul>

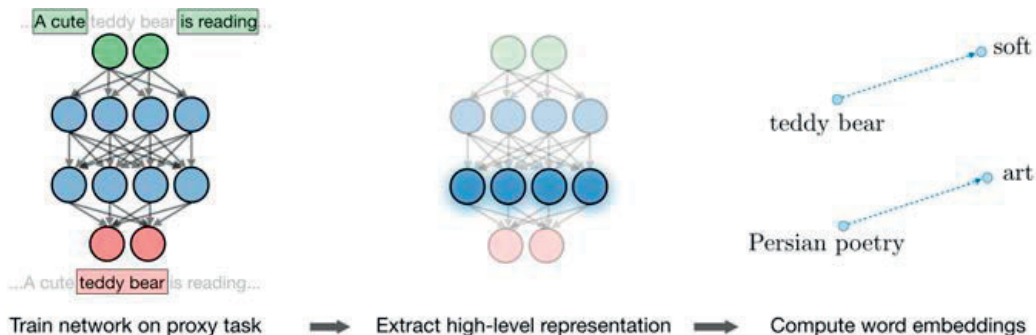
□ **Embedding matrix** – For a given word  $w$ , the embedding matrix  $E$  is a matrix that maps its 1-hot representation  $o_w$  to its embedding  $e_w$  as follows:

$$e_w = E o_w$$

*Remark: learning the embedding matrix can be done using target/context likelihood models.*

#### 2.3.2 Word embeddings

□ **Word2vec** – Word2vec is a framework aimed at learning word embeddings by estimating the likelihood that a given word is surrounded by other words. Popular models include skip-gram, negative sampling and CBOW.



□ **Skip-gram** – The skip-gram word2vec model is a supervised learning task that learns word embeddings by assessing the likelihood of any given target word  $t$  happening with a context word  $c$ . By noting  $\theta_t$  a parameter associated with  $t$ , the probability  $P(t|c)$  is given by:



$$P(t|c) = \frac{\exp(\theta_t^T e_c)}{\sum_{j=1}^{|V|} \exp(\theta_j^T e_c)}$$

*Remark: summing over the whole vocabulary in the denominator of the softmax part makes this model computationally expensive. CBOW is another word2vec model using the surrounding words to predict a given word.*

□ **Negative sampling** – It is a set of binary classifiers using logistic regressions that aim at assessing how a given context and a given target words are likely to appear simultaneously, with the models being trained on sets of  $k$  negative examples and 1 positive example. Given a context word  $c$  and a target word  $t$ , the prediction is expressed by:

$$P(y = 1|c,t) = \sigma(\theta_t^T e_c)$$

*Remark: this method is less computationally expensive than the skip-gram model.*

□ **GloVe** – The GloVe model, short for global vectors for word representation, is a word embedding technique that uses a co-occurrence matrix  $X$  where each  $X_{i,j}$  denotes the number of times that a target  $i$  occurred with a context  $j$ . Its cost function  $J$  is as follows:

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{|V|} f(X_{ij})(\theta_i^T e_j + b_i + b'_j - \log(X_{ij}))^2$$

here  $f$  is a weighting function such that  $X_{i,j} = 0 \implies f(X_{i,j}) = 0$ .

Given the symmetry that  $e$  and  $\theta$  play in this model, the final word embedding  $e_w^{(\text{final})}$  is given by:

$$e_w^{(\text{final})} = \frac{e_w + \theta_w}{2}$$

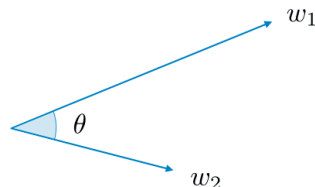
*Remark: the individual components of the learned word embeddings are not necessarily interpretable.*

## 2.4 Comparing words

□ **Cosine similarity** – The cosine similarity between words  $w_1$  and  $w_2$  is expressed as follows:

$$\text{similarity} = \frac{w_1 \cdot w_2}{\|w_1\| \|w_2\|} = \cos(\theta)$$

*Remark:  $\theta$  is the angle between words  $w_1$  and  $w_2$ .*



□  **$t$ -SNE** –  $t$ -SNE ( $t$ -distributed Stochastic Neighbor Embedding) is a technique aimed at reducing high-dimensional embeddings into a lower dimensional space. In practice, it is commonly used to visualize word vectors in the 2D space.



## 2.5 Language model

□ **Overview** – A language model aims at estimating the probability of a sentence  $P(y)$ .

□  **$n$ -gram model** – This model is a naive approach aiming at quantifying the probability that an expression appears in a corpus by counting its number of appearance in the training data.

□ **Perplexity** – Language models are commonly assessed using the perplexity metric, also known as PP, which can be interpreted as the inverse probability of the dataset normalized by the number of words  $T$ . The perplexity is such that the lower, the better and is defined as follows:

$$PP = \prod_{t=1}^T \left( \frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \cdot \hat{y}_j^{(t)}} \right)^{\frac{1}{T}}$$

*Remark: PP is commonly used in t-SNE.*

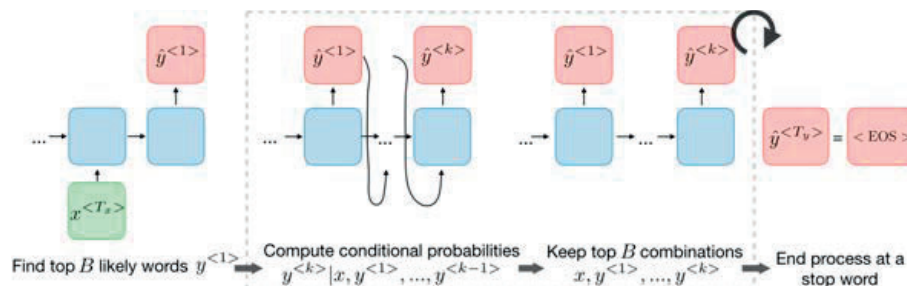
## 2.6 Machine translation

□ **Overview** – A machine translation model is similar to a language model except it has an encoder network placed before. For this reason, it is sometimes referred as a conditional language model. The goal is to find a sentence  $y$  such that:

$$y = \arg \max_{y^{<1>, \dots, y^{<T_y>}} P(y^{<1>, \dots, y^{<T_y>} | x)$$

□ **Beam search** – It is a heuristic search algorithm used in machine translation and speech recognition to find the likeliest sentence  $y$  given an input  $x$ .

- Step 1: Find top  $B$  likely words  $y^{<1>}$
- Step 2: Compute conditional probabilities  $y^{<k> | x, y^{<1>, \dots, y^{<k-1>}}$
- Step 3: Keep top  $B$  combinations  $x, y^{<1>, \dots, y^{<k>}}$



*Remark: if the beam width is set to 1, then this is equivalent to a naive greedy search.*

□ **Beam width** – The beam width  $B$  is a parameter for beam search. Large values of  $B$  yield to better result but with slower performance and increased memory. Small values of  $B$  lead to worse results but is less computationally intensive. A standard value for  $B$  is around 10.

□ **Length normalization** – In order to improve numerical stability, beam search is usually applied on the following normalized objective, often called the normalized log-likelihood objective, defined as:

$$\text{Objective} = \frac{1}{T_y^\alpha} \sum_{t=1}^{T_y} \log \left[ p(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>}) \right]$$

*Remark: the parameter  $\alpha$  can be seen as a softener, and its value is usually between 0.5 and 1.*

□ **Error analysis** – When obtaining a predicted translation  $\hat{y}$  that is bad, one can wonder why we did not get a good translation  $y^*$  by performing the following error analysis:

Case	$P(y^*   x) > P(\hat{y}   x)$	$P(y^*   x) \leq P(\hat{y}   x)$
<b>Root cause</b>	Beam search faulty	RNN faulty
<b>Remedies</b>	Increase beam width	<ul style="list-style-type: none"> <li>- Try different architecture</li> <li>- Regularize</li> <li>- Get more data</li> </ul>

□ **Bleu score** – The bilingual evaluation understudy (bleu) score quantifies how good a machine translation is by computing a similarity score based on  $n$ -gram precision. It is defined as follows:

$$\text{bleu score} = \exp \left( \frac{1}{n} \sum_{k=1}^n p_k \right)$$

where  $p_n$  is the bleu score on  $n$ -gram only defined as follows:

$$p_n = \frac{\sum_{\text{n-gram} \in \hat{y}} \text{count}_{\text{clip}}(\text{n-gram})}{\sum_{\text{n-gram} \in \hat{y}} \text{count}(\text{n-gram})}$$

*Remark: a brevity penalty may be applied to short predicted translations to prevent an artificially inflated bleu score.*

## 2.7 Attention

□ **Attention model** – This model allows an RNN to pay attention to specific parts of the input that is considered as being important, which improves the performance of the resulting model in practice. By noting  $\alpha^{<t,t'>}$  the amount of attention that the output  $y^{<t>}$  should pay to the activation  $a^{<t'>}$  and  $c^{<t>}$  the context at time  $t$ , we have:

$$c^{<t>} = \sum_{t'} \alpha^{<t,t'>} a^{<t'>} \quad \text{with} \quad \sum_{t'} \alpha^{<t,t'>} = 1$$

*Remark: the attention scores are commonly used in image captioning and machine translation.*



A cute teddy bear is reading Persian literature



A cute teddy bear is reading Persian literature

□ **Attention weight** – The amount of attention that the output  $y^{<t>}$  should pay to the activation  $a^{<t'>}$  is given by  $\alpha^{<t,t'>}$  computed as follows:

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t''=1}^{T_x} \exp(e^{<t,t''>})}$$

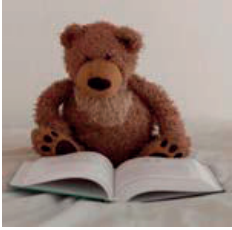



*Remark: computation complexity is quadratic with respect to  $T_x$ .*





\* \* \*

### 3 Deep Learning Tips and Tricks

#### 3.1 Data processing

□ **Data augmentation** – Deep learning models usually need a lot of data to be properly trained. It is often useful to get more data from the existing ones using data augmentation techniques. The main ones are summed up in the table below. More precisely, given the following input image, here are the techniques that we can apply:

Original	Flip	Rotation	Random crop
			
- Image without any modification	- Flipped with respect to an axis for which the meaning of the image is preserved	- Rotation with a slight angle - Simulates incorrect horizon calibration	- Random focus on one part of the image - Several random crops can be done in a row

Color shift	Noise addition	Information loss	Contrast change
			
- Nuances of RGB is slightly changed - Captures noise that can occur with light exposure	- Addition of noise - More tolerance to quality variation of inputs	- Parts of image ignored - Mimics potential loss of parts of image	- Luminosity changes - Controls difference in exposition due to time of day

□ **Batch normalization** – It is a step of hyperparameter  $\gamma, \beta$  that normalizes the batch  $\{x_i\}$ . By noting  $\mu_B, \sigma_B^2$  the mean and variance of that we want to correct to the batch, it is done as follows:

$$x_i \leftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

It is usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

### 3.2 Training a neural network

#### 3.2.1 Definitions

□ **Epoch** – In the context of training a model, epoch is a term used to refer to one iteration where the model sees the whole training set to update its weights.

□ **Mini-batch gradient descent** – During the training phase, updating weights is usually not based on the whole training set at once due to computation complexities or one data point due to noise issues. Instead, the update step is done on mini-batches, where the number of data points in a batch is a hyperparameter that we can tune.

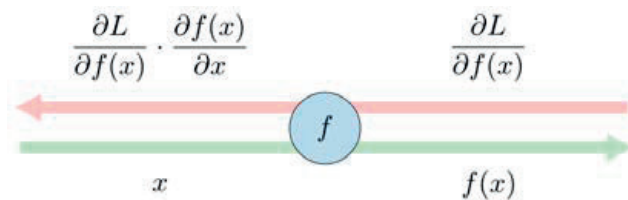
□ **Loss function** – In order to quantify how a given model performs, the loss function  $L$  is usually used to evaluate to what extent the actual outputs  $y$  are correctly predicted by the model outputs  $z$ .

□ **Cross-entropy loss** – In the context of binary classification in neural networks, the cross-entropy loss  $L(z,y)$  is commonly used and is defined as follows:

$$L(z,y) = - \left[ y \log(z) + (1 - y) \log(1 - z) \right]$$

#### 3.2.2 Finding optimal weights

□ **Backpropagation** – Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to each weight  $w$  is computed using the chain rule.

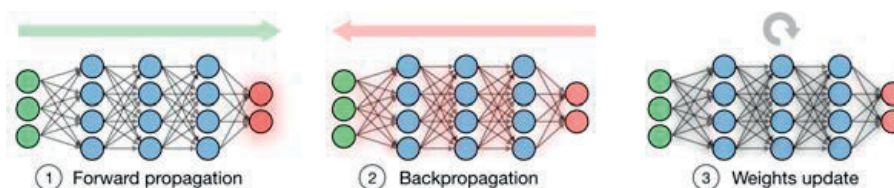


Using this method, each weight is updated with the rule:

$$w \leftarrow w - \alpha \frac{\partial L(z,y)}{\partial w}$$

□ **Updating weights** – In a neural network, weights are updated as follows:

- Step 1: Take a batch of training data and perform forward propagation to compute the loss.
- Step 2: Backpropagate the loss to get the gradient of the loss with respect to each weight.
- Step 3: Use the gradients to update the weights of the network.

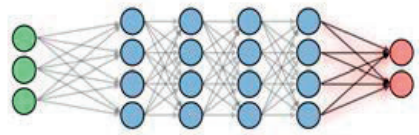
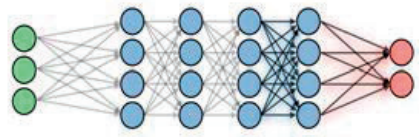
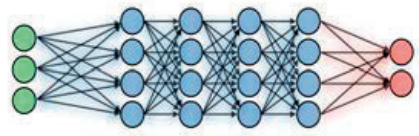


### 3.3 Parameter tuning

#### 3.3.1 Weights initialization

□ **Xavier initialization** – Instead of initializing the weights in a purely random manner, Xavier initialization enables to have initial weights that take into account characteristics that are unique to the architecture.

□ **Transfer learning** – Training a deep learning model requires a lot of data and more importantly a lot of time. It is often useful to take advantage of pre-trained weights on huge datasets that took days/weeks to train, and leverage it towards our use case. Depending on how much data we have at hand, here are the different ways to leverage this:

Training size	Illustration	Explanation
Small		Freezes all layers, trains weights on softmax
Medium		Freezes most layers, trains weights on last layers and softmax
Large		Trains weights on layers and softmax by initializing weights on pre-trained ones

#### 3.3.2 Optimizing convergence

□ **Learning rate** – The learning rate, often noted  $\alpha$  or sometimes  $\eta$ , indicates at which pace the weights get updated. It can be fixed or adaptively changed. The current most popular method is called Adam, which is a method that adapts the learning rate.

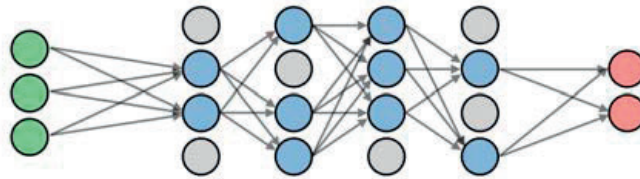
□ **Adaptive learning rates** – Letting the learning rate vary when training a model can reduce the training time and improve the numerical optimal solution. While Adam optimizer is the most commonly used technique, others can also be useful. They are summed up in the table below:

Method	Explanation	Update of $w$	Update of $b$
Momentum	<ul style="list-style-type: none"> <li>- Dampens oscillations</li> <li>- Improvement to SGD</li> <li>- 2 parameters to tune</li> </ul>	$w - \alpha v_{dw}$	$b - \alpha v_{db}$
RMSprop	<ul style="list-style-type: none"> <li>- Root Mean Square propagation</li> <li>- Speeds up learning algorithm by controlling oscillations</li> </ul>	$w - \alpha \frac{dw}{\sqrt{s_{dw}}}$	$b \leftarrow b - \alpha \frac{db}{\sqrt{s_{db}}}$
Adam	<ul style="list-style-type: none"> <li>- Adaptive Moment estimation</li> <li>- Most popular method</li> <li>- 4 parameters to tune</li> </ul>	$w - \alpha \frac{v_{dw}}{\sqrt{s_{dw}} + \epsilon}$	$b \leftarrow b - \alpha \frac{v_{db}}{\sqrt{s_{db}} + \epsilon}$

Remark: other methods include Adadelta, Adagrad and SGD.

### 3.4 Regularization

**Dropout** – Dropout is a technique used in neural networks to prevent overfitting the training data by dropping out neurons with probability  $p > 0$ . It forces the model to avoid relying too much on particular sets of features.



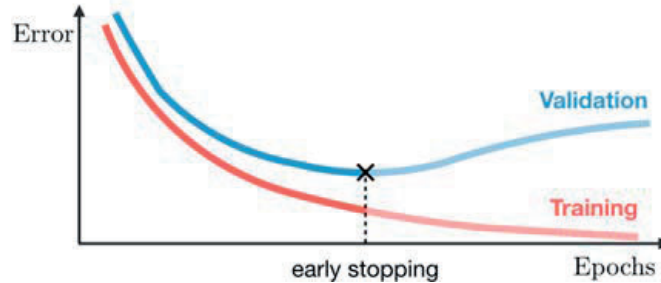
Remark: most deep learning frameworks parametrize dropout through the 'keep' parameter  $1 - p$ .

**Weight regularization** – In order to make sure that the weights are not too large and that the model is not overfitting the training set, regularization techniques are usually performed on the model weights. The main ones are summed up in the table below:

LASSO	Ridge	Elastic Net
<ul style="list-style-type: none"> <li>- Shrinks coefficients to 0</li> <li>- Good for variable selection</li> </ul>	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
$\dots + \lambda \ \theta\ _1$ $\lambda \in \mathbb{R}$	$\dots + \lambda \ \theta\ _2^2$ $\lambda \in \mathbb{R}$	$\dots + \lambda \left[ (1 - \alpha) \ \theta\ _1 + \alpha \ \theta\ _2^2 \right]$ $\lambda \in \mathbb{R}, \alpha \in [0, 1]$



□ **Early stopping** – This regularization technique stops the training process as soon as the validation loss reaches a plateau or starts to increase.



### 3.5 Good practices

□ **Overfitting small batch** – When debugging a model, it is often useful to make quick tests to see if there is any major issue with the architecture of the model itself. In particular, in order to make sure that the model can be properly trained, a mini-batch is passed inside the network to see if it can overfit on it. If it cannot, it means that the model is either too complex or not complex enough to even overfit on a small batch, let alone a normal-sized training set.

□ **Gradient checking** – Gradient checking is a method used during the implementation of the backward pass of a neural network. It compares the value of the analytical gradient to the numerical gradient at given points and plays the role of a sanity-check for correctness.

	Numerical gradient	Analytical gradient
<b>Formula</b>	$\frac{df}{dx}(x) \approx \frac{f(x+h) - f(x-h)}{2h}$	$\frac{df}{dx}(x) = f'(x)$
<b>Comments</b>	<ul style="list-style-type: none"> <li>- Expensive; loss has to be computed two times per dimension</li> <li>- Used to verify correctness of analytical implementation</li> <li>- Trade-off in choosing <math>h</math> not too small (numerical instability) nor too large (poor gradient approx.)</li> </ul>	<ul style="list-style-type: none"> <li>- 'Exact' result</li> <li>- Direct computation</li> <li>- Used in the final implementation</li> </ul>

\* \* \*