Deep Learning Introduction

Lesson 1

Tiago Cabo

Lesson Plan

- What is deep Learning
- Why deep learning is such big thing now?
- Be able to recognize different kinds of deep learning
- Be able to understand the building blocks of deep neural network

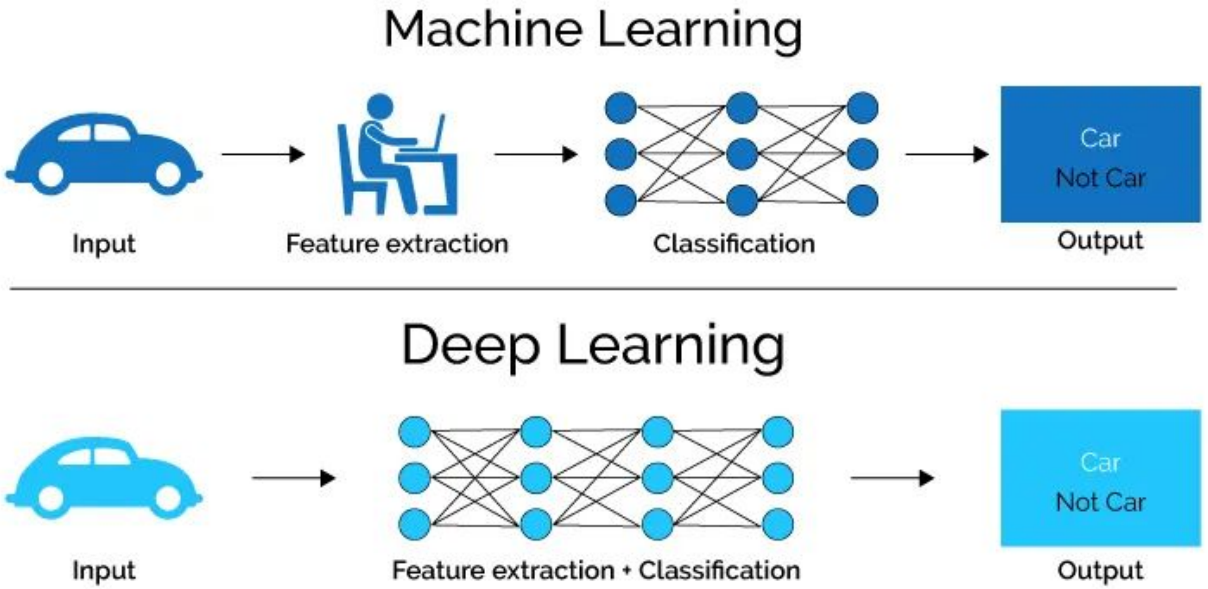
Lesson goals

- Be able to explain what deep learning is
- Be able to enumerate the main building blocks of deep neural networks

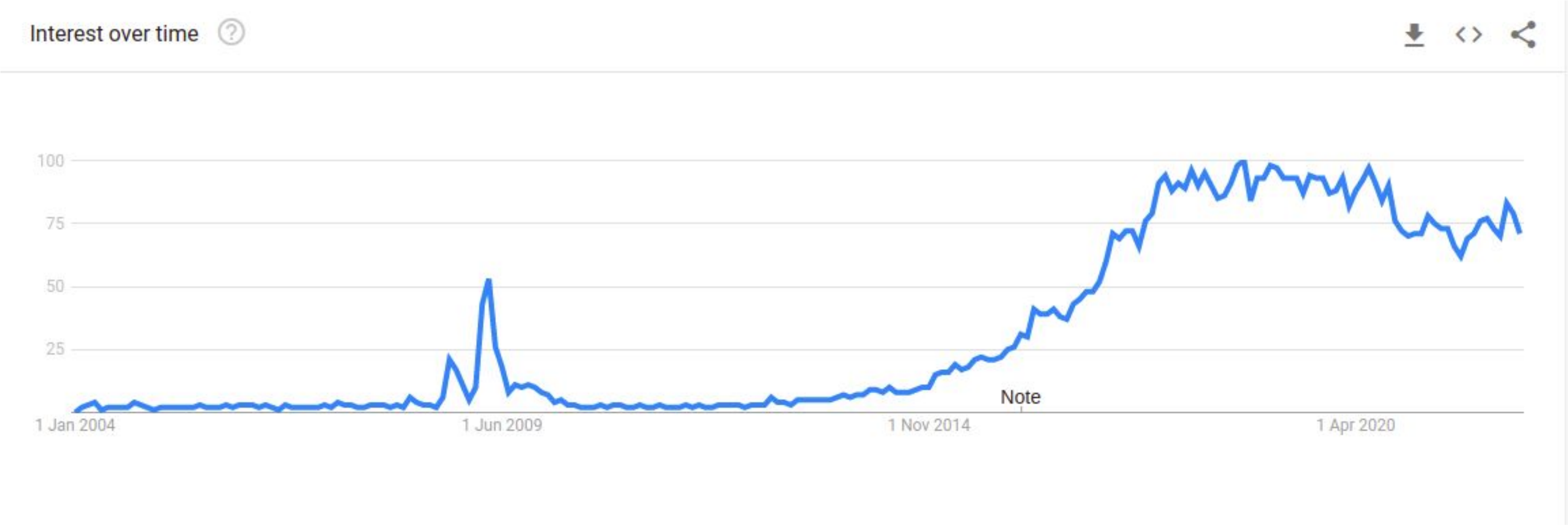
What is deep learning?

Deep Learning

Deep Learning

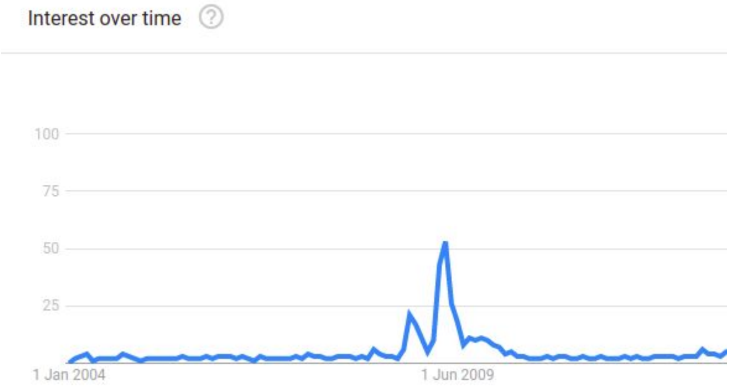


Deep Learning Neural Networks



What happened in 2009?

In 2009 a join of events raised a lot of headlight to the field either my publishing major breakthroughs in the algorithms, but also was the first presentation of the practical uses of GPUs for Deep learning training.



Example

https://this-person-does-not-exist.com/en

Deep Learning Introduction

One of the biggest advantages of NN algorithms is the possibility of reducing the amount of manual work during the feature engineering process.

Imagine having to perform feature extraction in pictures?

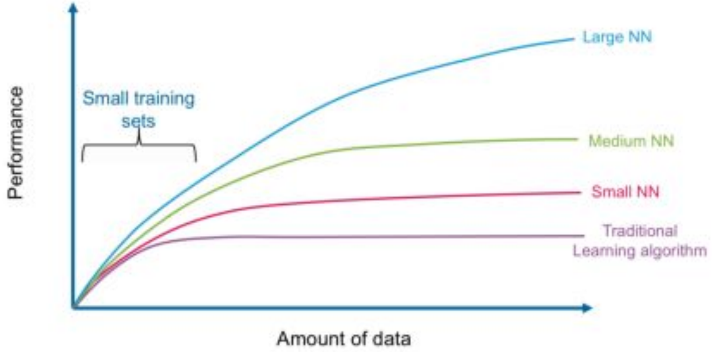
Removing this feature extractor process and nn still today cannot beat for example XGBoost algorithms in tabular data. However, DL masters:

- Image
- Speach
- Text

Why current boom?

Similar the normal Machine Learning deep learning benefits from the easily available data, but in this case the data needs to be huge!

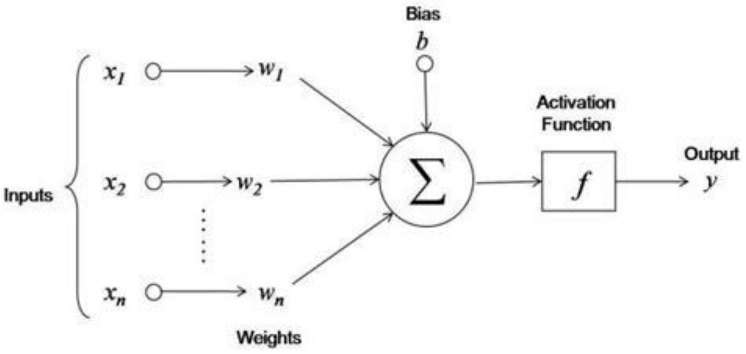
Also the hardware make huge difference. As we saw earlier, there is a before and after the use of GPU.

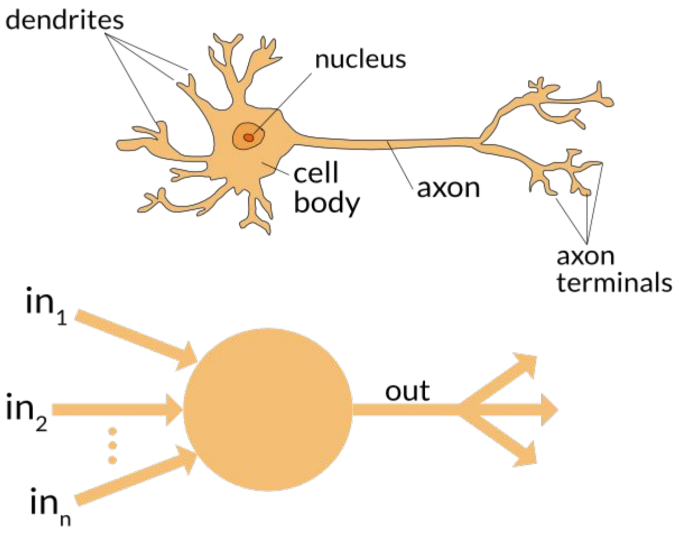


Neural Networks

Let’s deep dive on neural networks. Most concepts in deep learning try to mimic the human brain.

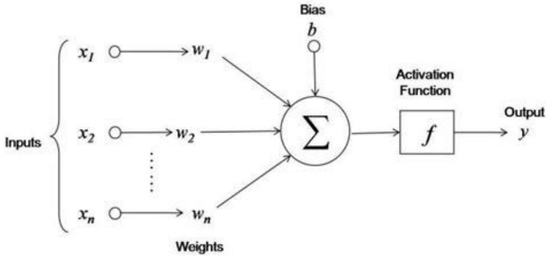
In the first days it was called perceptron, and it was invented in 1958 at Cornell Aeronautical by Frank Rosenblatt.



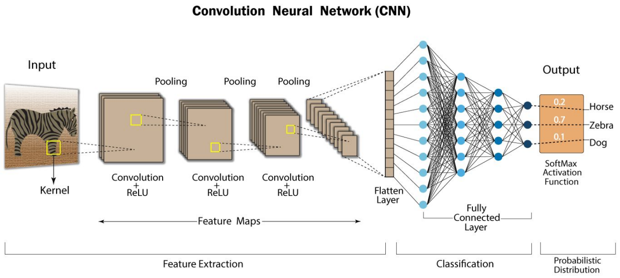


Types of Deep Learning

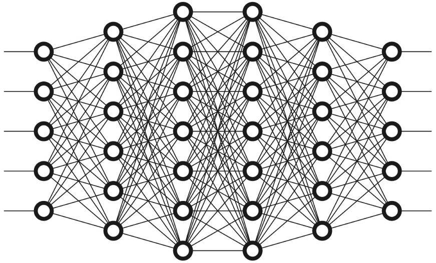
NN



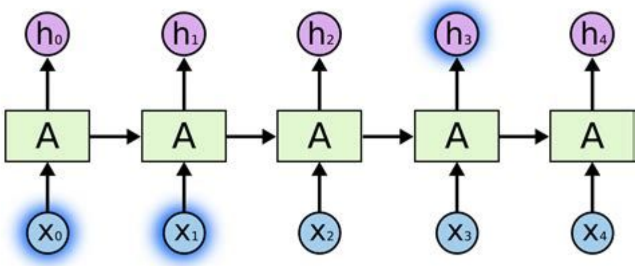
CNN



DNN (Deep neural network)

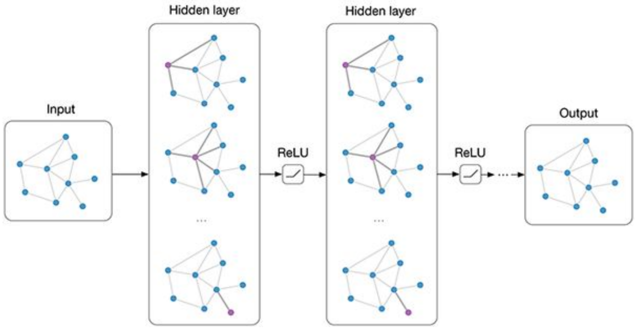


RNN

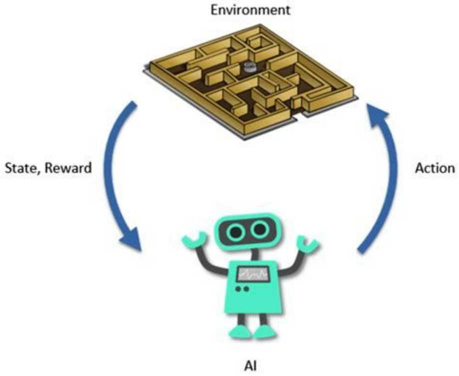


Other types

Graph neural networks

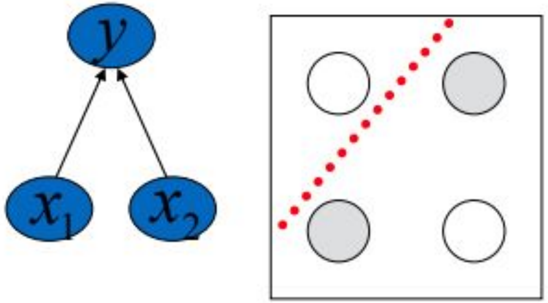


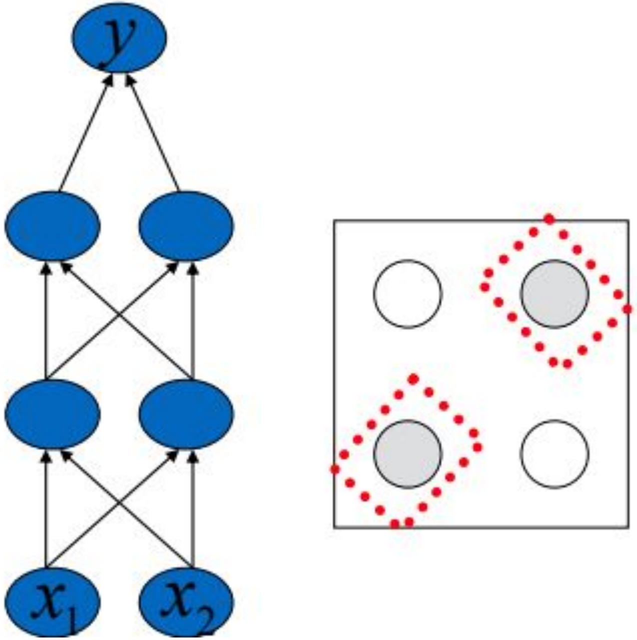
Reinforcement Learning



Number of layers effect

The higher the number of decision layers, more complex the decision boundary is.

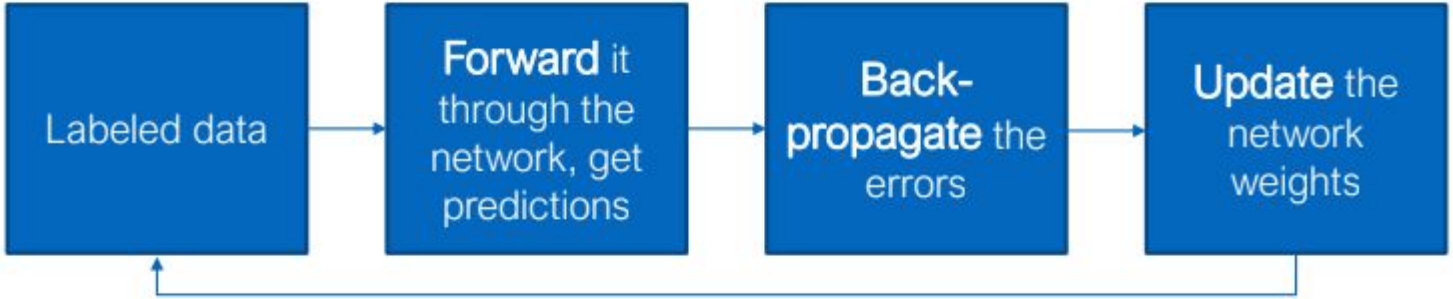




Let’s see how DNN work

Steps:

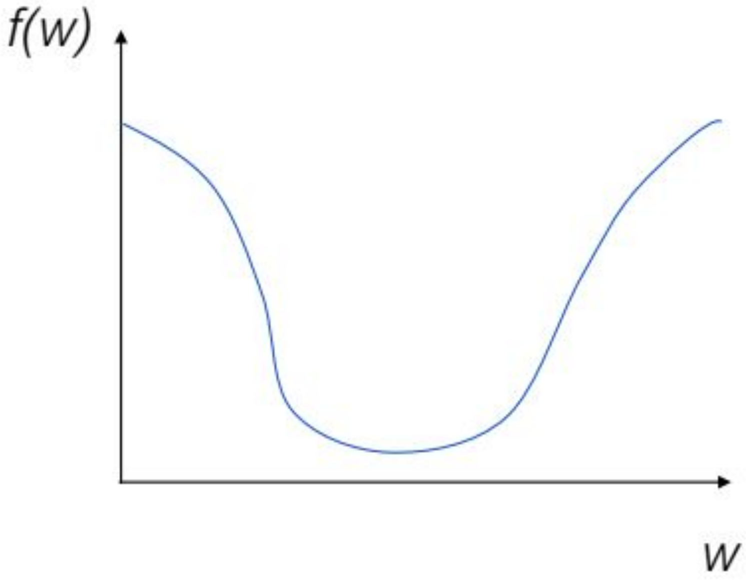
- Run forward pass and compute loss function
- Use loss function to compute learning change
- Perform weights update (Back propagation)
- Repeat


Labeled data
Forward it through the network, get predictions
Back-propagate the errors
Update the network weights

How the NN learns

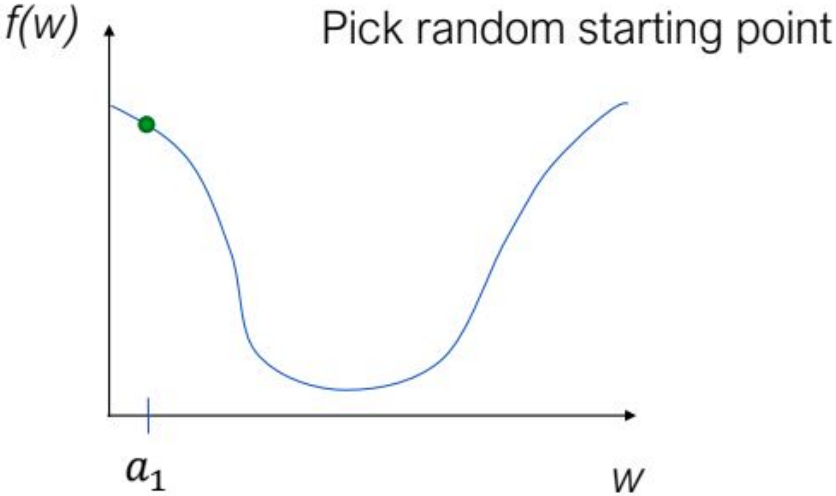
The hole learning process is about finding the minimum or local minimum in a function.

Let’s see how this process works



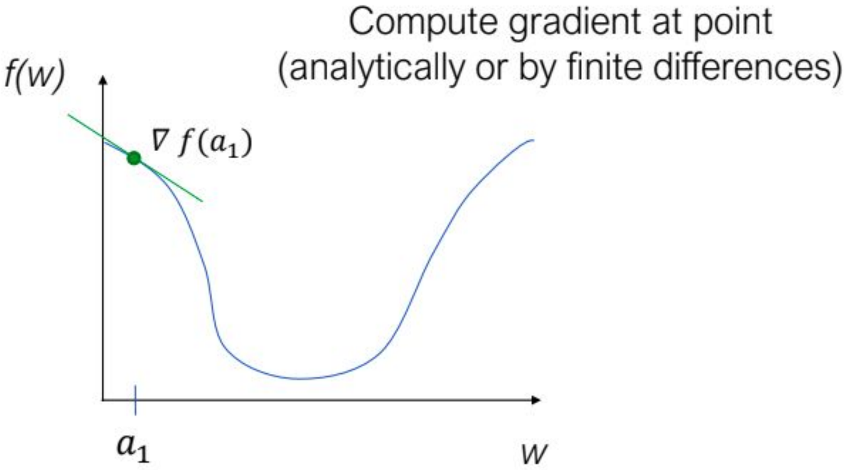
Gradient Descent - Step 1

This is a step that most network initialize by assigning a random values. It can be an issue in case the objective function as a lot of local minimums. Because this might light to very different results



Gradient Descent - Step 2

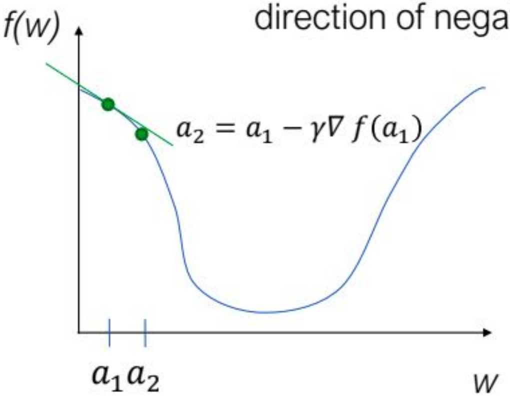
Compute the derivative of the function at that point. This is important because the derivative represents the slope of the function in that points. When the slope is zero, we know that we achieve the minimum.


Compute gradient at point
(analytically or by finite differences)
$f(w)$
$\nabla f(a_1)$
a_1
w

Gradient Descent - Step 3

Now that we know where to move, we need to know how much. This is done, by applying what is called learning rate. This is again a very important parameter because different values can lead to different results.

Move along parameter space in direction of negative gradient

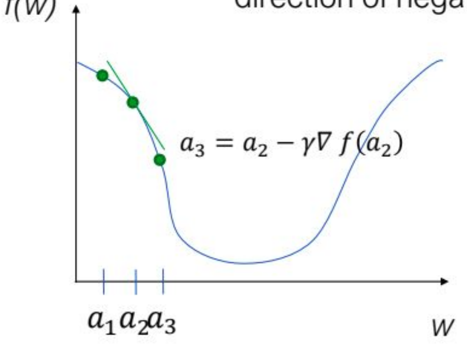


$$a_2 = a_1 - \gamma \nabla f(a_1)$$

γ = amount to move
= *learning rate*

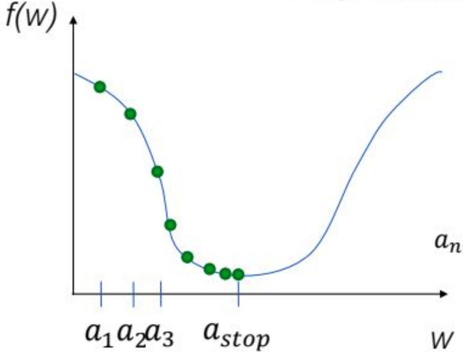
Gradient Descent - Step 3

Move along parameter space in direction of negative gradient



γ = amount to move
= *learning rate*

Stop when we don't move any more



a_{stop}:
$a_{n-1} - \gamma \nabla f(a_{n-1}) = 0$

Exercises

1. Plot a quadratic function between [-10, 10] with 0.1 of gap between the values, x*x
2. Compute derivative of the function
3. Apply gradient descent with lr (learning rate of 0.1)
4. Test with different values
5. Discuss the results

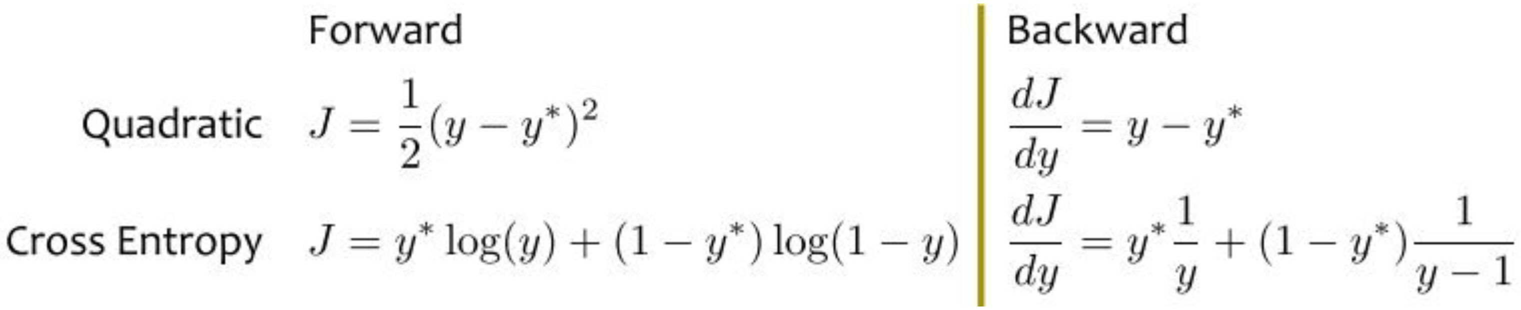
Loss Functions

Regression:

- Quadratic loss i.e. Mean Square Error

Classification

- Cross Entropy i.e. negative log likelihood



	Forward	Backward
Quadratic	$J = \frac{1}{2}(y - y^*)^2$	$\frac{dJ}{dy} = y - y^*$
Cross Entropy	$J = y^* \log(y) + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = y^* \frac{1}{y} + (1 - y^*) \frac{1}{y - 1}$

Backpropagation

Given: y = g(u) and u = h(x)

Given the chain rule of derivatives

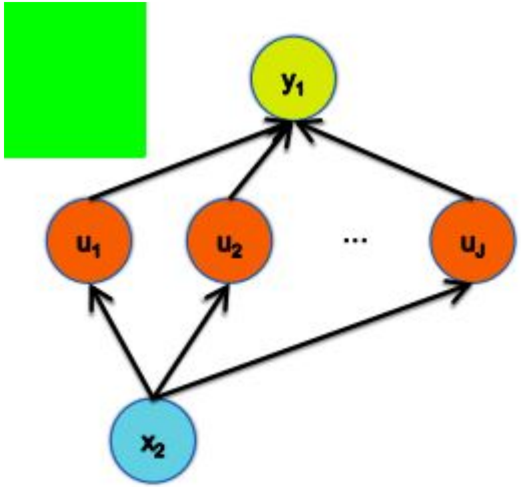
$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j}\frac{du_j}{dx_k}$$

Computing for each node, the result needs to be subtracted

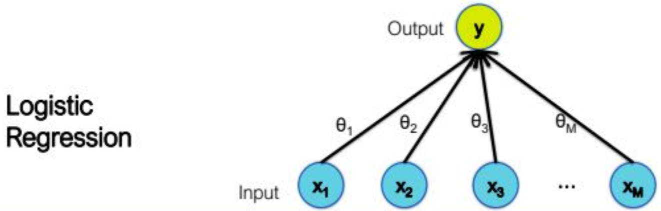
$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{d}{d\theta_j^{old}} J(\theta)$$

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

learning rate



Example with logistic regression



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^{D} \theta_j x_j$$

Backward

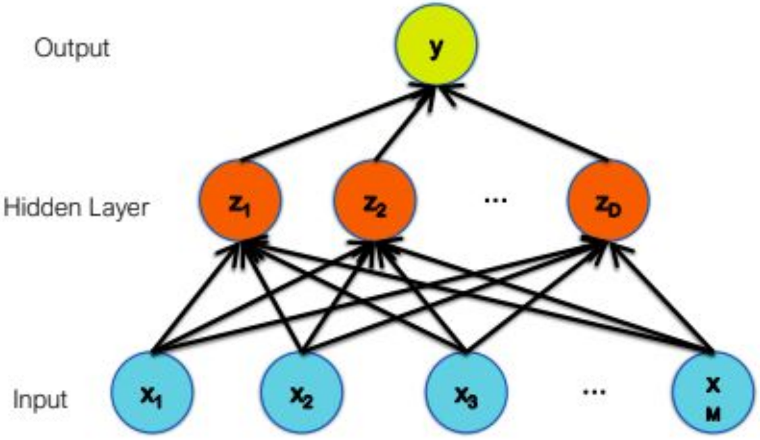
$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

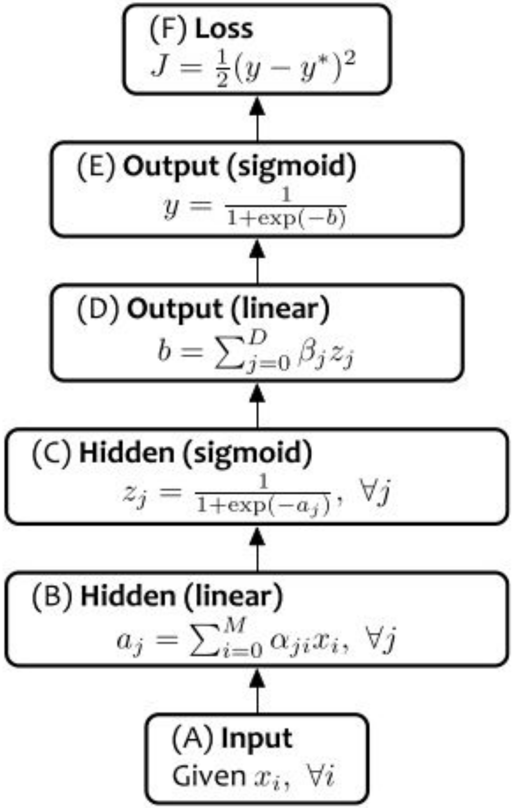
$$\frac{dJ}{da} = \frac{dJ}{dy}\frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

$$\frac{dJ}{d\theta_j} = \frac{dJ}{da}\frac{da}{d\theta_j}, \quad \frac{da}{d\theta_j} = x_j$$

$$\frac{dJ}{dx_j} = \frac{dJ}{da}\frac{da}{dx_j}, \quad \frac{da}{dx_j} = \theta_j$$

Example with NN





(F) **Loss**
$J = \frac{1}{2}(y - y^*)^2$

(E) **Output (sigmoid)**
$y = \frac{1}{1+\exp(-b)}$

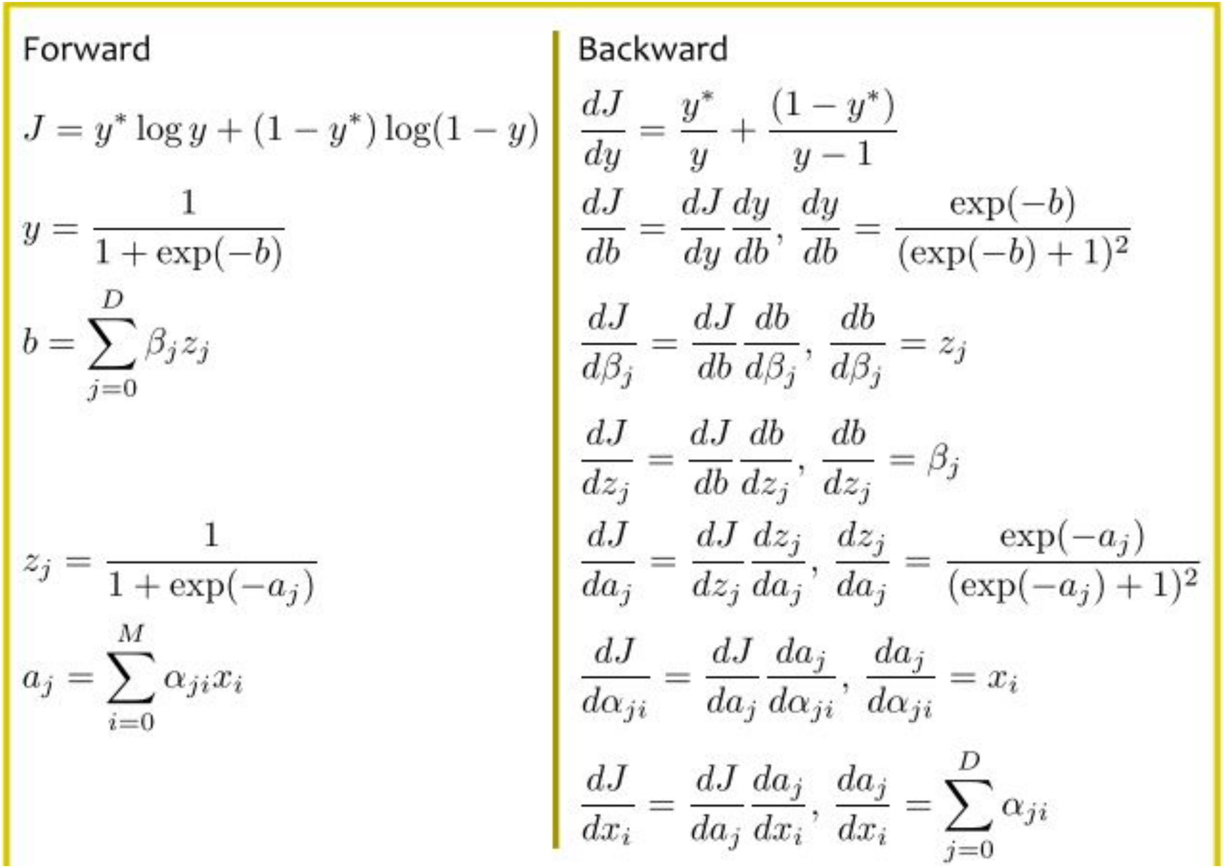
(D) **Output (linear)**
$b = \sum_{j=0}^{D} \beta_j z_j$

(C) **Hidden (sigmoid)**
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) **Input**
Given $x_i, \ \forall i$

Example with NN



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

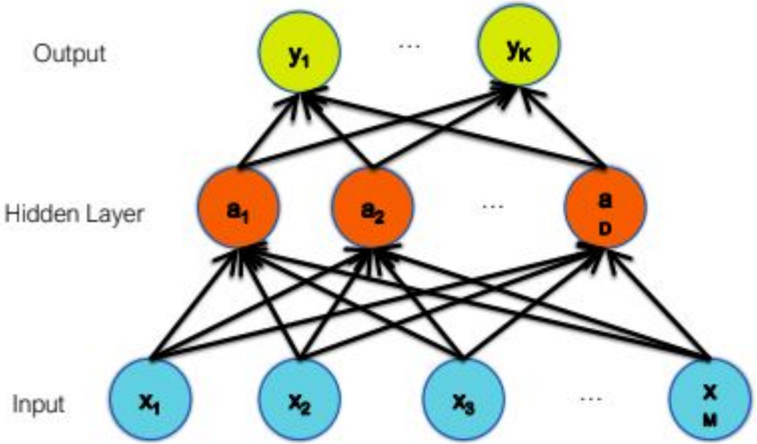
$$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

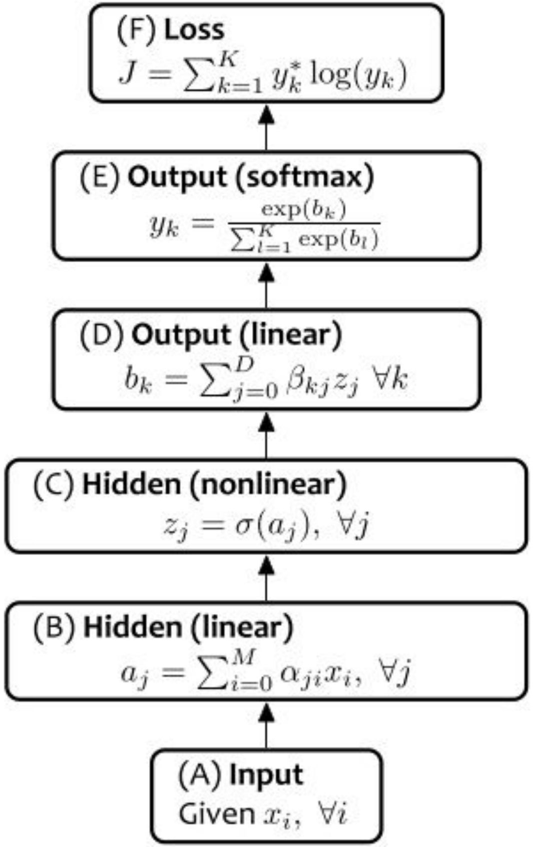
$$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji}$$

Example with Multiclass NN





(F) **Loss**
$J = \sum_{k=1}^{K} y_k^* \log(y_k)$

(E) **Output (softmax)**
$y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}$

(D) **Output (linear)**
$b_k = \sum_{j=0}^{D} \beta_{kj} z_j \;\; \forall k$

(C) **Hidden (nonlinear)**
$z_j = \sigma(a_j), \;\; \forall j$

(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \;\; \forall j$

(A) **Input**
Given $x_i, \;\; \forall i$

Many stuff to tune

As we saw NN have a wide range of parameter that can be set. All have huge impact in the outcome. THe most important are:

Architecture:

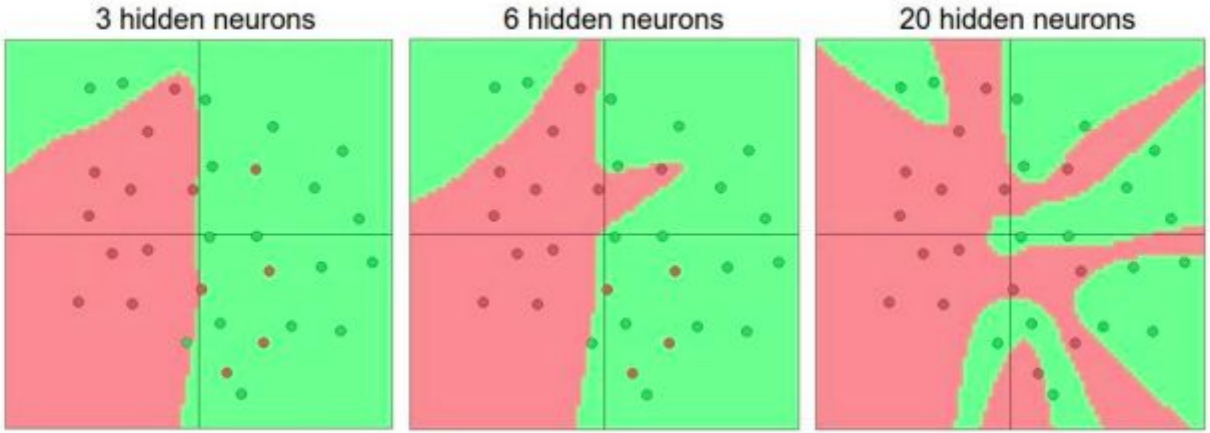
- Number of layers
- Unit for each layer
- Type of activation function
- Loss Function
- Learning rate
- Regularization
- Drop-out

There isn’t a solution for all problems so in each problem we need to think and iterate until reach the best possible set of parameters.

Activation functions

This are the functions that add the non-linearity to the Neural networks. Otherwise NN would only learn linear patterns.

Some examples



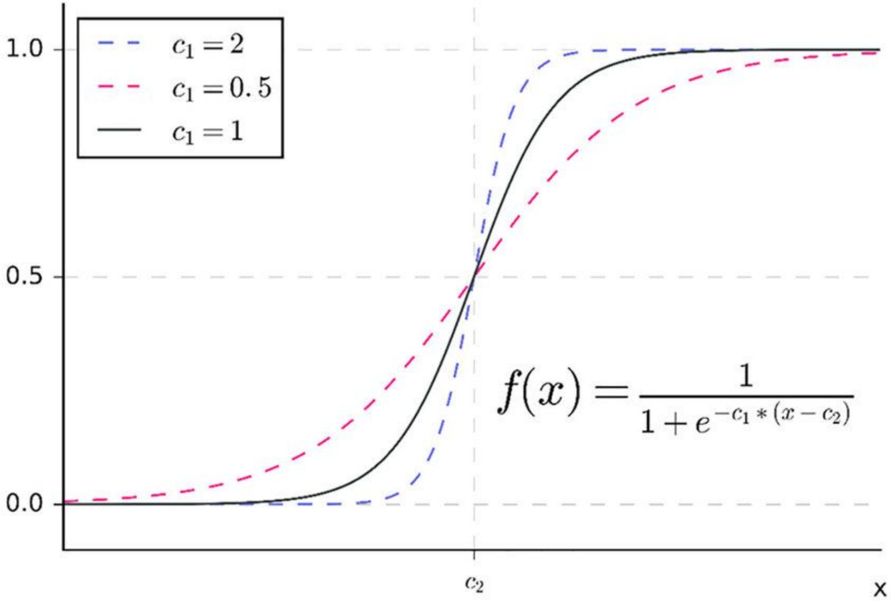
https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html

Sigmoid

Fits activation values between 0 and 1. Is always positive and increasing.

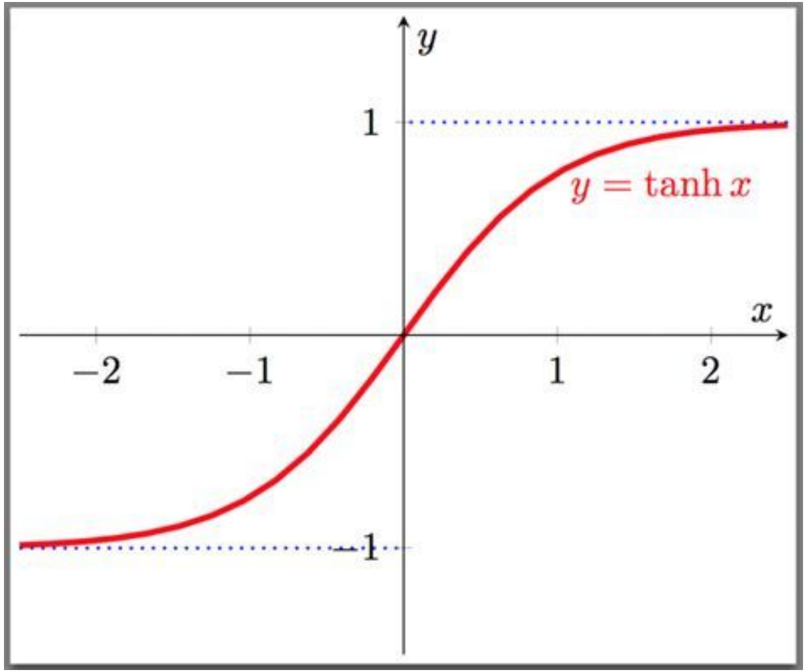
Properties:

- For big activation values the gradient is almost zero, which makes the function less sensitive
- This makes the nodes with that values to barely learn
- This happens for example in case of having huge initial weights, because may lease to saturation is a lot of nodes and slow the training.



Tanh

- Neurons also tend to saturate
- Unlike sigmoid, output is zero-centered which makes learning a bit faster
- Normally, tanh is preferred over sigmoid except for binary classification in the output layer

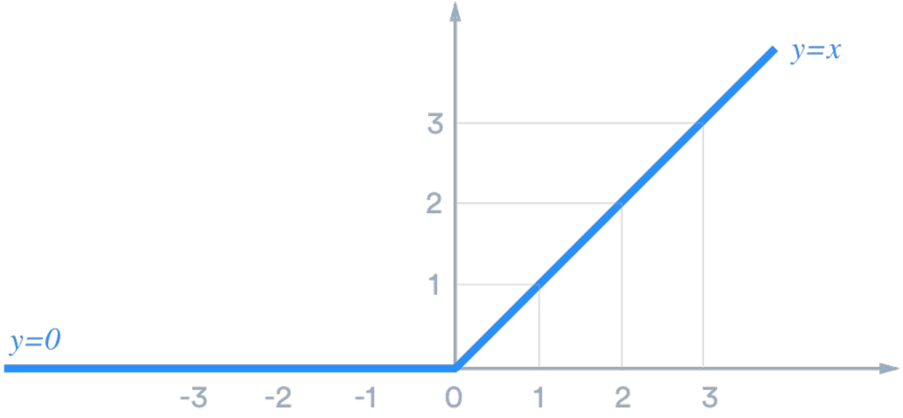


RELU (Rectified Linear Unit)

ReLu is commonly used in other layers rather than the output layer.

Properties:

- Trains much faster
- Accelerates the convergence of the SGD due to linear and not saturating form
- Less expensive operations due to is linear nature
- Prevents the vanishing gradient problem
- When z is 0, gradient is zero, so de model do not learn



Parameter initialization

For bias:

- Initialize all to zero

For weights:

- Can’t be all equal because otherwise the weights will learn the exact the same.
- We need to break symmetry. So random model can be applied
- Special initialization algorithms:
    - Xavier initialization for Sigmoid or Tanh
    - He initialization for ReLu

Exploding/Vanishing gradients

Exploding gradients

Exploding gradients are a problem where large error gradients accumulate and result in very large updates to neural network model weights during training.

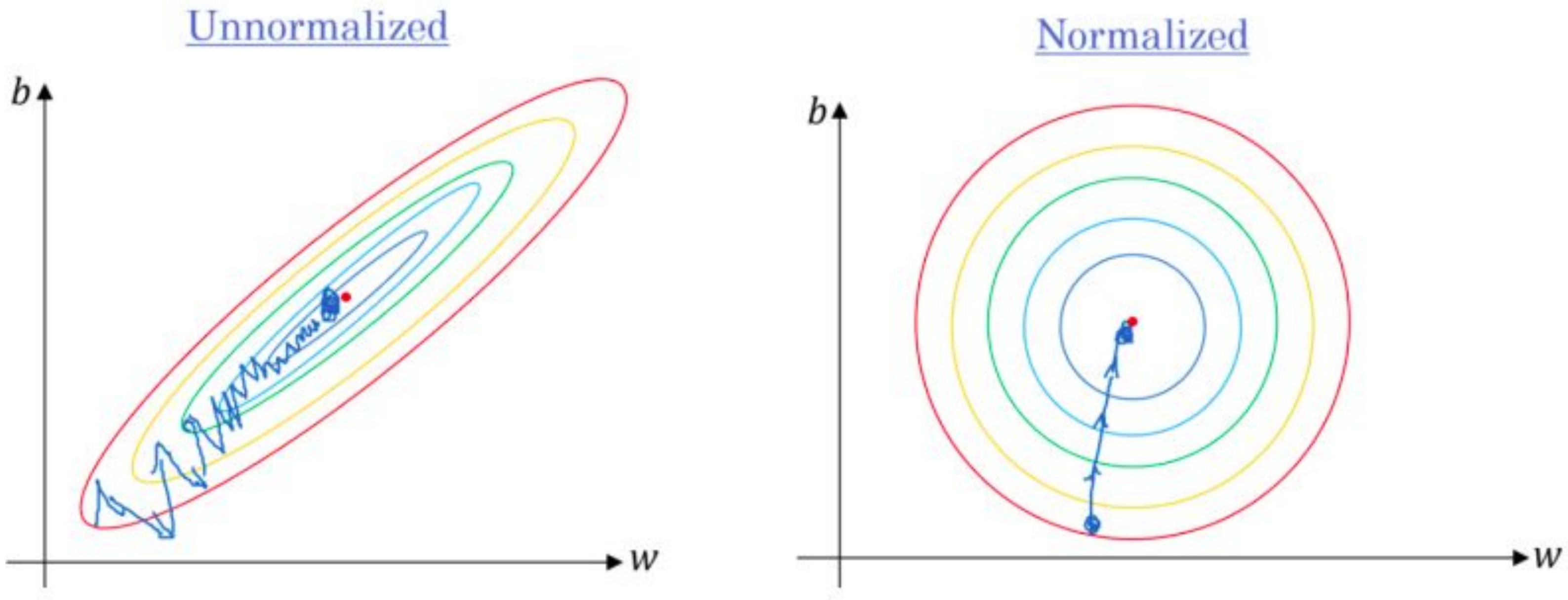
This has the effect of your model being unstable and unable to learn from your training data.

Vanishing gradients

This most often occurs in neural networks that have several neuronal layers such as in a deep learning system, but also occurs in recurrent neural networks. The key point is that the calculated partial derivatives used to compute the gradient as one goes deeper into the network.

Since the gradients control how much the network learns during training, if the gradients are very small or zero, then little to no training can take place, leading to poor predictive performance.

Feature Normalization



More parameters

Hyperparameters: Parameters that influence training:

- Learning rate
- Epoch number
- Number of hidden layers
- Hidden units for each layer
- Activation function
- MOmentum, mini-batch size, regularation

Bias - Variance Trade Off

In the deep learning work this problem still exists, however is less problematic because this kind of models allow more independent tuning.

High variance: -> Overfitting

- Train longer
- Different NN architecture search

High bias:

- Get more data
- Regularization
- Different NN architecture search

Overfitting

Again, this is also an issue. To solve we can:

- Add layer dropout - this basically consists on dropping nodes connection while training in order to reduce the model complexity
- Each node with retained probability p. An example is L2 regularization
- Regularization term that penalizes big weights - added in the loss function
- Add early stop in order - this simply means that after a certain value of loss we stop training
- Improve data quality - for example using data augmentation

Batch vs mini-batch Gradient Descent

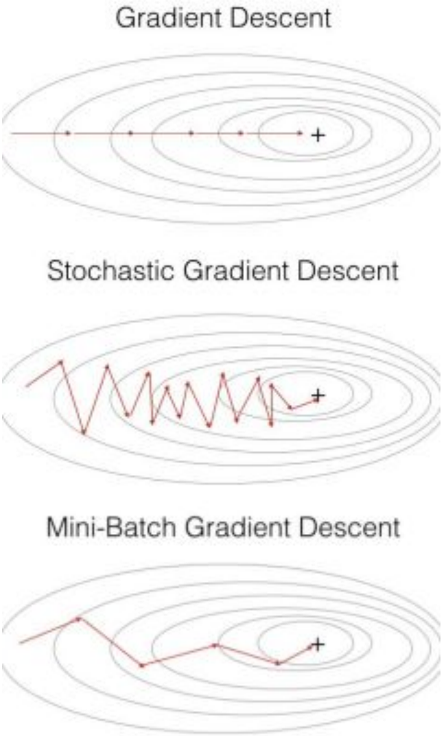
In mini-batch, gradient descent starts making progress before the hole dataset does the forward pass. Example:

If our dataset has size m:

Our batch are for example m/5. This means that we pass our inputs in groups of 5, average the results and update the weights. After that an epoch is completed.

In a mini-batch approach, each group is computed and the networks weights are updated.

The advantages of the bigger batches is the speed of training, however is more computation expensive because requires bigger memory.



Gradient Descent

Stochastic Gradient Descent

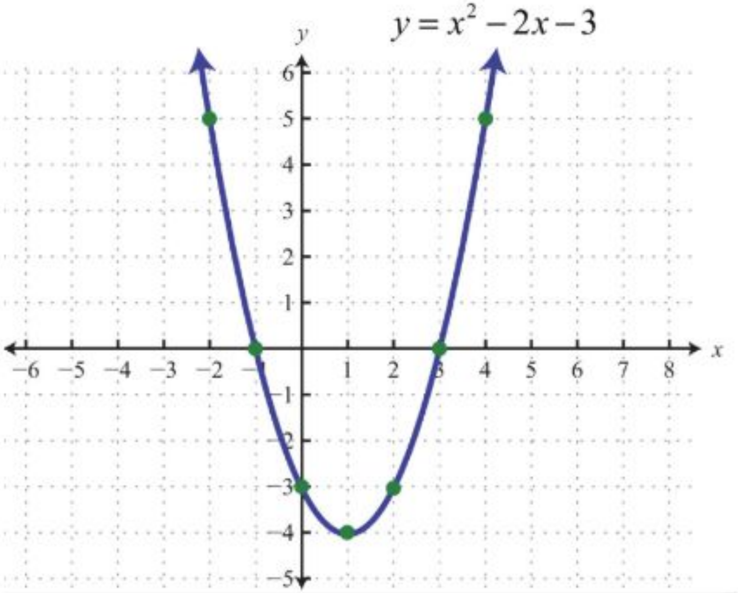
Mini-Batch Gradient Descent

Stochastic gradient descent

Most algorithms that have stochastic in the name, means that at least some part of it is random.

In this algorithm, the random part is used on the selection of the samples used to perform the algorithms.

In the example we did only one data point, but in reality we have many many more. Let's imagine we have 10 000 samples and 10 features , this means we need to compute the loss for all the 10 000 samples x 10 features. In order to be able to update the weights. In a stochastic approach we select a random number of samples is order to do the computation.



What other alternatives exist?

Until now we say learning algorithms with fixed learning rate. But there are alternatives that update the learning rate while training. For example image that the training is going towards a clear direction, the learning rate is changed in order to follow that path quicker. As example is the **Gradient Descent with Momentum.**

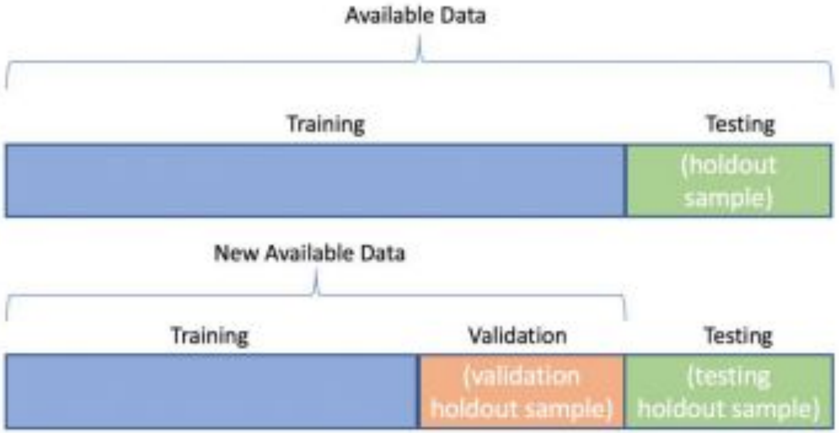
Another possibility is the **RMSprop (Root mean square)** that ensure that once the algorithm gets a clear direction that does not changes.

A joint of this two approaches lead to the **Adam algorithm** (Adaptive moment estimation).

Evaluation techniques

In deep learning world we must have huge amounts of data, otherwise, classic methods are preferred. So on those cases we can have different training/test/validation datasets

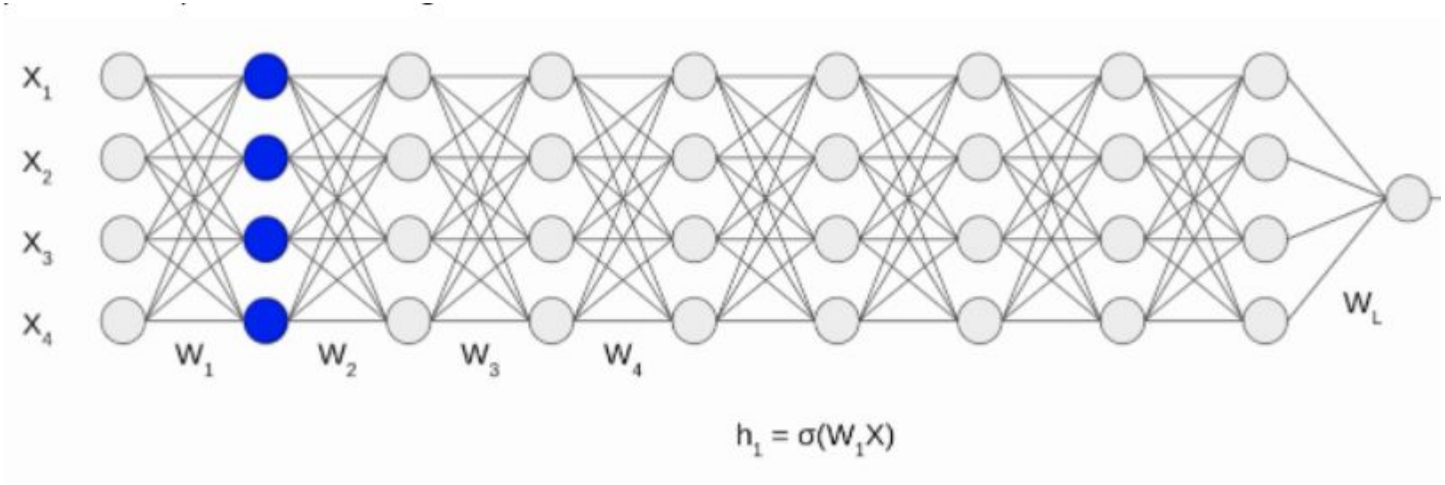
- 98% / 1% / 1%



Batch normalization

Helps to make hyper-parameter tuning easier. Also makes the train faster

Consists on normalizing the input after the first layers, and this is done in batches.



2015 original paper https://arxiv.org/abs/1502.03167

DL Frameworks

- TensorFlow (Google’s open source) https://www.tensorflow.org/
- Pytorch (Facebook’s open-source) https://pytorch.org/
- Keras https://keras.io/
- Theano https://pypi.org/project/Theano/

Tensorflow

In tensorflow the base concept is the tensor object. Is is very similar to the np.array object that represent high dimensional matrices.

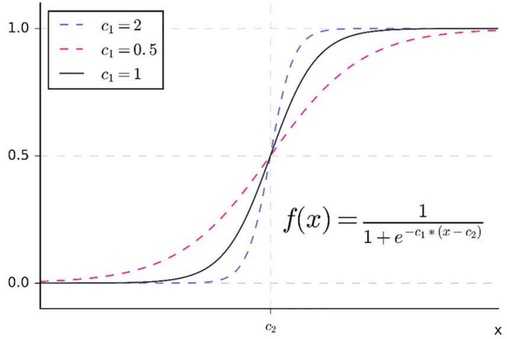
There are other form of tensors like the tf.Constant that are immutable

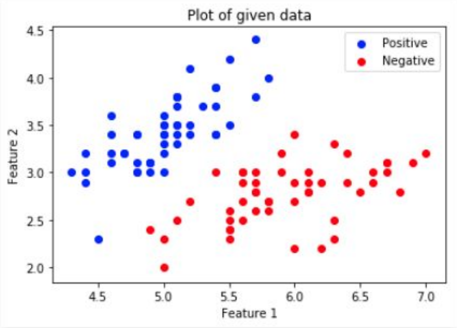
Let’s create a virtualenv and install tensorflow == 2.4.3

Logistic regression

Logistic Regression is Classification algorithm commonly used in Machine Learning. It allows categorizing data into discrete classes by learning the relationship from a given set of labeled data. It learns a linear relationship from the given dataset and then introduces a non-linearity in the form of the Sigmoid function.

https://www.youtube.com/watch?v=yIYKR4sgzI8&ab_channel=StatQuestwithJoshStarmer





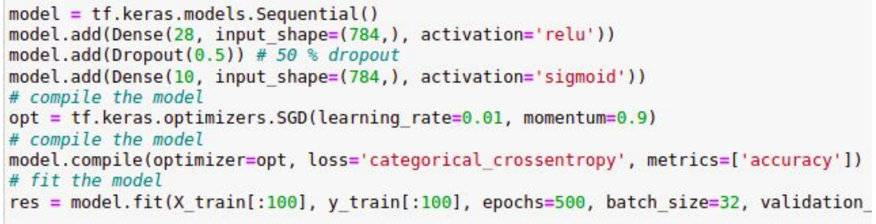
Definitions: Epoch

An epoch is defined when all data passed through the network in the forward pass and the backward pass.

In terms of artificial neural networks, an epoch refers to one cycle through the full training dataset.

More details:

https://deepai.org/machine-learning-glossary-and-terms/epoch



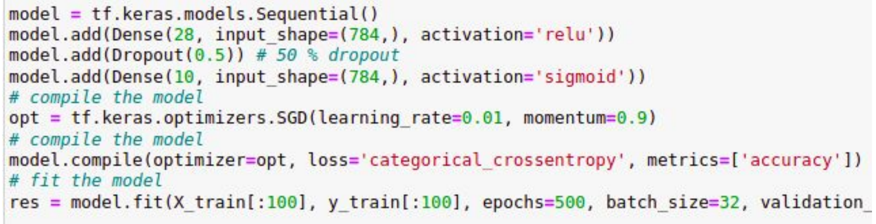
```
model = tf.keras.models.Sequential()
model.add(Dense(28, input_shape=(784,), activation='relu'))
model.add(Dropout(0.5)) # 50 % dropout
model.add(Dense(10, input_shape=(784,), activation='sigmoid'))
# compile the model
opt = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
# compile the model
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
# fit the model
res = model.fit(X_train[:100], y_train[:100], epochs=500, batch_size=32, validation_
```

Definitions: batch_size

Let's say we have 1000 images of dogs that we want to train our network on in order to identify different breeds of dogs. Now, let's say we specify our batch size to be 10. This means that 10 images of dogs will be passed as a group, or as a batch, at one time to the network.

Well, for one, generally the larger the batch size, the quicker our model will complete each epoch during training. This is because, depending on our computational resources, our machine may be able to process much more than one single sample at a time.

The trade-off, however, is that even if our machine can handle very large batches, the quality of the model may degrade as we set our batch larger and may ultimately cause the model to be unable to generalize well on data it hasn't seen before.



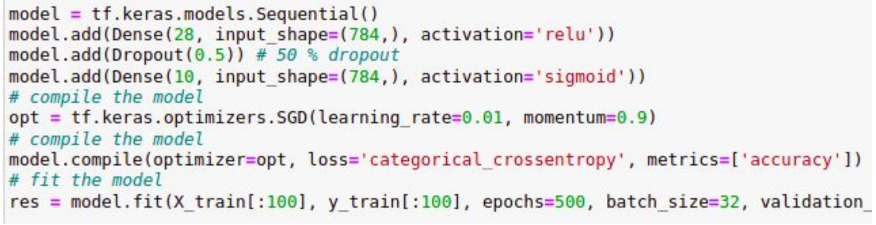
```
model = tf.keras.models.Sequential()
model.add(Dense(28, input_shape=(784,), activation='relu'))
model.add(Dropout(0.5)) # 50 % dropout
model.add(Dense(10, input_shape=(784,), activation='sigmoid'))
# compile the model
opt = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
# compile the model
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
# fit the model
res = model.fit(X_train[:100], y_train[:100], epochs=500, batch_size=32, validation_
```

More details: https://deeplizard.com/learn/video/U4WB9p6ODjM

Definitions: lr i.e learning rate

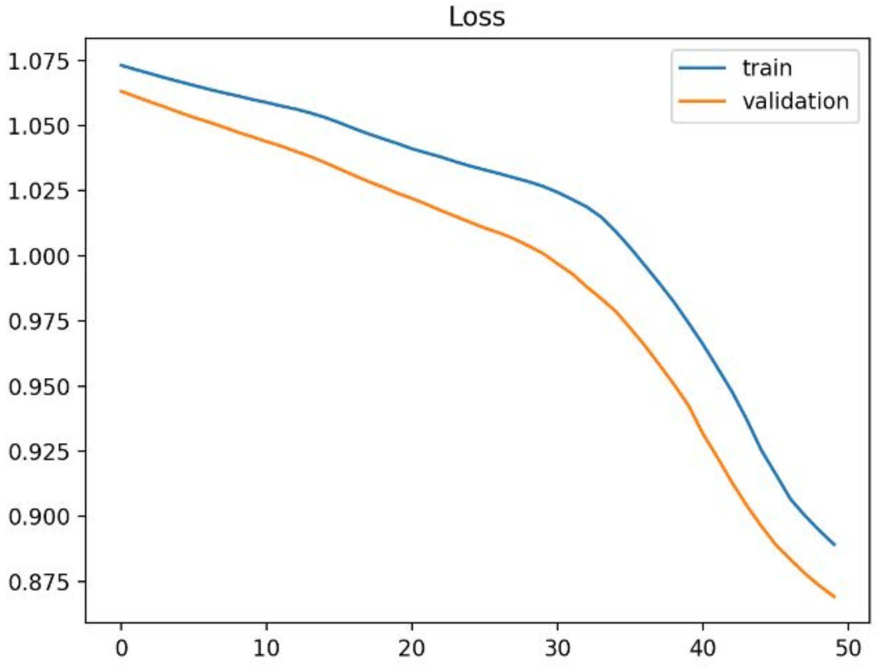
Learning rate defines how big are the actualization of the network weights.

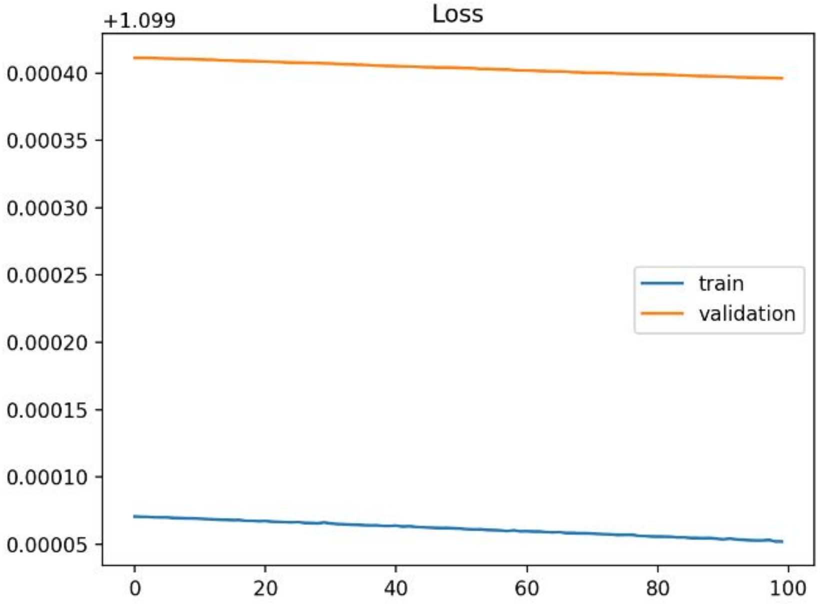
Usually the bigger it is, the bigger is the risk of achieving local minimas.



```
model = tf.keras.models.Sequential()
model.add(Dense(28, input_shape=(784,), activation='relu'))
model.add(Dropout(0.5)) # 50 % dropout
model.add(Dense(10, input_shape=(784,), activation='sigmoid'))
# compile the model
opt = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
# compile the model
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
# fit the model
res = model.fit(X_train[:100], y_train[:100], epochs=500, batch_size=32, validation_
```

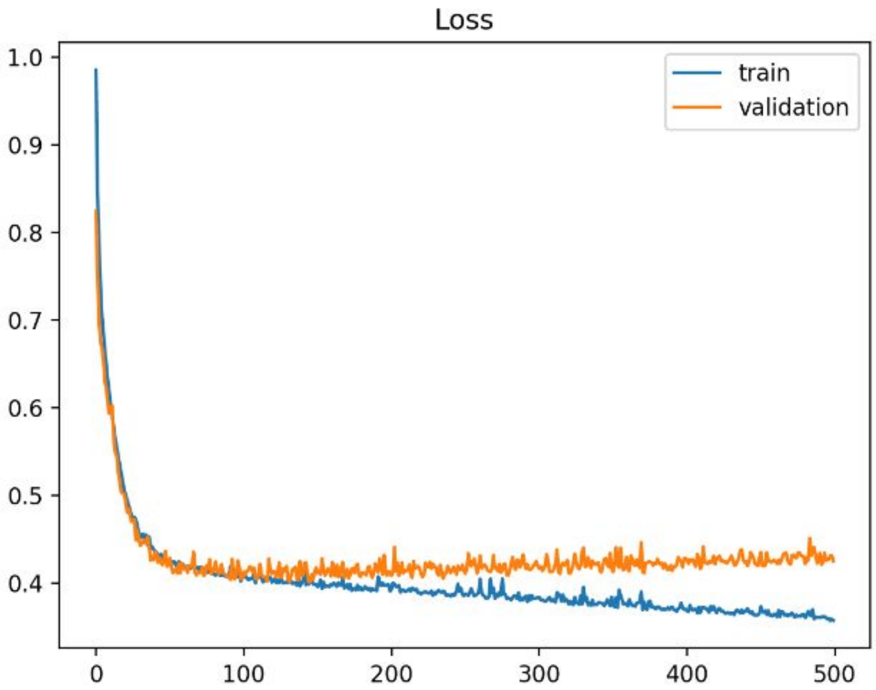
Definitions: learning curve - underfitting



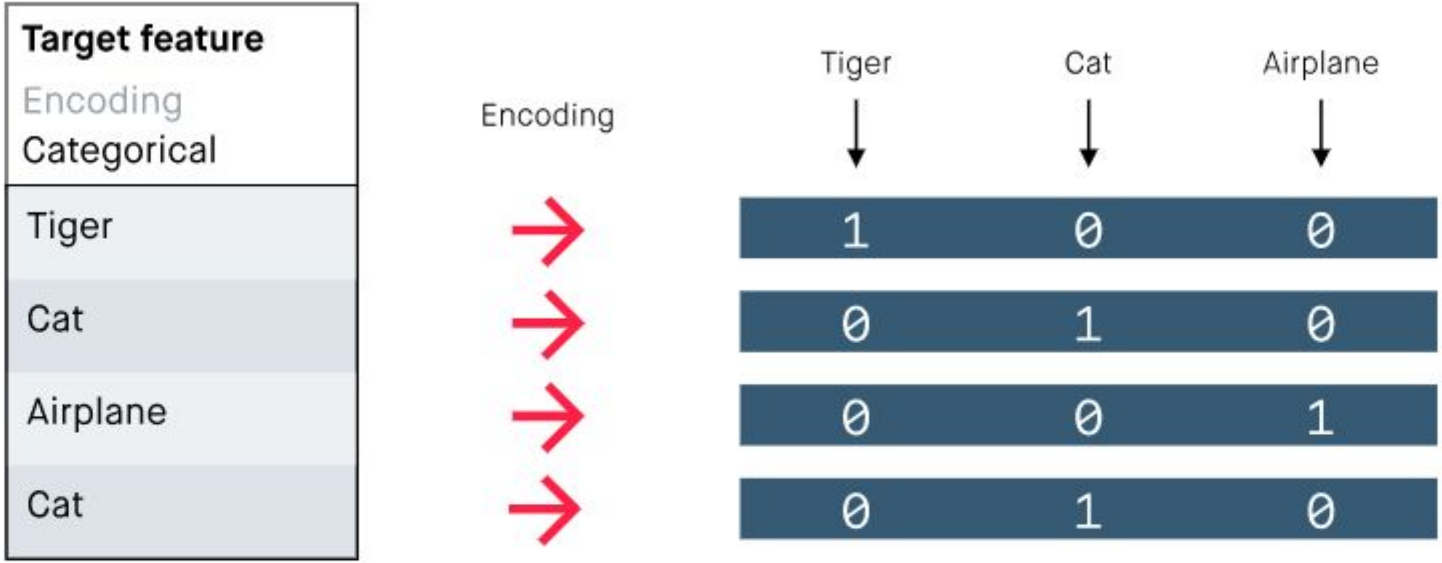


Tiago Cabo

Definitions: learning curve - overfitting



One-hot encoding



Target feature Encoding Categorical
Tiger
Cat
Airplane
Cat

Tiger	Cat	Airplane
1	0	0
0	1	0
0	0	1
0	1	0

Let’s build our first NN

1. Load with from sklearn.datasets import load_digits
2. Perform quick EDA
3. Plot one example of each different image with plt.imshow()
4. Let’s do the same with from tensorflow.keras.datasets import mnist
5. Let’s perform a min-max scaler
6. Perform one hot encoding using tf.one_hot(data, depth=n)
7. Please fill the input_shape and the number of nodes in the layer present in the notebook.
8. Evaluate the model and check accuracy. What do you think?
9. Predict for test data and check some results
10. Plot learning curve using model.fit results
11. Compute model.summary()
12. Improve model by adding dropout by doing
13. Training With Batch Normalization
14. Halt Training at the Right Time With Early Stopping
15. Try adam
16. Try a bigger batch size
17. Try more data
18. Create a more complex model