UFM UNIVERSIDAD FRANCISCO MARROQUÍN
VERITAS • LIBERTAS • IVSTITIA

FACULTAD de CIENCIAS ECONÓMICAS

Redes Neuronales

Alan Reyes-Figueroa

Elements of Machine Learning

(Aula 21) 25.abril.2023

Table of Contents

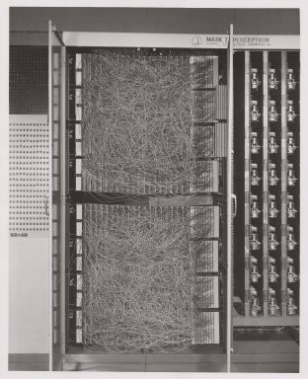
Introduction to Neural Networks

Crash course on Neural Networks

Neural networks are mathematical / computational models inspired in how the neurons connect each other in the brain.

The beginnings: (1940's-1960's)

- (1943). McCulloch-Pitts neuron model, proposed by Warren S. McCulloch, a neuroscientist, and Walter Pitts, a logician.
- (Late 40's). Other precursors of neural networks: "Threshold Logic" – converting continuous input to discrete output; and "Hebbian Learning" – a model of learning based on neural plasticity, proposed by Donald Hebb. First Hebbian networks implemented at MIT in 1954.
- (1958). Frank Rosenblatt, a psychologist at Cornell, proposed the Perceptron, modelen on the McCulloch-Pitts neuron.

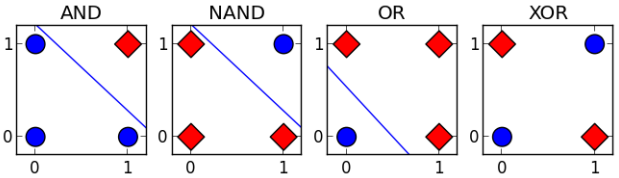


The Mark I Perceptron.

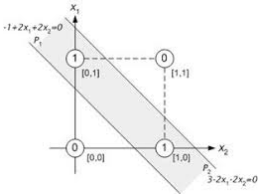
Crash course on neural networks

The beginnings: (1940's-1960's)

- (1959). Bernard Widrow and Marcian Hoff, atf Stanford, developed models ADALINE and MADALINE. (Multiple) ADAptive LINear Elements. MADALINE was the first neural network to be applied to a real-world problem. It is an adaptive filter which eliminates echoes on phone lines. This neural network is still in commercial use.
- (Late 59's). Marvin Minsky and Seymour Papert published the book *Perceptrons*. They proved the perceptron model to be limited. Mayor drawbacks and hiatus.



The XOR problem.



UFM UNIVERSIDAD FRANCISCO MARROQUÍN

Crash course on neural networks

Resurgence: (1980's-1990's)

- (1982). John Hopfield presented a paper to the national Academy of Sciences. His approach to create useful devices.
- (1982). US-Japan Joint Conference on Cooperative/ Competitive Neural Networks. Funding was flowing once again.
- (1985). IEEE first International Conference on Neural Networks.
- (1986). Rumelhart, Hinton and Williams rediscover the *backpropagation method*.
- (1992). Belief neural networks and statistical graphical models.
- (1997). Long Short-Term Memory (LSTM) was proposed by Schmidhuber Hochreiter.
- (1998). Yann LeCun published Gradient-Based Learning Applied to Document Recognition. Start of the field applied to vision.
- (1990's). Limitations of backpropagation method. Exploding or vanishing gradient problem.
- (2000's). Neural networks surpassed by other popular methods: SVMs and Random Forests. Second hiatus.

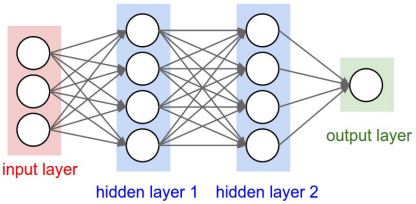
Crash course on neural networks

Decade of Deep Learning: (2010's–)

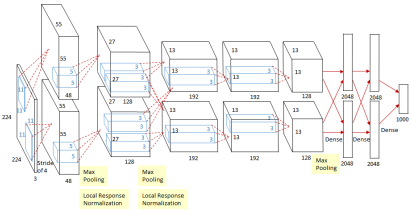
- (2010–). Development of modern computation technologies: More powerful PC's, development of GPU's.
- (2012). ImageNet competition. First massive neural architectures, *e.g.* AlexNet.
- (2015–). Deep learning tsunami.

More information at

- `https://www.skynettoday.com/overviews/neural-net-history`
- `https://cs231n.github.io/neural-networks-1/`



A multilayer perceptron.



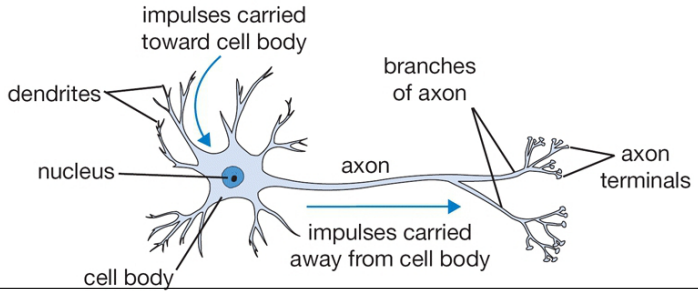
Alexnet, 2017

UFM UNIVERSIDAD FRANCISCO MARROQUÍN

Neural Networks

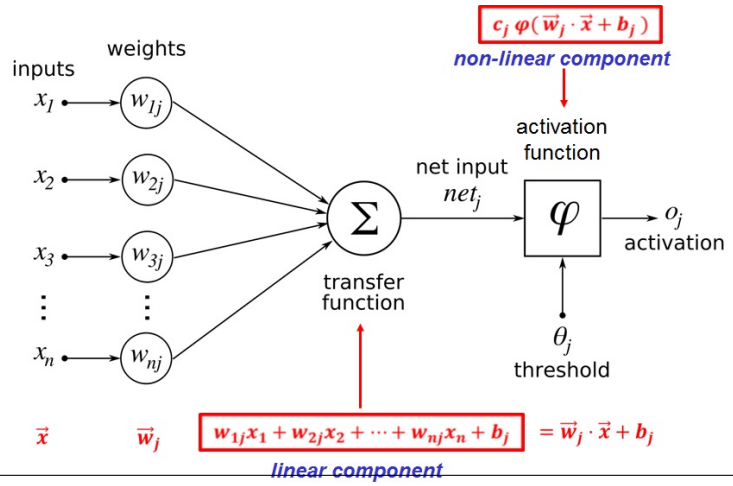
The area of Neural Networks has originally been primarily inspired by the goal of modeling biological neural systems, but has since diverged and become a matter of engineering and achieving good results in Machine Learning tasks.

Biological motivation and connections: The basic computational unit of the brain is a neuron. Approximately 86 billion neurons can be found in the human nervous system and they are connected with approximately $10^{14} - 10^{15}$ synapses.



Neural Networks

We can model a biological neuron in the following way

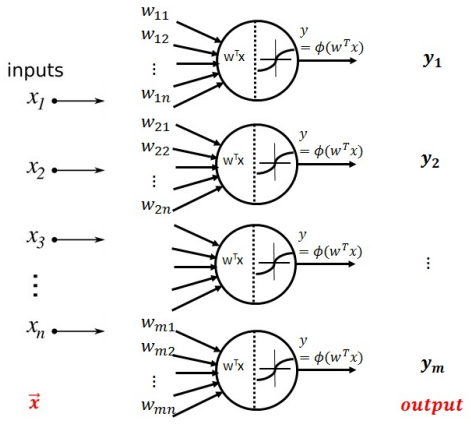


Neural Networks

Each neuron receives input signals from its dendrites and produces output signals along its (single) axon. The axon eventually branches out and connects via synapses to dendrites of other neurons. In the computational model of a neuron, the signals that travel along the axons (e.g. $\mathbf{x}$) interact multiplicatively (e.g. $\mathbf{w}_j^T\mathbf{x}$) with the dendrites of the other neuron based on the synaptic strength at that synapse (e.g. $\mathbf{w}_j$). The idea is that the synaptic strengths (the weights w_{ij}) are learnable and control the strength of influence (and its direction: excitory (positive weight) or inhibitory (negative weight)) of one neuron on another. In the basic model, the dendrites carry the signal to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can fire, sending a spike along its axon. In the computational model, we assume that the precise timings of the spikes do not matter, and that only the frequency of the firing communicates information. Based on this rate code interpretation, we model the firing rate of the neuron with an activation function φ, which represents the frequency of the spikes along the axon. Historically, a common choice of activation function is the sigmoid function σ, since it takes a real-valued input (the signal strength after the sum) and squashes it to range between 0 and 1.

Neural Networks

We usually combine several neurons in a layer



$$W = \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & & \vdots \\ w_{m1} & \cdots & w_{mn} \end{pmatrix} \in \mathbb{R}^{m \times n}$$

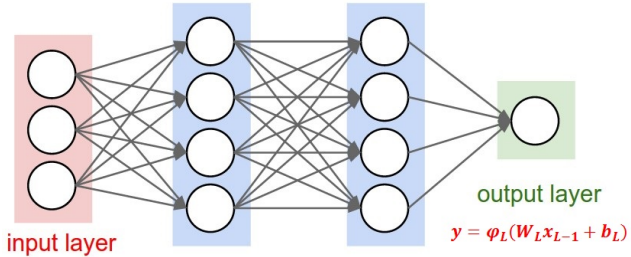
$$b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \in \mathbb{R}^m$$

$$\varphi: \mathbb{R}^m \to \mathbb{R}^m$$

$$y = \varphi(Wx + b)$$

Neural Networks

Typical fully-connected artificial neural networks consists of a series of layers.



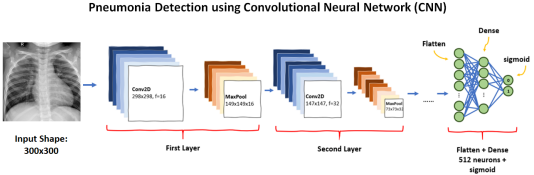
Finally, you have a model of the form

$$\widehat{\mathbf{y}} = \varphi_L\left(\mathbf{W}_L \ldots \varphi_3\Big(\mathbf{W}_3\ \varphi_2\big(\mathbf{W}_2\ \varphi_1(\mathbf{W}_1\,\mathbf{x}_0 + \mathbf{b}_1) + \mathbf{b}_2\big) + \mathbf{b}_3\Big) \ldots + \mathbf{b}_L\right).$$

Deep Neural Networks

In the era of deep learning, neural networks are specialized and specifically designed:

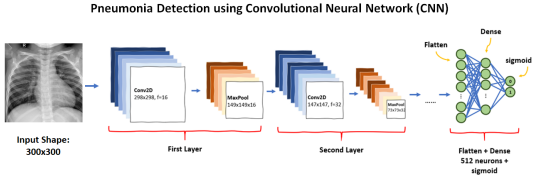
- Convolutional neural networks: Are useful for image processing and computer vision tasks. They use a specialized units called Convolutional Layers.



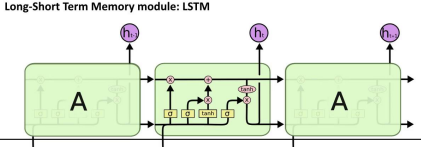
Deep Neural Networks

In the era of deep learning, neural networks are specialized and specifically designed:

- Convolutional neural networks: Are useful for image processing and computer vision tasks. They use a specialized units called Convolutional Layers.



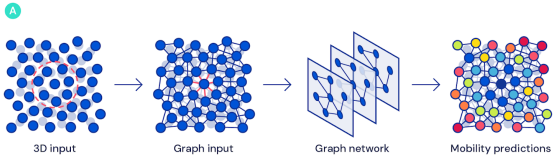
- Recurrent neural networks: Are useful for image treating sequencial data (e.g. text, time series, speech). They use a specialized repetitive units as LSTM or GRU.



UFM UNIVERSIDAD FRANCISCO MARROQUÍN

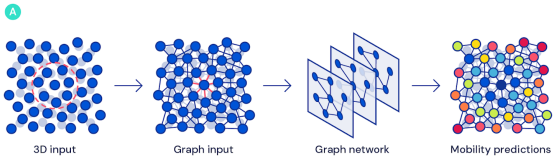
Deep Neural Networks

- Graph neural networks: Are useful for processing connected structures as graphs or networks.

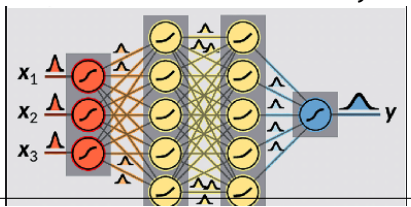


Deep Neural Networks

- Graph neural networks: Are useful for processing connected structures as graphs or networks.

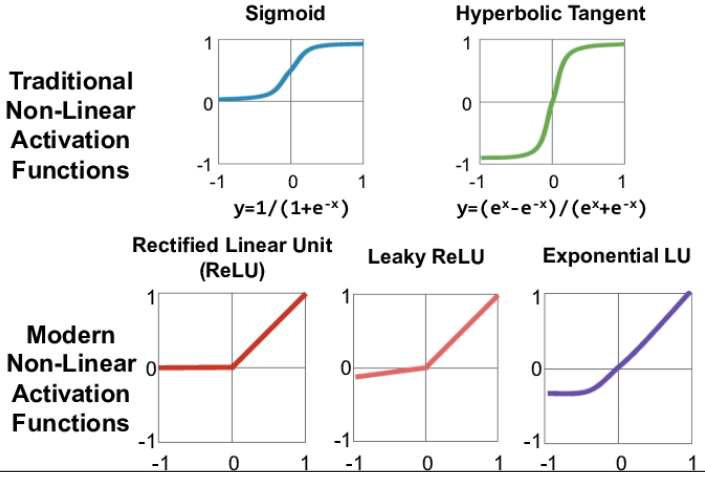


- Bayesian neural networks: They try to produce not single weights, but probability distributions of weights for each neuron. Useful for bayesian analysis.



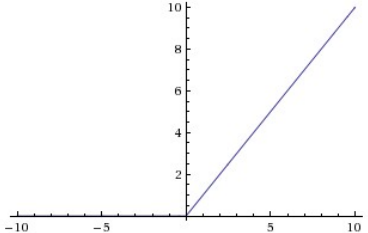
Activation Functions

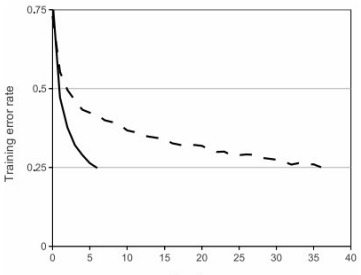
Some of the most common activation functions are



Activation Functions

Today, the most used activation function is the so-called *ReLU* (Rectified Linear Unit). It has useful convergence properties.

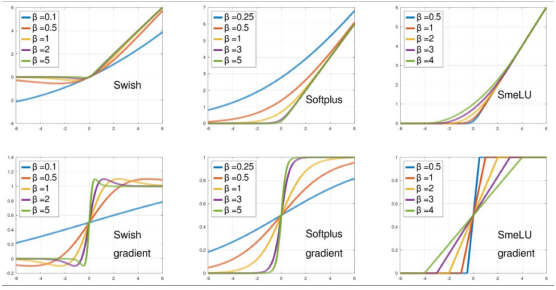




Obs: The activation function of the output layer must be set depending on the problem objective, or the purpose of the neural network. For example, in a classification problem, common activation functions are the sigmoid (binary classification) or the soft-max (multi-classification). In a regression problem, usually one defines the linear

Activation Functions

SmeLU (Smooth ReLU). Released in October 2020.



$$y_{\text{SmeLU}} = \begin{cases} 0; & x \leq -\beta \\ \frac{(x+\beta)^2}{4\beta}; & |x| \leq \beta \\ x; & x \geq \beta. \end{cases}$$

Parameters

The weights $\mathbf{W}_L = (w_{ij})_\ell$, and the biases $\mathbf{b}_\ell$, $\ell = 1, \ldots, L$ are the learnable parameters of the model. This means that we will develop an automated way to compute the weights to "learn" an specific task. This automated mechanism come in the way of an optimization algorithm.

Neural networks can bee seen in two different forms:

- As classifiers. Suppose we want to solve a classification problem (we have data, each corresponding to a one of k different labels. I this case, usually one has k neurons in the last layer (output layer) of the network. Each of this neurons, encodes the probability of the data being in each class

$$y_i = \text{output at neuron i} = \mathbb{P}(\mathbf{x} \text{ is in the class i}).$$

- As function approximators. Suppose we want to solve a regression of fitting problem (we want to approximate a function $\mathbf{y} = f(\mathbf{x})$). I this case, usually one has d neurons in the last layer, where $\mathbf{y} \in \mathbb{R}^d$. Each of this neurons, encodes the approximation of the data $\mathbf{y}$ at the $i-$th dimensional component

$$y_i = \text{output at neuron i, and } \widehat{\mathbf{y}} = (y_1, \ldots, y_d) \text{ is the approximation of } \mathbf{y}$$

Learn by examples

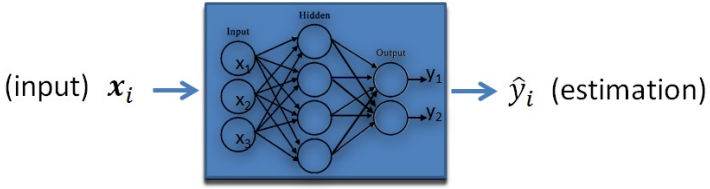
Neural networks are supervised machine learning models. This means they learn by examples. That means the following.

Suppose we have a sample of data $\left\{(\mathbf{x}_i), y_i)\right\}_{i=1}^{n}$. Here the $\mathbf{x}_i \in \mathbb{R}^p$ are the input data, while the $y_i \in \mathbb{R}^d$ are the "desired" data (the *ground-truth*). For each $i = 1, 2, \ldots, n$, the neural network will produce an approximation (this is what we want) of each y_i. We will denote $\widehat{y}_i$ the output of the neural network when we input $\mathbf{x}_i$, respectively.

Learn by examples

Neural networks are supervised machine learning models. This means they learn by examples. That means the following.

Suppose we have a sample of data $\{(\mathbf{x}_i), y_i)\}_{i=1}^{n}$. Here the $\mathbf{x}_i \in \mathbb{R}^p$ are the input data, while the $y_i \in \mathbb{R}^d$ are the "desired" data (the *ground-truth*). For each $i = 1, 2, \ldots, n$, the neural network will produce an approximation (this is what we want) of each y_i. We will denote $\widehat{y}_i$ the output of the neural network when we input $\mathbf{x}_i$, respectively.



Learn by examples

Neural networks are supervised machine learning models. This means they learn by examples. That means the following.

Suppose we have a sample of data $\{(\mathbf{x}_i), y_i)\}_{i=1}^{n}$. Here the $\mathbf{x}_i \in \mathbb{R}^p$ are the input data, while the $y_i \in \mathbb{R}^d$ are the "desired" data (the *ground-truth*). For each $i = 1, 2, \ldots, n$, the neural network will produce an approximation (this is what we want) of each y_i. We will denote $\widehat{y}_i$ the output of the neural network when we input $\mathbf{x}_i$, respectively.

Learn by examples

Neural networks are supervised machine learning models. This means they learn by examples. That means the following.

Suppose we have a sample of data $\{(\mathbf{x}_i), y_i)\}_{i=1}^{n}$. Here the $\mathbf{x}_i \in \mathbb{R}^p$ are the input data, while the $y_i \in \mathbb{R}^d$ are the "desired" data (the *ground-truth*). For each $i = 1, 2, \ldots, n$, the neural network will produce an approximation (this is what we want) of each y_i. We will denote $\widehat{y}_i$ the output of the neural network when we input $\mathbf{x}_i$, respectively.

Thus, a neural network can be seen as an approximation $\widehat{f} : \mathbf{x}_i \to \widehat{y}_i$ of the function $f : \mathbf{x}_i \to y_i$.

Loss function

The *loss function* is the objective function that describes the difference between the approximations $\widehat{y}_i$ and the ground-truths y_i, of the given sample. It will measure the error of the discrepancy between each pair $(\widehat{y}_i, y_i)$.

- Classification: In a classification problem, one common way to measure the difference between probability values is the *binary-crossentropy*

$$\mathcal{L}(\mathbf{x}_i, y_i) = -\sum_{i=1}^{n} y_i \log(\widehat{y}_i) + (1 - y_i) \log(1 - \widehat{y}_i).$$

(if we have only two labels) or the *categorical-crossentropy* for a multi-label classification

$$\mathcal{L}(\mathbf{x}_i, y_i) = -\sum_{i=1}^{n} \sum_{j=1}^{labels} y_{ij} \log(\widehat{y}_{ij}).$$

Loss function

- Regression: In a regression problem, common ways to measure the difference between the estimation $\widehat{y}_i$ and the desired ground-truth y_i are the MSE

$$\mathcal{L}(\mathbf{x}_i, y_i) = MSE = \frac{1}{n}\sum_{i=1}^{n} ||y_i - \widehat{y}_i||_2^2,$$

  and MAE errors

$$\mathcal{L}(\mathbf{x}_i, y_i) = MAE = \frac{1}{n}\sum_{i=1}^{n} ||y_i - \widehat{y}_i||_1.$$

Sometimes, the loss function also incorporates some regularization terms, depending on the problem and user preferences. For example

$$\mathcal{L}(\mathbf{x}_i, y_i) = \frac{1}{n}\sum_{i=1}^{n} ||y_i - \widehat{y}_i||_1 + \lambda_1 \sum_{i,j} ||w_{ij}||_1 + \lambda_2 ||\nabla_{\mathbf{w}} \widehat{y}_i||_2^2.$$

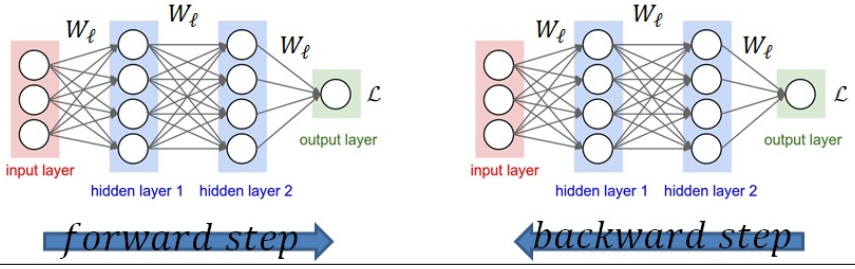
Backpropagation

Backpropagation is the mechanism that is used to compute the parameters of the neural network (weights $w_{ij\ell}$ and biases $b_{i\ell}$) from the loss function.

From a mathematical point of view, this is simply to compute the derivatives

$$\nabla_{w_{ij\ell}} \mathcal{L}(\mathbf{x}, y) \qquad \text{and} \qquad \nabla_{b_{j\ell}} \mathcal{L}(\mathbf{x}, y),$$

for each parameter $w_{ij\ell}$ and biases $b_{i\ell}$.



Backpropagation

- In the forward step, given the actual weights $w_{ij\ell}$ in the network, each input $\mathbf{x}_i$ is passed trough the network (in fact all $\mathbf{x}_i$ are passed at the same time), and we compute te estimations y_i. Then, the los function $\mathcal{L}$ is computed.

- In the backward step, we compute the derivatives $\nabla_{w_{ij\ell}}\mathcal{L}$ and $\nabla_{b_{j\ell}}\mathcal{L}$. Then the weights $w_{ij\ell}$ and biases $b_{i\ell}$ are recalculated.

- Both steps are repeated until we reach convergence, or some other stop criterion holds.

The re-calculation process is done by using the *gradient descent* optimization algorithm,

$$\begin{aligned} w_{ij\ell}^{(k+1)} &= w_{ij\ell}^{(k)} - \alpha\nabla_{w_{ij\ell}^{(k)}}\mathcal{L}(\mathbf{x}, y), \\ b_{i\ell}^{(k+1)} &= b_{i\ell}^{(k)} - \alpha\nabla_{b_{i\ell}^{(k)}}\mathcal{L}(\mathbf{x}, y), \end{aligned}$$

Here, $\alpha > 0$ is the step size of *learning rate*. In practice, today we use any of the stochastic variants of gradient descent.

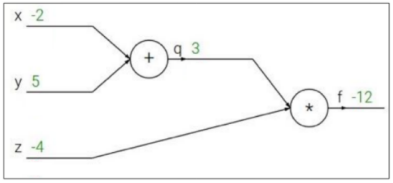
Backpropagation Example

$$f(x, y, z) = (x + y)z$$

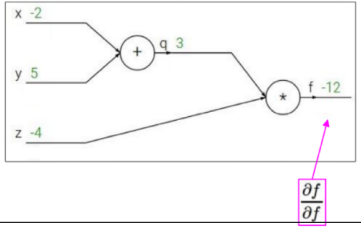
e.g. x = -2, y = 5, z = -4

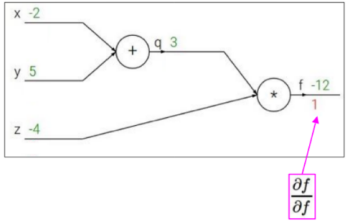
$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

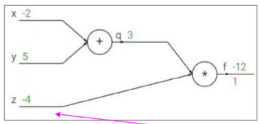


Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

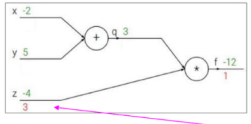




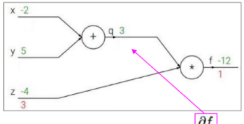
Backpropagation Example



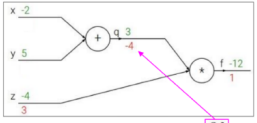
$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$

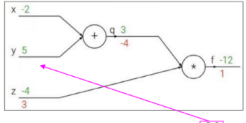


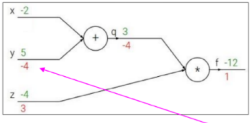
$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

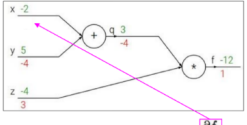


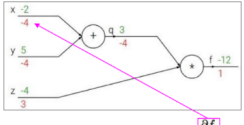
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$











$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Chain rule:

UFM UNIVERSIDAD FRANCISCO MARROQUÍN

Stochastic gradient descent

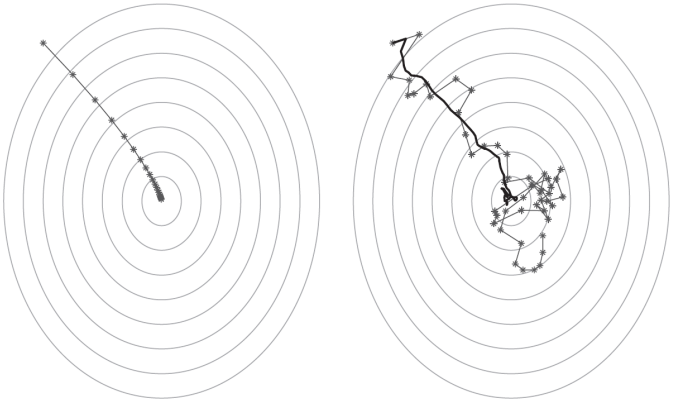
- Classic gradient descent: It computes the loss function $\mathcal{L}$ from all samples $x_1, \ldots, x_n$, and then actualize the weights.

- Stochastic gradient descent: At each iteration, it randomly chooses one sample x_i (with equal probability and no replacement), it computes the loss $\mathcal{L}$ using only the sample x_i, and then actualize the weights. This is repeated until all the samples were chosen. After all samples are used, we complete one *epoch*.

- Mini-batch gradient descent: We split the sample in b subsets or batches $B_1, \ldots, B_b$. At each iteration, it randomly chooses one batch B_j (with equal probability and no replacement), it computes the loss $\mathcal{L}$ using only the samples in batch B_j, and then actualize the weights. This is repeated until all batches are chosen. After all samples are used, we complete one *epoch*.

There are a lot of variants and improvements of stochastic gradient descent. See

Optimization process



Differences between gradient descent and stochastic gradient descent: (a) gradient descent. (b) Stochastic gradient descent.

In summary

Parameters: (learnable by gradient descent)

- weights $w_{ij\ell}$ and biases $b_{i\ell}$, for $\ell = 1, \ldots, L$.

Hyper-parameters: (user-defined, non-learnable by gradient descent)

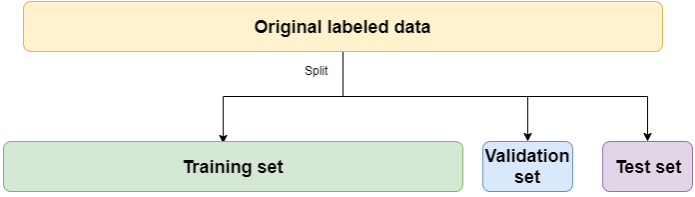
- Number of layers, and the type of each layer.
- Size of each layer (number of neurons in each layer).
- Activation function of each layer.
- Connections between the layers.

- Loss function (and other metrics).
- Optimization algorithms (GD, SGD, Nesterov, Adam, Adagrad, Adamax, RMSProp, ...), and parameters of those algorithms.
- learning rate or step-size α.
- Size ob the batch.
- Number of epochs.

Train / Validation / Test

To evaluate the performance of our neural network model, and evaluate the training process, we usually split our data in two (or three) subsets: training data, validation data, test data.

- Training data: used for the training process, this is the sample to compute loss function and weights.
- Validation data: used also in the training process, but just to evaluate the loss with new data. They are not used to compute the weights.
- Test data: used at the end for the final evaluation of the performance. Typically used to report performance in papers.



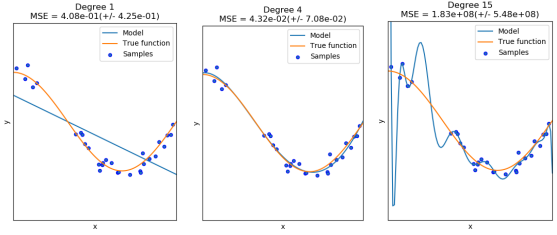
Evaluation

There is no a recipe for the size or percentage of each subset. Common values are:
Training (80%), Validation (10%), Test (10%),
Training (60%), Validation (20%), Test (20%),
Training (50%), Validation (40%), Test (10%),
Depends on how much data you have, how much data you can sacrifice for testing, ...

Evaluation

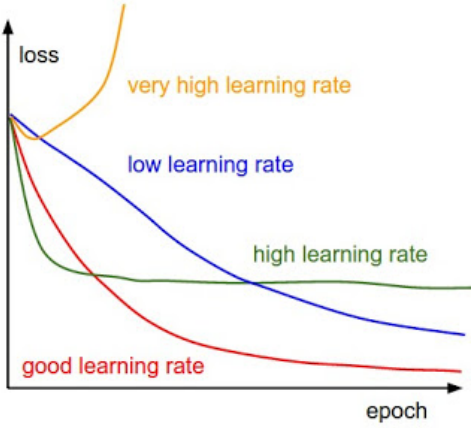
There is no a recipe for the size or percentage of each subset. Common values are:
Training (80%), Validation (10%), Test (10%),
Training (60%), Validation (20%), Test (20%),
Training (50%), Validation (40%), Test (10%),
Depends on how much data you have, how much data you can sacrifice for testing, ...

We want to evaluate our model to see if it generalizes well new unseen samples (bias-variance trading).



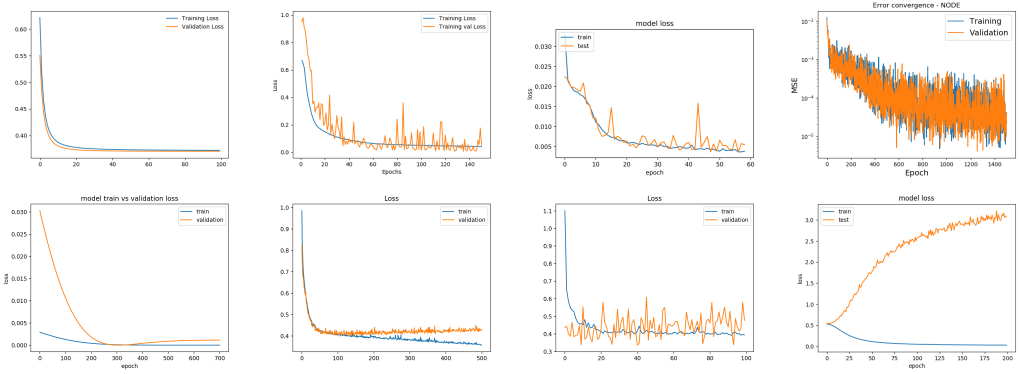
UFM UNIVERSIDAD FRANCISCO MARROQUÍN

Evaluation



Training/validation learning curves. The ideal case is the red curve.

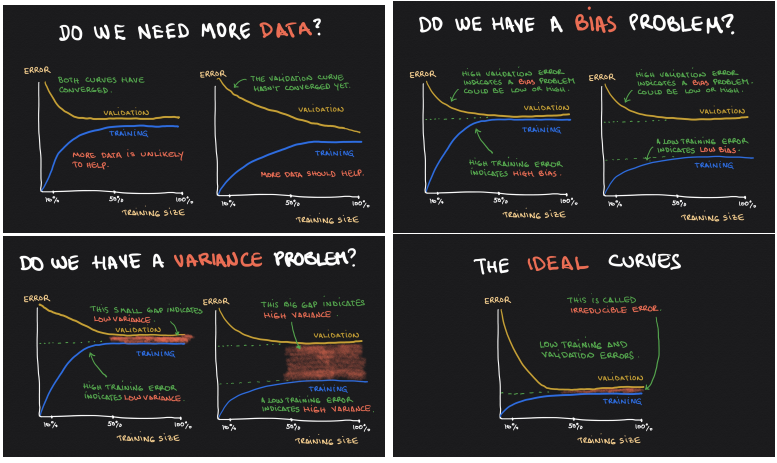
Evaluation



Examples of training/validation learning curves

UFM UNIVERSIDAD FRANCISCO MARROQUÍN

Evaluation



Other useful insights from the learning curves.

Implement neural networks in Keras

Libraries for deep learning

There are a lot of libraries and online resources to do deep learning.

Libraries for deep learning

The most popular are

Keras documentation: `https://keras.io/api/`

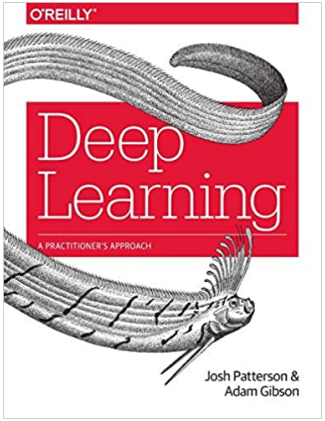
Keras

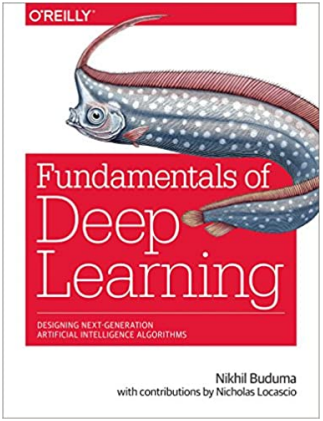
Keras was created by Francois Chollet (2015-2016), while working at Google. It began as an independent library, but today comes as a module inside Tensorflow.

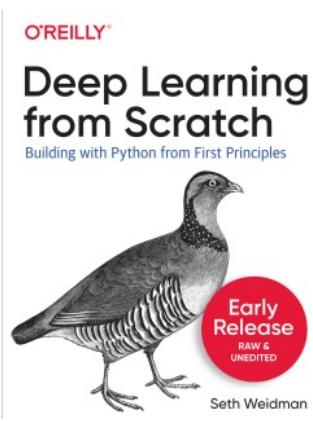
Books by F. Chollet. (2018). *Deep Learning with Python*. *Deep Learning with Python*. Manning.

Other References

Manning and O'Reilly are editorial with good practical books on deep learning, and machine learning in general (as well as other computational resources).



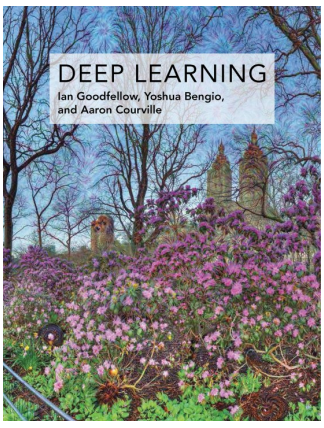


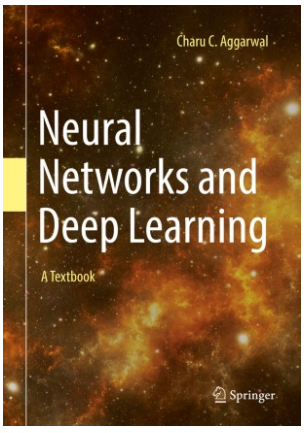


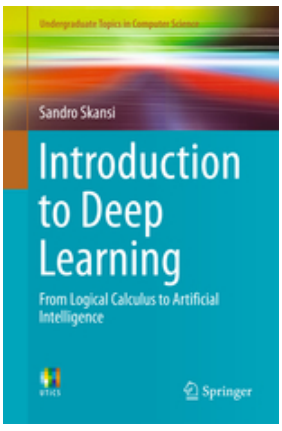
UFM UNIVERSIDAD FRANCISCO MARROQUÍN

Other References

There are few (serious) books on deep learning. The most part of material comes in the form of papers. In the web there are a lot of resources in form of blogs and tutorials.







Objectives for Practical Session 1

- Implement a fully-connected neural network on Keras, suitable for a regression problem.

- Learn how to prepare the data ready for input the neural network.
- Learn how to define the optimization algorithm.
- Learn how to define the loss function and evaluation metrics.
- Learn how to set some hyper-parameters: (learning rate, batch size, stop criteria).

- Fitting the model.
- Learn how to plot learning curves, and get useful information of the model from them.
- Learn how to predict from new data.
- Evaluate the performance on new data.