



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI
SCIENZE STATISTICHE "PAOLO FORTUNATI"



3 - Introduction to Deep Learning

Giovanni Della Lunga
giovanni.dellalunga@unibo.it

Halloween Conference in Quantitative Finance

Bologna - October 26-28, 2021

We will talk about...

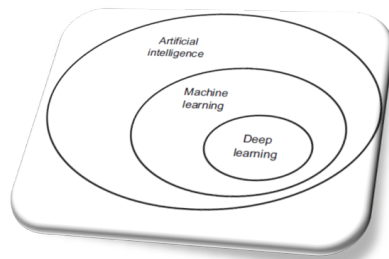
- What is a Neural Network
- Feedforward Neural Networks
- Keras: the Python Deep Learning API

Subsection 1

Machine Learning and Deep Learning

Introduction: Machine Learning and Deep Learning

- As we know to do machine learning we need three things:
- Input data points
- Examples of the expected output
- A way to measure whether the algorithm is doing a good job

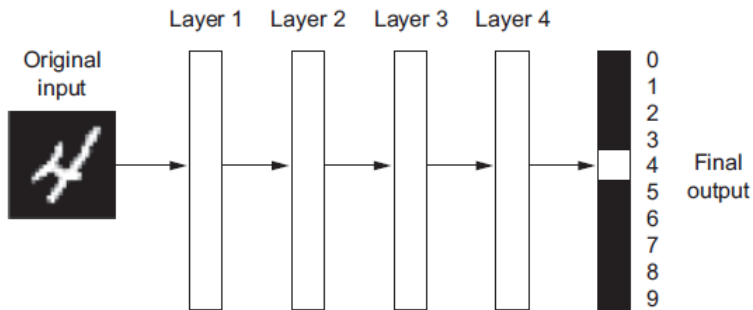


Introduction: Machine Learning and Deep Learning

- A machine-learning model transforms its input data into meaningful outputs, a process that is **learned** from exposure to known examples of inputs and outputs.
- Therefore, the central problem in machine learning and deep learning is to **meaningfully transform data**: in other words, **to learn useful representations of the input data at hand, representations that get us closer to the expected output**.
- What is a representation? At its core, it is simply a different way to look at data, to represent or encode data.

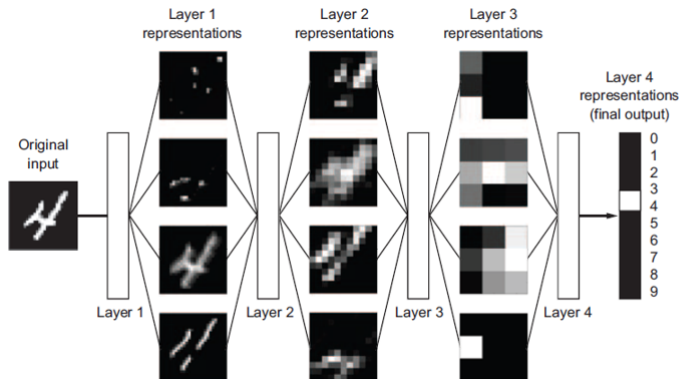
Introduction: Machine Learning and Deep Learning

Deep learning is a specific subfield of machine learning: a new take on learning representations from data that puts an emphasis on learning successive layers of increasingly meaningful representations.



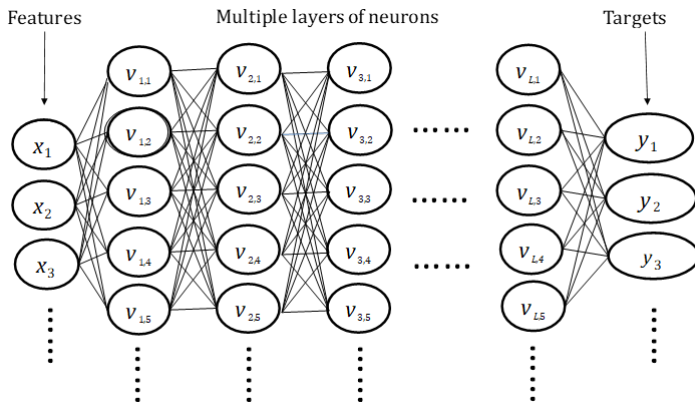
Introduction: Machine Learning and Deep Learning

Deep learning is a specific subfield of machine learning: a new take on learning representations from data that puts an emphasis on learning successive layers of increasingly meaningful representations.



Introduction: Machine Learning and Deep Learning

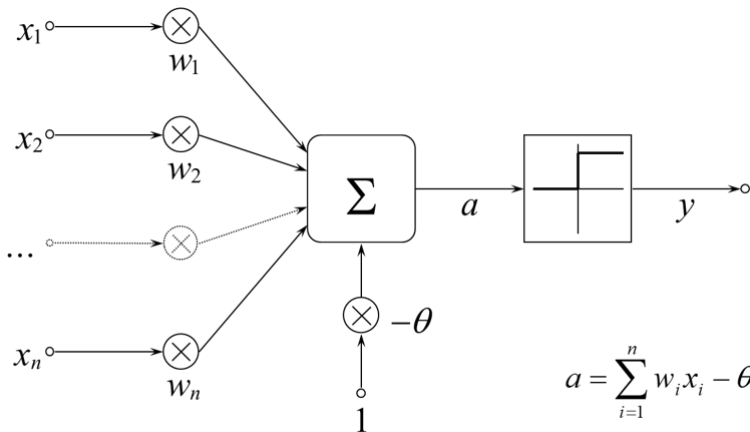
In deep learning, these layered representations are (almost always) learned via models called **neural networks**, structured in literal layers stacked on top of each other.



Subsection 2

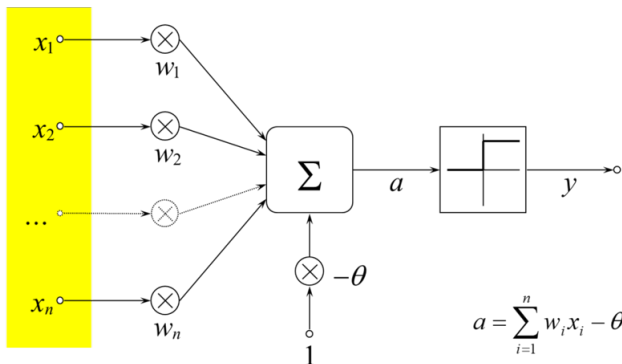
The McCulloch-Pitts Neuron

McCulloch and Pitts Neuron



NN Data Flow

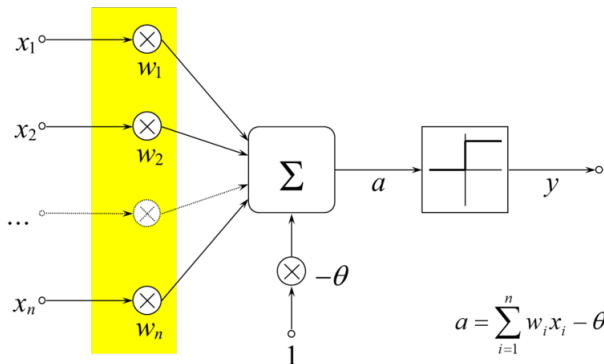
McCulloch and Pitts Neuron



INPUT DATA	WEIGHTS	WEIGHTED INPUT	BIAS	ACTIVATION FUNCTION	OUTPUT
------------	---------	----------------	------	---------------------	--------

NN Data Flow

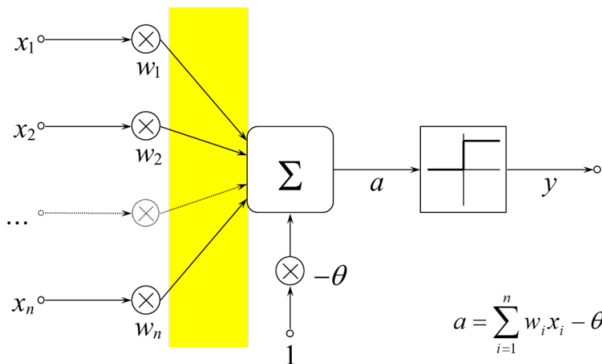
McCulloch and Pitts Neuron



INPUT DATA	WEIGHTS	WEIGHTED INPUT	BIAS	ACTIVATION FUNCTION	OUTPUT
------------	---------	----------------	------	---------------------	--------

NN Data Flow

McCulloch and Pitts Neuron



INPUT DATA	WEIGHTS	WEIGHTED INPUT	BIAS	ACTIVATION FUNCTION	OUTPUT
------------	---------	-----------------------	------	---------------------	--------

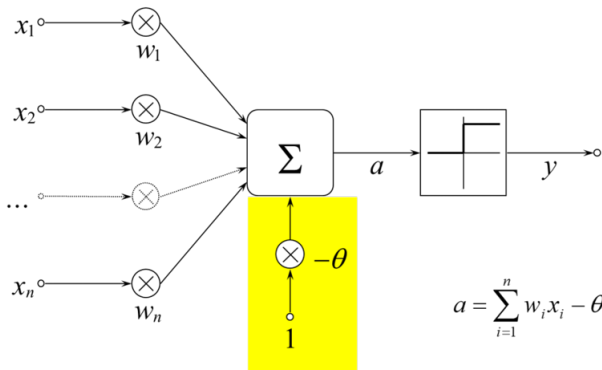
McCulloch and Pitts Neuron

From a functional point of view

- an input signal formally present but associated with a zero weight is equivalent to an absence of signal;
- the threshold can be considered as an additional synapse, connected in input with a fixed weight equal to 1;

NN Data Flow

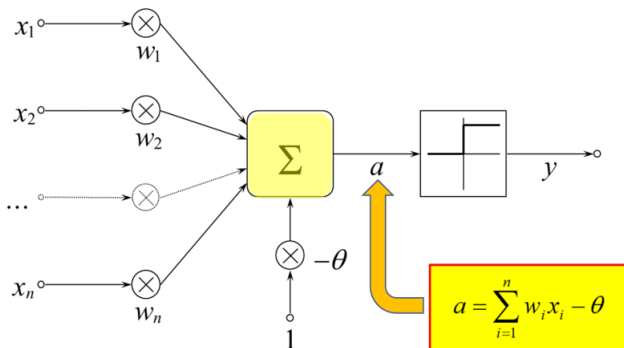
Mc-Culloch and Pitts Neuron



INPUT DATA WEIGHTS WEIGHTED INPUT **BIAS** ACTIVATION FUNCTION OUTPUT

NN Data Flow

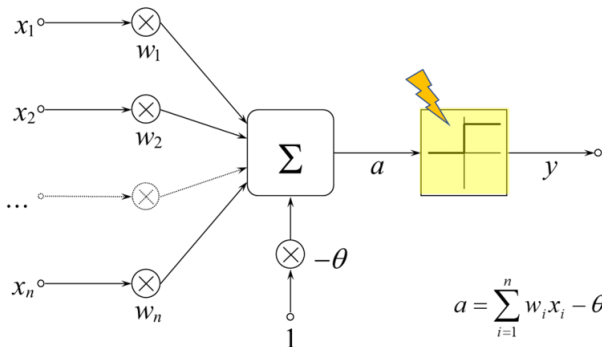
Mc-Culloch and Pitts Neuron



INPUT DATA WEIGHTS WEIGHTED INPUT BIAS ACTIVATION FUNCTION OUTPUT

NN Data Flow

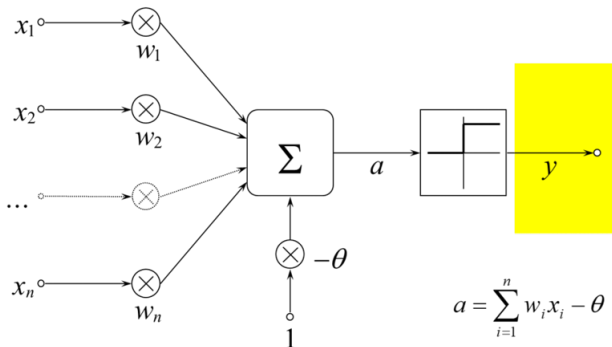
McCulloch and Pitts Neuron



INPUT DATA	WEIGHTS	WEIGHTED INPUT	BIAS	ACTIVATION FUNCTION	OUTPUT
------------	---------	----------------	------	----------------------------	--------

NN Data Flow

McCulloch and Pitts Neuron



INPUT DATA	WEIGHTS	WEIGHTED INPUT	BIAS	ACTIVATION FUNCTION	OUTPUT
------------	---------	----------------	------	---------------------	--------

Activation Function

$$a = \sum_{i=1}^n w_i x_i - \theta \quad (1)$$

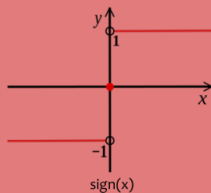
$$y = f(a) = \begin{cases} 0, & \text{if } a \leq 0 \\ 1, & \text{if } a > 0 \end{cases} \quad (2)$$

- The function f is called the response or activation function:
- in the McCulloch and Pitts neuron f is simply the step function, so the answer is binary: it is 1 if the weighted sum of the stimuli exceeds the internal threshold; 0 otherwise.
- Other models of artificial neurons predict continuous response functions

Activation Function

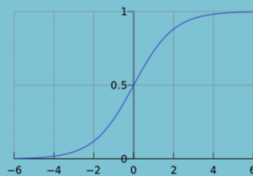
This decision function isn't differentiable:

$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$



Use a differentiable function instead:

$$p_{\boldsymbol{\theta}}(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x})}$$

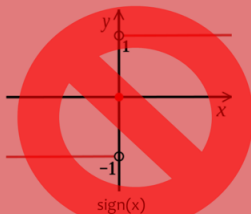


$$\text{logistic}(u) \equiv \frac{1}{1 + e^{-u}}$$

Activation Function

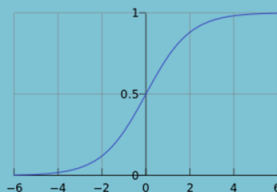
This decision function isn't differentiable:

$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$



Use a differentiable function instead:

$$p_{\theta}(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x})}$$



$$\text{logistic}(u) \equiv \frac{1}{1 + e^{-u}}$$

Neural Network Basic Constituents

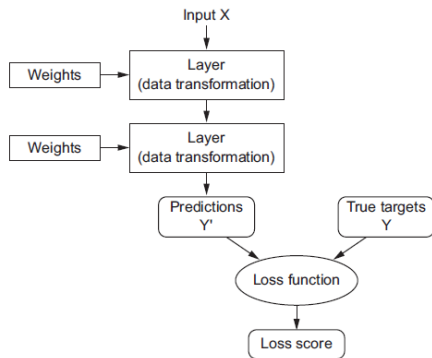
- A neural network consists of:
- A set of nodes (neurons), or units connected by links.
- A set of **weights** associated with links.
- A set of thresholds or activation levels.
- Neural network design requires:
 1. The choice of the number and type of units.
 2. The determination of the morphological structure.
 3. Coding of training examples, in terms of network inputs and outputs.
 4. Initialization and training of weights on interconnections, through the set of learning examples.

Neural Network Basic Constituents

- The specification of what a layer does to its input data is stored in the layer's weights, which in essence are a bunch of numbers.
- In technical terms, we could say that the transformation implemented by a layer is parameterized by its weights (Weights are also sometimes called the parameters of a layer.)
- In this context, **learning means finding a set of values for the weights of all layers in a network, such that the network will correctly map example inputs to their associated targets.**

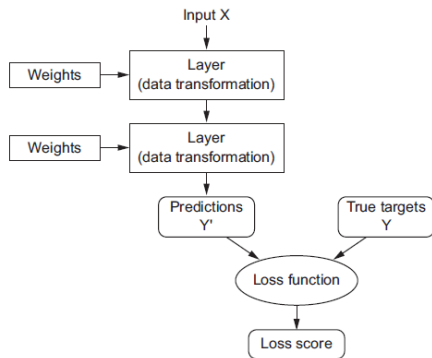
Loss Function

- To control the output of a neural network, you need to be able to measure how far this output is from what you expected.
- This is the job of the **loss function** of the network, also called the objective function.

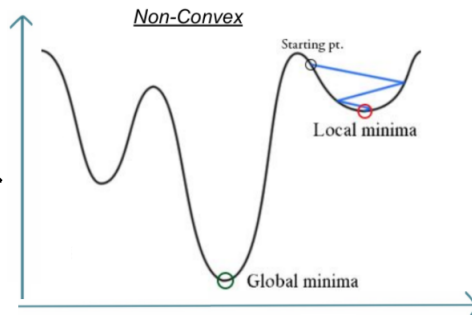
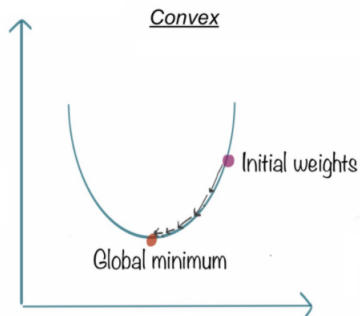


Loss Function

- The loss function takes the predictions of the network and the true target (what you wanted the network to output) and **computes a distance score, capturing how well the network has done on this specific example**

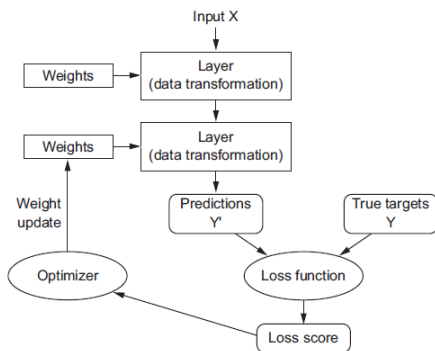


Loss Function



Backpropagation

- The fundamental trick in deep learning is to use this score as a feedback signal to adjust the value of the weights a little, in a direction that will lower the loss score for the current example.
- This adjustment is the job of the optimizer, which implements what is called the Backpropagation algorithm: the central algorithm in deep learning.

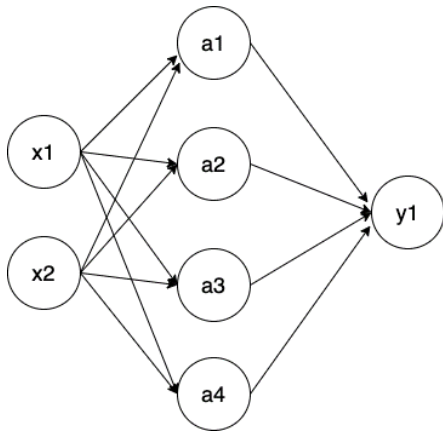


Implementing a Single Layer NN

In each hidden unit, take a_1 as example, a linear operation followed by an activation function, f , is performed. So given input $x = (x_1, x_2)$, inside node a_1 , we have:

$$z_1 = w_{11}x_1 + w_{12}x_2 + b_1$$

$$a_1 = f(w_{11}x_1 + w_{12}x_2 + b_1) = f(z_1)$$



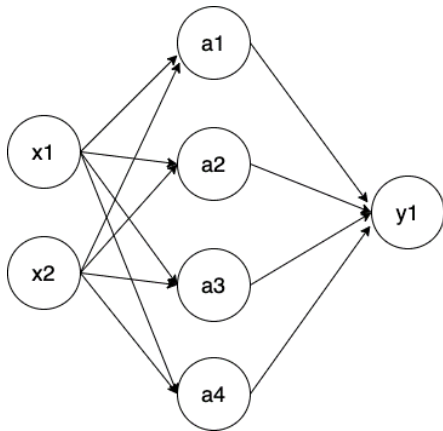
Implementing a Single Layer NN

Same for node a_2 , it would have:

$$z_2 = w_{21}x_1 + w_{22}x_2 + b_2$$

$$a_2 = f(w_{21}x_1 + w_{22}x_2 + b_2) = f(z_2)$$

And same for a_3 and a_4 and so on



Implementing a single Layer NN

We can also write in a more compact form

$$\begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} \Rightarrow Z^{[1]} = W^{[1]} \cdot X + B^{[1]} \quad (3)$$

The output is one value y_1 in $[0, 1]$, consider this a binary classification task with a prediction of probability

Implementing a single Layer NN

Let's assume that the first activation function is the \tanh and the output activation function is the *sigmoid*. So the result of the hidden layer is:

$$A^{[1]} = \tanh Z^{[1]}$$

This result is applied to the output node which will perform another linear operation with a different set of weights, $W^{[2]}$:

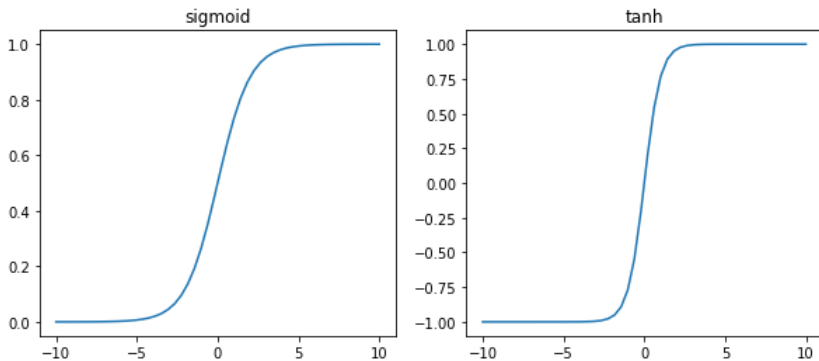
$$Z^{[2]} = W^{[2]} \cdot A^{[1]} + B^{[2]}$$

and the final output will be the result of the application of the output node activation function (the sigmoid) to this value:

$$\hat{y} = \sigma(Z^{[2]}) = A^{[2]}$$

Activation Functions

Function tanh and sigmoid looks as below. Notice that the only difference of these functions is the scale of y .



Logistic Loss Function

Since we have a binary classification problem, we can assume a Logistic Loss Function (see the problem of logistic regression)

$$L(y, \hat{y}) = \begin{cases} -\log \hat{y} & \text{when } y = 1 \\ -\log(1 - \hat{y}) & \text{when } y = 0 \end{cases} \quad (4)$$

$$L(y, \hat{y}) = -[y \log \hat{y} + (1 - y) \log (1 - \hat{y})]$$

Where \hat{y} is our **prediction** ranging in $[0, 1]$ and y is the **true** value.

Delta Rule

- Given a generic actual value y , we want to minimize the loss L , and the technic we are going to apply here is gradient descent;
- basically what we need to do is to apply derivative to our variables and move them slightly down to the optimum.
- Here we have 2 variables, W and b , and for this example, the update formula of them would be:

$$W = W - \frac{\partial L}{\partial W}$$
$$b = b - \frac{\partial L}{\partial b}$$

Delta Rule

- The delta rule algorithm works by computing the gradient of the loss function with respect to each weight.
- Remember that

$$\begin{aligned}
 \hat{y} &= A^{[2]} = \sigma(Z^{[2]} = \sigma(W^{[2]} \cdot A^{[1]} + B^{[2]}) \\
 &= \sigma(W^{[2]} \cdot \tanh Z^{[1]} + B^{[2]}) \\
 &= \sigma\left[W^{[2]} \cdot \tanh(W^{[1]} \cdot X + B^{[1]}) + B^{[2]}\right]
 \end{aligned}$$

- As you can see \hat{y} depends on both $W^{[1]}$ and $W^{[2]}$.

Delta Rule

- In order to get the derivative of our targets, chain rules would be applied:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial Z} \frac{\partial Z}{\partial W}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial Z} \frac{\partial Z}{\partial b}$$

- In particular the derivative of the Loss Function with respect to \hat{y} is very easy and can be calculated once for all because it does not depend on the particular layer:

$$L(y, \hat{y}) = -[y \log \hat{y} + (1 - y) \log (1 - \hat{y})]$$

$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

$$= \frac{A^{[2]} - y}{A^{[2]}(1 - A^{[2]})}$$

Gradient Calculation

Hidden Layer Activation Function (Hyperbolic Tangent)

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}} \Rightarrow \frac{d}{dx} \tanh x = 1 - (\tanh x)^2 \quad (5)$$

Output Layer Activation Function (Sigmoid Function)

$$\sigma(x) = \left[\frac{1}{1 + e^{-x}} \right] \Rightarrow \frac{d}{dx} \sigma(x) = \sigma(x) \cdot (1 - \sigma(x)) \quad (6)$$

Gradient Calculation

Output Layer

$$\frac{\partial L}{\partial A^{[2]}} = \frac{A^{[2]} - y}{A^{[2]}(1 - A^{[2]})}$$

$$\frac{\partial A^{[2]}}{\partial Z^{[2]}} = \frac{\partial \sigma(Z^{[2]})}{\partial Z^{[2]}} = \sigma(Z^{[2]}) \cdot (1 - \sigma(Z^{[2]})) = A^{[2]}(1 - A^{[2]})$$

$$\frac{\partial Z^{[2]}}{\partial W^{[2]}} = A^{[1]}$$

So the complete gradient is:

$$\begin{aligned} \frac{\partial L}{\partial W^{[2]}} &= \frac{A^{[2]} - y}{A^{[2]}(1 - A^{[2]})} \cdot A^{[2]}(1 - A^{[2]}) \cdot A^{[1]T} \\ &= (A^{[2]} - y) \cdot A^{[1]T} \end{aligned}$$

Gradient Calculation

Hidden Layer Now we have to calculate

$$\frac{\partial Z^{[2]}}{\partial W^{[1]}}$$

Remember that

$$Z^{[2]} = W^{[2]} \cdot \tanh \left(W^{[1]} \cdot X + b^{[1]} \right) + b^{[2]}$$

and

$$\frac{\partial Z^{[2]}}{\partial W^{[1]}} = W^{[2]} \cdot \frac{\partial \tanh(\dots)}{\partial W^{[1]}} \cdot X = W^{[2]} \cdot (1 - \tanh^2(\dots)) \cdot X$$

Gradient Calculation

Hidden Layer

Finally

$$\begin{aligned}\frac{\partial L}{\partial W^{[1]}} &= (A^{[2]} - y) \cdot W^{[2]} \cdot X \cdot (1 - \tanh^2(\dots)) \\ &= (A^{[2]} - y) \cdot W^{[2]} \cdot X \cdot (1 - A^{[1]^2})\end{aligned}$$

Weights Update

Defining

$$\Delta^{[2]} = A^{[2]} - Y \quad (7)$$

$$\Delta^{[1]} = \Delta^{[2]} \cdot W^{[2]T} \cdot (1 - A^{[1]})^2 \quad (8)$$

We have

Output Layer

$$dW^{[2]} = \frac{1}{m} [A^{[2]} - Y] A^{[1]T} = \frac{1}{m} \Delta^{[2]} A^{[1]T} \quad (9)$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True) \quad (10)$$

Weights Update

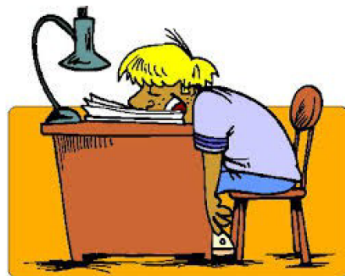
Hidden Layer

$$\begin{aligned}
 dW^{[1]} &= \frac{1}{m} \left[A^{[2]} - Y \right] \cdot X^T \cdot W^{[2]T} \cdot (1 - A^{[1]^2}) \\
 &= \frac{1}{m} \Delta^{[2]} \cdot W^{[2]T} \cdot (1 - A^{[1]^2}) \cdot X^T \\
 &= \frac{1}{m} \Delta^{[1]} \cdot X^T
 \end{aligned} \tag{11}$$

$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True) \tag{12}$$

Example: Implementing a Shallow Neural Network

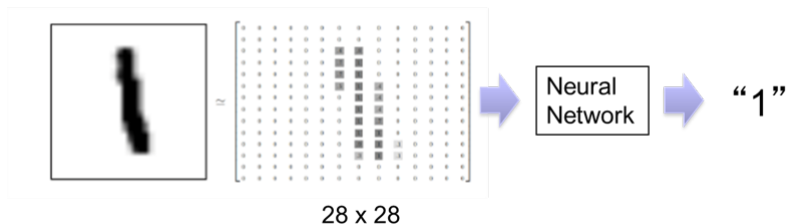
- Using **03-introduction-to-deep-learning** Notebook
- Par. 3.3



Subsection 3

Keras

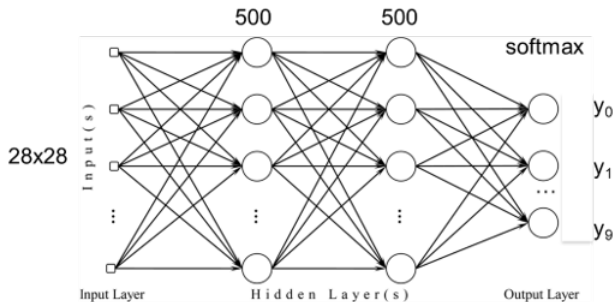
Introduction to keras



- 0-9 handwritten digit recognition:
- MNIST Data maintained by Yann LeCun:
<http://yann.lecun.com/exdb/mnist/>
- Keras provides data sets loading function at <http://keras.io/datasets>

Implementing in Keras

Fully Connected NN



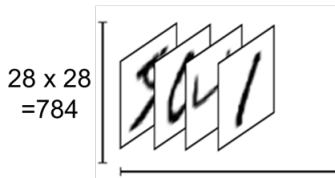
```
model = sequential() # layers are sequentially added
model.add(Dense(input_dim=28*28, output_dim=500))
model.add(Activation('sigmoid')) #: softplus, softsign, relu, tanh, hard_sigmoid
model.add(Dense(output_dim = 500))
model.add(Activation('sigmoid'))
model.add(Dense(output_dim=10))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

Implementing in Keras

Training

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

numpy array



Number of training examples



Number of training examples