Technical introduction to Neural Networks

October 4, 2019

Elementary neuron

A **neuron** is a **mapping** from a multidimensional input $x = (x_1, ..., x_n)$ to a real number y.

Elementary neuron

A **neuron** is a **mapping** from a multidimensional input $x = (x_1, ..., x_n)$ to a real number y. In it's simplest form, this function depends on **parameters** called **weights** $w = (w_1, ..., w_n)$. We can see it as a function $f : x \to y$ with :

$$y = \sigma(\sum_{i=1}^{n} x_i w_i) \tag{1}$$

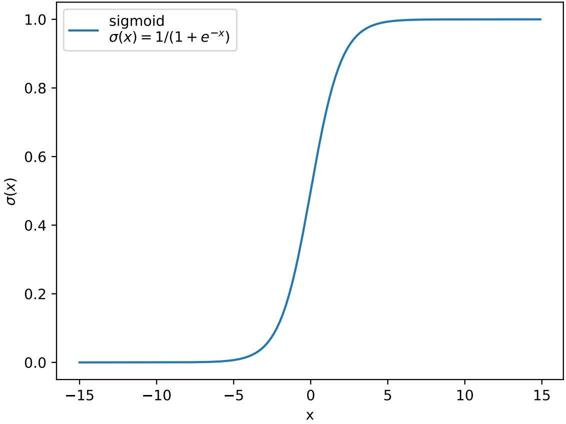
Elementary neuron

A **neuron** is a **mapping** from a multidimensional input $x = (x_1, ..., x_n)$ to a real number y. In it's simplest form, this function depends on **parameters** called **weights** $w = (w_1, ..., w_n)$. We can see it as a function $f : x \rightarrow y$ with :

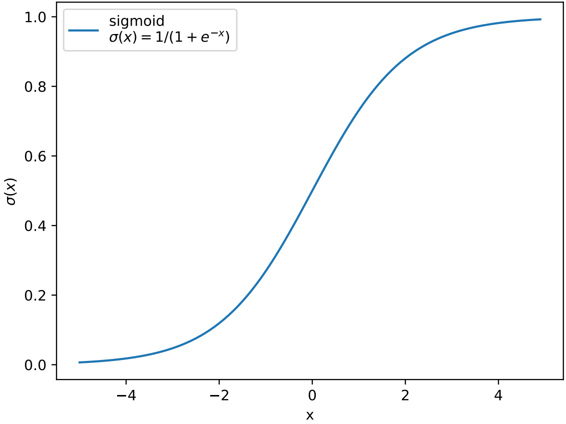
$$y = \sigma(\sum_{i=1}^{n} x_i w_i) \tag{2}$$

Where σ is a non linear function, for instance a **sigmoid**.

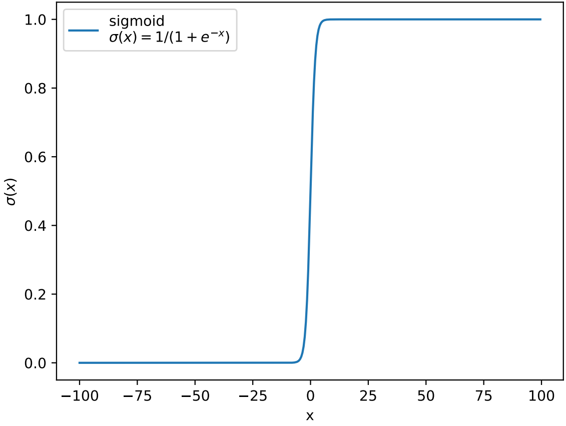
Sigmoid function



Sigmoid function



Sigmoid function



Notation with vectors

The sum $\sum_{i=1}^{n} x_i w_i$ can also be written this way :

$$xw^T \tag{3}$$

This means a **product** of **two matrices** (a vector is also a matrix : it is just a matrix with only one line or only one column) : :

Notation with vectors

The sum $\sum_{i=1}^{n} x_i w_i$ can also be written this way :

$$xw^T \tag{4}$$

This means a **product** of **two matrices** (a vector is also a matrix : it is just a matrix with only one line or only one column) : :

- $x = (x_1, ..., x_n)$
- $$w^T = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}$$

Matrices

A matrix is an array used to store data. It has **lines** and **columns**

$$A = \begin{pmatrix} 1 & 4 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Matrices

A matrix is an array used to store data. It has **lines** and **columns**

$$A = \begin{pmatrix} 1 & 4 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

A_{ij} means the element at line i and columns j.

Matrices

A matrix is an array used to store data. It has **lines** and **columns**

$$A = \begin{pmatrix} 1 & 4 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

A_{ij} means the element at line i and columns j.

- $A_{12} = ?$
- $A_{31} = ?$
- $A_{33} = ?$

Matrices

A matrix is an array used to store data. It has **lines** and **columns**

$$A = \begin{pmatrix} 1 & 4 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

A_{ij} means the element at line i and columns j.

- $A_{12} = 4$
- $A_{31} = 0$
- $A_{33} = 1$

Product of matrices

- If matrix A has p columns and matrix B has p lines, we can compute the product of the two AB of the two matrices in the following way :

$$AB_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} \tag{5}$$

Product of matrices

- If matrix A has p columns and matrix B has p lines, we can compute the product of the two AB of the two matrices in the following way :

$$AB_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} \tag{6}$$

- This kind of computation is very often used

Product of matrices

- If matrix A has p columns and matrix B has p lines, we can compute the product of the two AB of the two matrices in the following way :

$$AB_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} \tag{7}$$

- This kind of computation is very often used
- It is way more convenient and concise to use
- We will use it when studying neural networks

Examples

$$\begin{pmatrix} 1 & 4 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = ?$$

Examples

$$\begin{pmatrix} 1 & 4 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Examples

$$\begin{pmatrix} 1 & 4 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = ?$$

Examples

$$\begin{pmatrix} 1 & 4 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 4 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Examples

$$\begin{pmatrix} 1 & 4 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \end{pmatrix} = ?$$

Examples

$$\begin{pmatrix} 1 & 4 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 8 & 0 \\ 2 & 2 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Examples

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

What is A^n ?

Examples

- $x = (x_1, ..., x_n)$
- $w = (w_1, ..., w_n)$

$$xw^T = ? \tag{8}$$

Examples

- $x = (x_1, ..., x_n)$
- $w = (w_1, ..., w_n)$

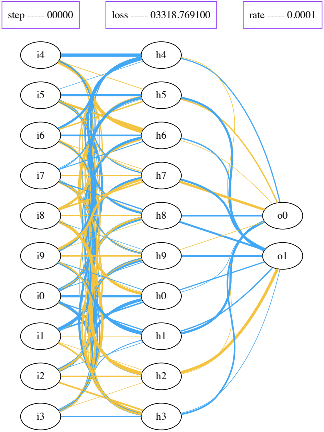
$$xw^T = \sum_{i=1}^{n} x_i w_i \tag{9}$$

Neural networks

- $\sigma(xw^T)$ allows us to compute the output of a **single neuron**
- But we will often have **several neurons** outputting a result.
- These neurons are organized in a network called neural network.

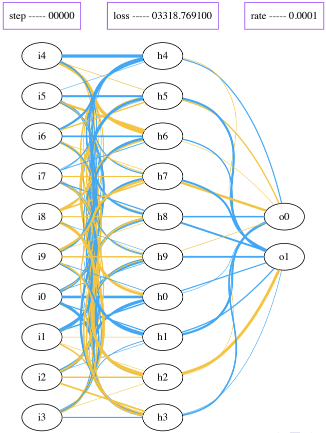
Neural networks

- These neurons are organized in a network called neural network.



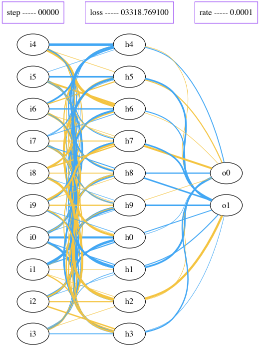
Matrices and networks

- Let $W = [W_{ij}]$ be the matrices of weights between neuron i of the left layer and neuron j of the middle layer.



Matrices and networks

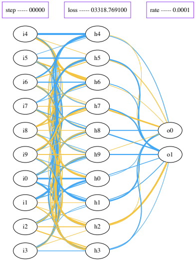
- Let $W = [W_{ij}]$ be the matrices of weights between neuron i of the left layer and neuron j of the middle layer.
- How can we write the inputs b_j of the middle layer as a function of the outpus x_k of the left layer ?



Matrices and networks

- let $w = [w_{ij}]$ be the matrices of weights between neuron i of the left layer and neuron j of the middle layer.
- how can we write the inputs b_j of the middle layer as a function of the outpus x_k of the left layer ?

$$b_j = \sum_{k=1}^{n} x_k w_{kj} \tag{10}$$

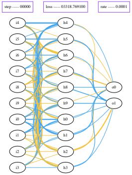


Matrices and networks

- let $w = [w_{ij}]$ be the matrices of weights between neuron i of the left layer and neuron j of the middle layer.

$$b_j = \sum_{k=1}^{n} x_k w_{kj} \tag{11}$$

- And in terms of matrices ? with :
  - $b = (b_1, ..., b_n)$
  - $x = (x_1, .., x_n)$



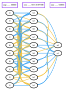
Matrices and networks

- let $w = [w_{ij}]$ be the matrices of weights between neuron i of the left layer and neuron j of the middle layer.

$$b_j = \sum_{k=1}^{n} x_k w_{kj} \tag{12}$$

- And in terms of matrices ? with :
  - $b = (b_1, ..., b_n)$
  - $x = (x_1, .., x_n)$

$$b = xw \tag{13}$$

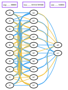


Matrices and networks

- let $w = [w_{ij}]$ be the matrices of weights between neuron i of the left layer and neuron j of the middle layer.

$$b_j = \sum_{k=1}^{n} x_k w_{kj} \tag{14}$$

- Finally, if we want to store the outputs for several input vectors x ?

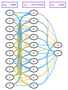


Matrices and networks

- let $w = [w_{ij}]$ be the matrices of weights between neuron i of the left layer and neuron j of the middle layer.

$$b_j = \sum_{k=1}^{n} x_k w_{kj} \tag{15}$$

- Finally, if we want to store the outputs for several input vectors x ?
  - use matrices $x = [x_{ij}]$ and $b = [b_{ij}]$
  - $$b = xw \tag{16}$$

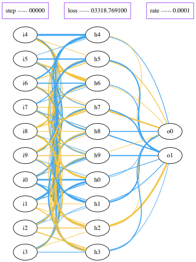


Layers

- We now know how to compute the input b of a layer as a function of the output x of the previous layer

$$b = xw \tag{17}$$

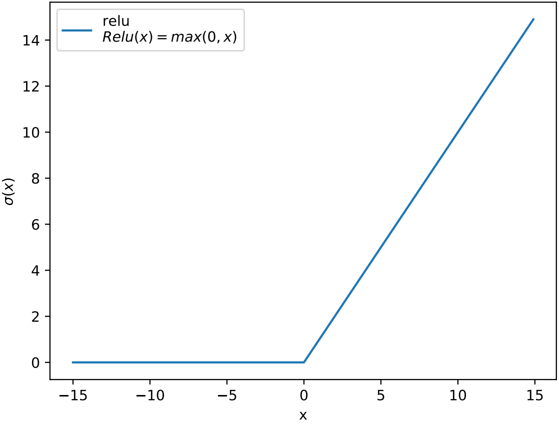
- Applying this rule **and** the non linearity σ, we can coompute the **forward propagation** of a neural network.



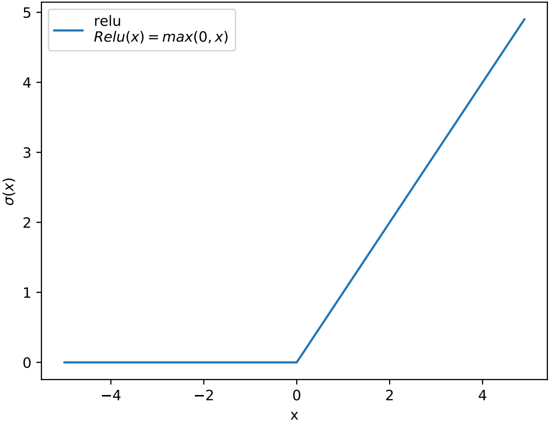
Layers and forward propagation

- We will use the **numpy** library to do so.
- In numpy, the **.dot** function is used to compute products of matrices
- We will use de ReLu non linearity.

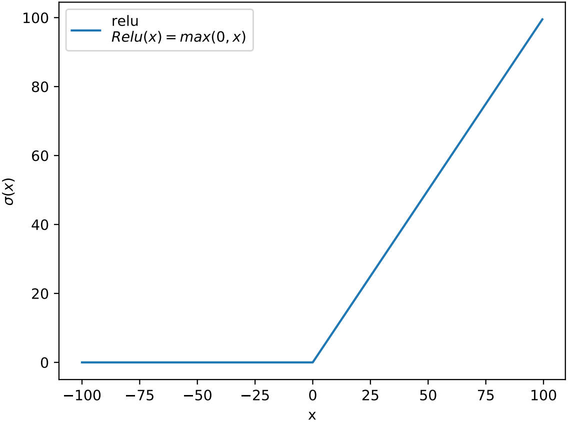
Relu function



Relu function



Relu function



Cost

- We saw how to compute the output of a neural network given an input
- But what do we want to do with the network ?

Cost

- We saw how to compute the output of a neural network given an input
- But what do we want to do with the network ?
- We want to solve a given problem, for instance a **supervised learning problem**
- This means being able to predict the output as a function of an input.

Supervised learning

- In order to learn the prediction, we **optimize** our network with training examples
- What exactly do we want to optimize ?

Supervised learning

- In order to learn the prediction, we **optimize** our network with training examples
- What exactly do we want to optimize ?
- We want to **minimize the error on the training examples.**

Supervised learning

- In order to learn the prediction, we **optimize** our network with training examples
- What exactly do we want to optimize ?
- We want to **minimize the error on the training examples.**
- We will use the Squared Error

$$\sum_{\text{training samples}} \text{error}^2 \tag{18}$$

Supervised learning

- In order to learn the prediction, we **optimize** our network with training examples
- What exactly do we want to optimize ?
- We want to **minimize the error on the training examples.**
- We will use the Squared Error

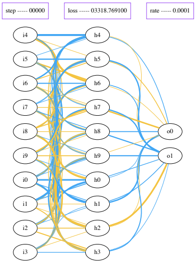
$$SE = \sum_{\text{training samples}} ||y_{\text{prediction}} - y_{\text{truth}}||^2 \tag{19}$$

Optimization

- What are the **parameters** of the network ? ie : what we have control on and what we can change in order to minimize the error.

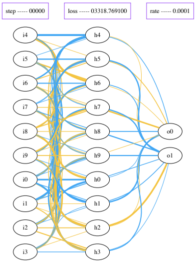
Optimization

- What are the **parameters** of the network ? ie : what we have control on and what we can change in order to minimize the error.
- The **weights** w_1 and w_2.
- In our examples we will use a network with three layers : input - hidden - output.

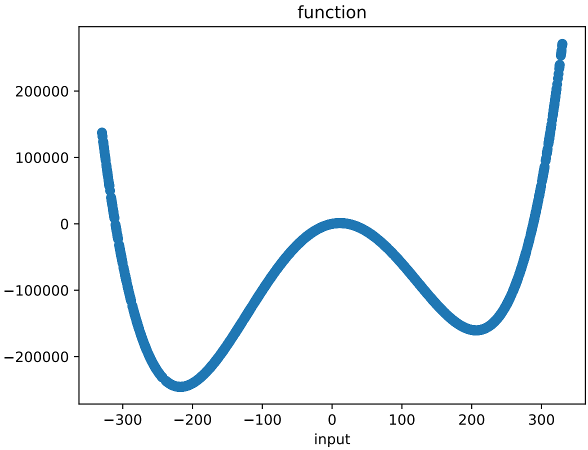


Optimization

- What are the **parameters** of the network ? ie : what we have control on and what we can change in order to minimize the error.
- The **weights matrices** w_1 and w_2.
- In our examples we will use a network with three layers : input - hidden - output.
- We wee the loss as a function $L(w1, w2)$



Minima of a function



Gradient descent

- In the case a function $f : \mathbb{R} \to \mathbb{R}$, we can study its variations by computing its derivative f', **if it exists**

Gradient descent

- In the case a function $f : \mathbb{R} \to \mathbb{R}$, we can study its variations by computing its derivative f', **if it exists**
- If $f'(x) > 0$, the function grows around x.
- If $f'(x) < 0$, the function decreases around x.
- If x is a local extremum, $f'(x) = 0$

Gradient descent

- In the case a function $f : \mathbb{R} \to \mathbb{R}$, we can study its variations by computing its derivative f', **if it exists**
- If $f'(x) > 0$, the function grows around x.
- If $f'(x) < 0$, the function decreases around x.
- If x is a local extremum, $f'(x) = 0$
- Is the reciprocal true ?

Derivation

- We can use the derivative to look for a minimum value for the function
- Example with analytic solution

Analytic minimum

What is the minimum of the function

$$f : x \to (x-1)^2 + 3.5 \tag{20}$$

And for what value x is it obtained ?

Gradient

- The **gradient** is similar to a derivative but in the case of a function with several inputs, such as our loss $l(w1, w2)$.
- Then we store the **partial derivative** with respect to each input in a **vector** called the gradient.
- Let us compute for instance the partial derivative with respect to $w1$

Gradient

- Then we store the **partial derivative** with respect to each input in a **vector** called the gradient.
- Let us compute for instance the partial derivative with respect to $w1$

$$\frac{\partial L}{\partial w1} = h^T 2(y_{\text{predicted}} - y_{truth}) \tag{21}$$

(where h is the output of the relu)

Backpropagation

- By repeating the same process we can also compute the gradient with respect to $w2$.
- This is called **backpropagation**.
- Knowing the gradient, we can **update the network parameters**.

Gradient update

- In one dimension :

$$x \leftarrow x - \alpha f'(x) \tag{22}$$

- In more dimensions :

$$w \leftarrow w - \alpha \nabla_w (f) \tag{23}$$

- α is the **learning rate.**

Application to our neural network

- We will apply this to do some supervised learning over two datasets:
  - A random dataset
  - A structured dataset

Libs

- We will need **numpy**
- **pygraphviz**
- optionally **pytorch**

Optimizing a neural network

Exercice 1 : **Random dataset**

- **cd ./neural_net**
- Use **learn_random_data.py** to learn to predict some output as a function of some input.
- You will need to tune the **hyperparameters** : these are the parameters setting up the network size, learning rate, etc.

Local minimum

Exercice 2 : **Not global minimum**

- Now make it find a local minimum that is not global

Overflow

Exercice 3 : **Explosion.**

- Now make it explode (diverge)

Plotting

Exercice 4 : **Observation of the network.**

- Uncomment the lines calling **show_net** so plot the evolution of the network
- You might need to use a smaller network otherwise it will be too long.

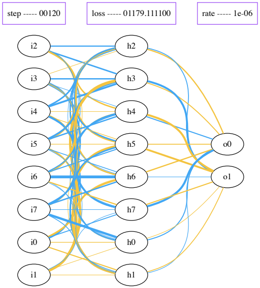
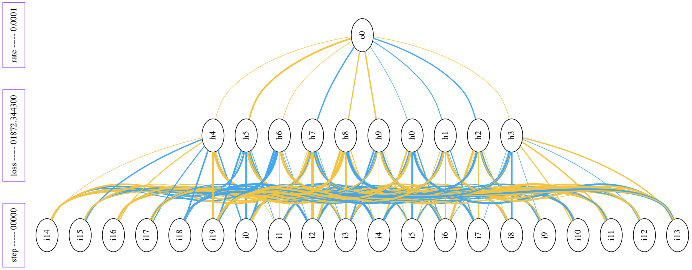
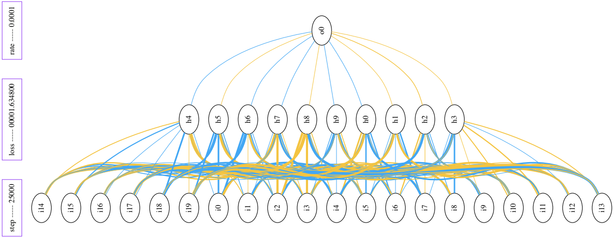


Figure:

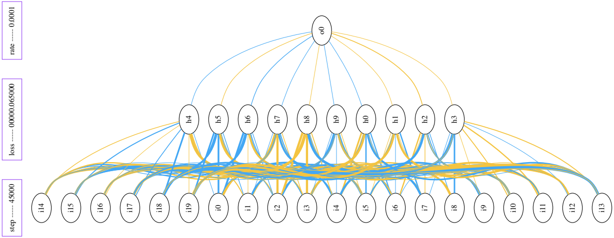
Initial network



After 25000 steps



After 45000 steps



- Does this make sense ?
- Can we generalize what we learned ?

- Does this make sense ?
- Can we generalize what we learned ?
- Since the data are random, and not correlated, we are not learning any **structure** in them.

- Does this make sense ?
- Can we generalize what we learned ?
- Since the data are random, and not correlated, we are not learning any **structure** in them.
- What is the test error ? (use the relevant function)

Learning structured data

Exercice 5 : **Learning structure.**

- We will now use structured data
- They are artificially generated in **create_structured_data.py** by a function, with a noise that you can choose.
- You can tune the standard deviation of the noise
- Use **create_structured_data.py** and **learn_structured_data.py** to generate a dataset and learn it in a supervised way

Modifying the network

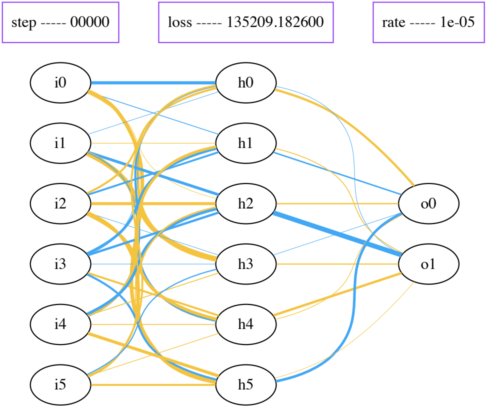
Exercice 6 : **Playing with hyperparameters**

- Make it find a bad local minimum, make it explode.

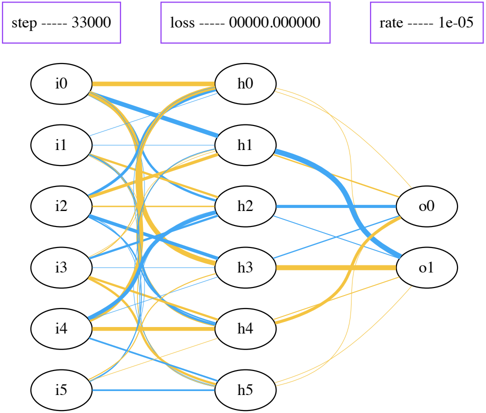
Observing the network.

- Uncomment the relevant stuff to visualize the evolution of the network with graphviz
- You wan choose the number of hidden layers

Structured data



Structured data



- How does the test error behave ?

- How does the test error behave ?
- It is a small as the training error.
- Now it makes more sense since we worked on data with structure. We did not overfit.

With libs

- We did things manually with numpy but when the networks are large or when we need to automate the seach for good parameters, it is more convenient to work with libraries such as :
  - pytorch
  - tensorflow
  - keras

Many techniques

- There are lots of variations around neural nets
  - in the number of neurons per layer
  - type of data processed
  - relationship between weights (shared weights)
  - number of hidden layers
  - recurrent neural networks RNN
  - convolutional neural networks CNN

Stochastic gradient

- Another method to compute the gradient

Other cost functions

- Until now we used the squared error cost function
- The slowdown problem
- The Cross entropy is another possible cost function used for classification

MNIST

- We will apply this theory to the canonical example MNIST

MNIST

- We will apply this theory to the canonical example MNIST
- With keras and tensorflow, we will achieve an accuracy of more than 97% in a few minutes for the classification.

Inputs

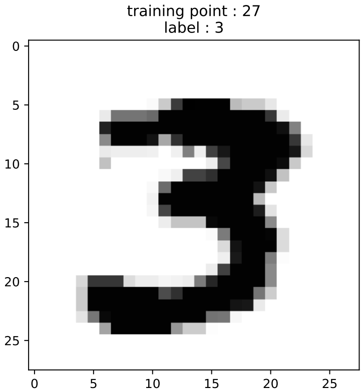


Figure: Datapoint 27

Inputs

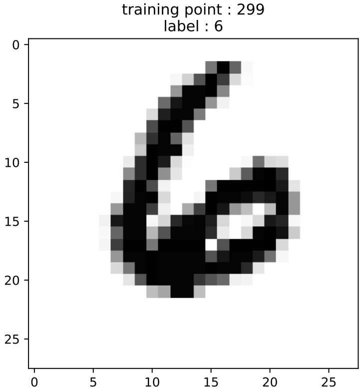


Figure: Datapoint 299

Inputs

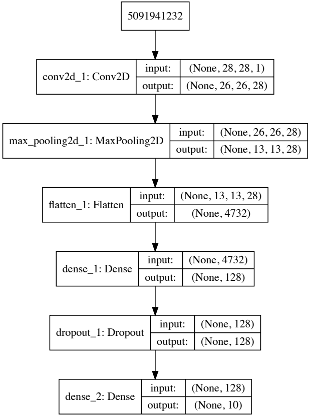


Figure: Network shape

Inputs

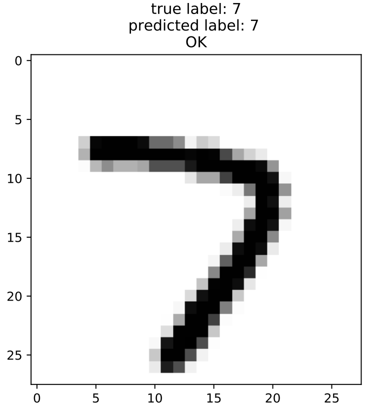


Figure: Predicion for point 17

Inputs

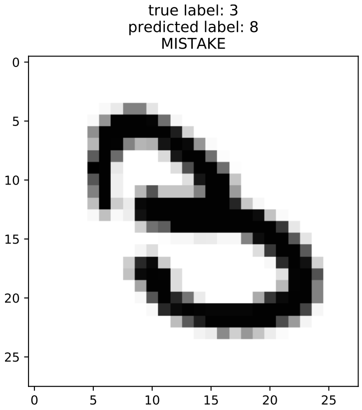


Figure: Prediction for point 18