Artificial Neural Networks

Aprendizaje Automático para la Robótica
Máster Universitario en Ingeniería Industrial

Departamento de Automática

Objectives

1. Describe biological neurons and networks
2. Describe artifical neurons
3. Introduce the MLP
4. Understand the role of trainning in ANNs
5. Understand MLP hyperparameters

Bibliography

- Géron, Aurélien. *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow*. 2nd Edition. O'Reilly. 2019

Table of Contents

Introduction

History

1888 Ramón y Cajal. Discovery of biological neurons

1943 McCulloch & Pitts. First neural network designers

1949 Hebb. First learning rule

1958 Rosenblatt. Perceptron

1969 Minsky & Papert. Perceptron limitation - Death of ANN

1986 Rumelhart *et al.* Re-emergence of ANN: Backpropagation

2012 CNNs popularity - AlexNet - Rise of Deep Learning

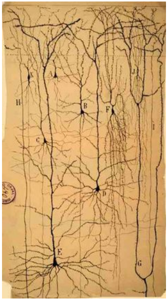
2014 Goodfellow *et al.* Generative Adversarial Networks (GANs)

2021 OpenAI. Dall-e 2

2022 Large Language Models (ChatGPT, GPT-3, new Bing)

2023 Multimodal Large Language Models (GPT-4)

20xx ... AGI?



Introduction

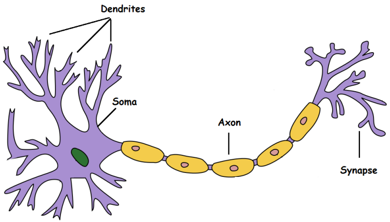
Biological neurons (I)

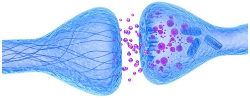
A neuron has a cell body (soma) ...

- ... a branching *input* structure (dendrite) and
- ... a branching *output* structure (axon)
- ... an axon termination named *synapses*

An *action potential* (or signal) may propagate from dentrites to synapses

- The synapses release a chemical named *neurotransmitter*
- Given enougth neurotransmitters, a new neuron can fire





Introduction

Biological neurons (II)

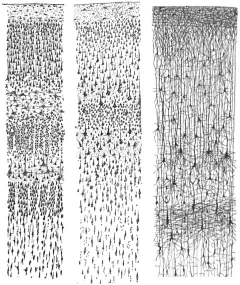
A neuron only fires if its input signal exceeds a threshold

- Good connections allowing a large signal
- Slight connections allowing a weak signal
- Synapses may be either excitatory or inhibitory

Synapses vary in strength

- Biological learning involves setting that strength

Biological neurons often are organized in layers

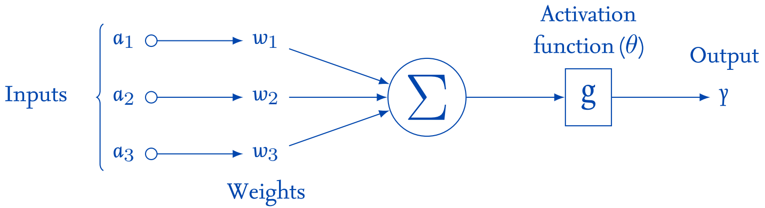


(Source)

Artificial neurons

Definition (I)

TLU: Threshold logic unit



- a_i Input
- w_i Weight of input j
- θ Threshold
- g Activation function

Neuron model (TLU)

$$y = g\left(\sum_i w_i a_i\right)$$

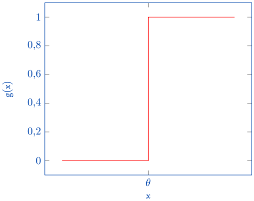
Artificial neurons

Definition (II)

The idealized activation function is a step function

$$g(x) = \begin{cases} 1 & \text{if } \sum_{i=1}^{N} w_i x_i > \theta \\ 0 & \text{otherwise} \end{cases}$$

The step function is rarely used in practice



Artificial neurons

Definition (III)

A single neuron can be used for *linear binary classification*

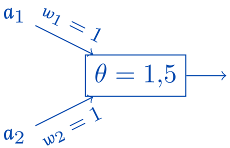
- Computes linear combination of inputs
- If the output exceeds the threshold, it assigns a positive class
- ... othershise it assigns a negative class

Comparable to logistic regression or SVM

Artificial neurons

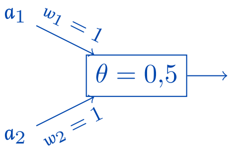
Logical gates with a neuron

AND



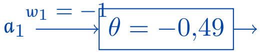
a_1	a_2	y
0	0	0
0	1	0
1	0	0
1	1	1

OR



a_1	a_2	y
0	0	0
0	1	1
1	0	1
1	1	1

NOT

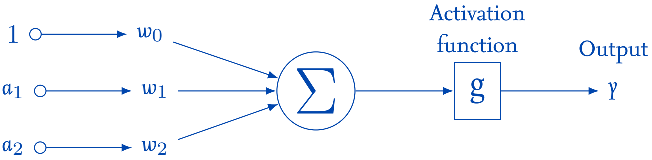


a_1	y
0	1
1	0

(A neuron in Excel)

Artificial neurons

Definition of neuron (alternative version)



a_i Input

w_i Weight of input j

w_0 Bias

g Activation function

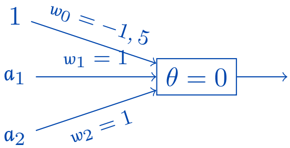
Neuron model

$$y = g\left(\sum_i w_i a_i\right)$$

Artificial neurons

Example of biased neuron

AND logical gate with a biased input



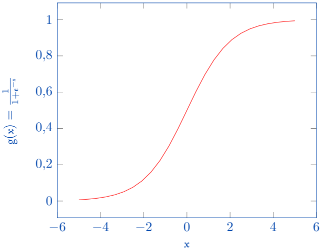
a_0	a_1	a_2	Output
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Artificial neurons

Activation functions: Sigmoid function

Also known as the logistic function

- Biological motivation
- S-shaped, continuous and everywhere differentiable
- Asymptotically approach saturation points
- Derivative fast computation
- Range $\in [0, 1]$



Sigmoid function

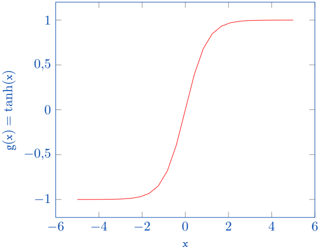
$$g(x) = \frac{1}{1 + e^{-x}}$$

$$g'(x) = g(x)(1 - g(x))$$

Artificial neurons

Activation functions: Tanh function

- Asymptotically approach saturation points
- Range $\in [-1, 1]$
- Bigger derivative than sigmoid (faster training)



Tanh function

$$g(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

$$g'(x) = 1 - g(x)^2$$

Artificial neurons

Activation functions: Softmax function

- Generalization of the logistic function
- Usually used in the output layer in classification problems
- Asymptotically approach saturation points

Softmax function

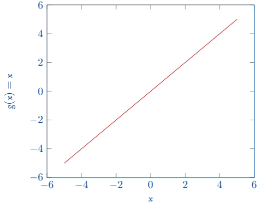
$$g(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } j = 1, \ldots, K$$

with $\mathbf{z}$ a K-dimensional vector

Artificial neurons

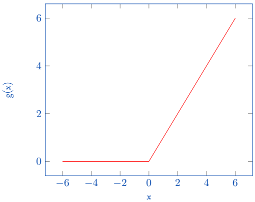
Other activation functions

Linear function



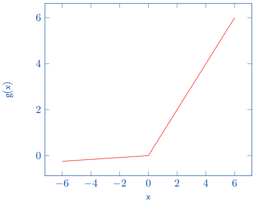
- Used in regression
- Last layer in regression

Rectified Linear (ReLU)



- Faster derivate
- Popular in DL

Leaky ReLU



- More informative derivate
- Popular in DL

The lack of non-linear activation function makes a network a simple linear regression

Artificial neurons

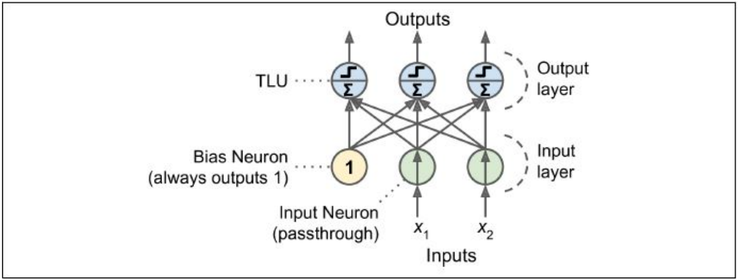
The Perceptron

The Perceptron is a very simple ANN architecture

- Disclaimer: different defintions of Perceptron
- Proposed by Rosenblatt in 1957

Perceptron: Layer of TLUs connected to all the inputs

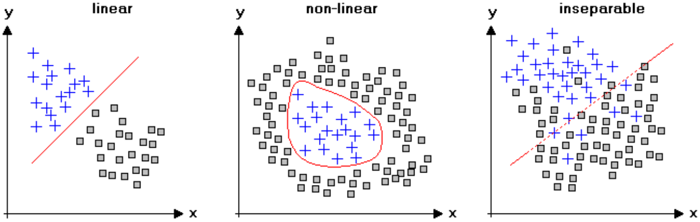
- *Input layer* contains special passthrough neurons
- Multilabel classification



(Source)

Artificial neurons

Learning limits (I)

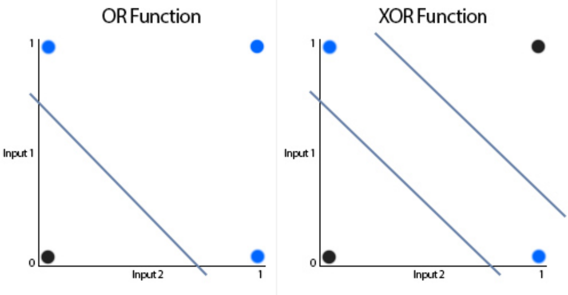


Problem: A single perceptron only can solve linearly separable problems

Artificial neurons

Learning limits (II)

XOR cannot be implemented with a perceptron



Solution: Stack several perceptrons

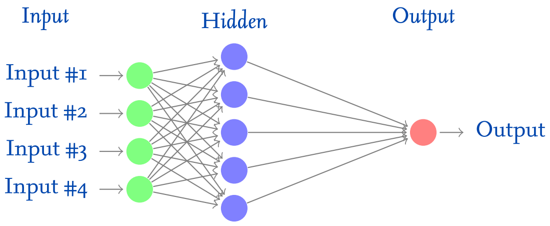
The Multilayer Perceptron

Definition (I)

Neurons are arranged in layers of TLU neurons

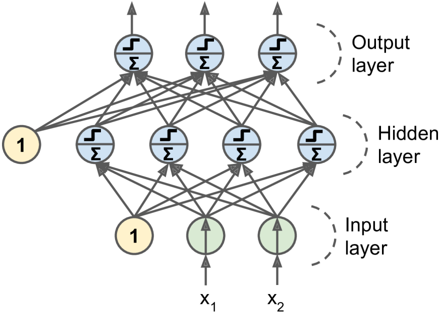
- *Input* Which consists of our data
- *Output* Which are the net outcome
- *Hidden* (Optional) No direct interaction

Multilayer Perceptron, or simply MLP



The Multilayer Perceptron

Definition (II)

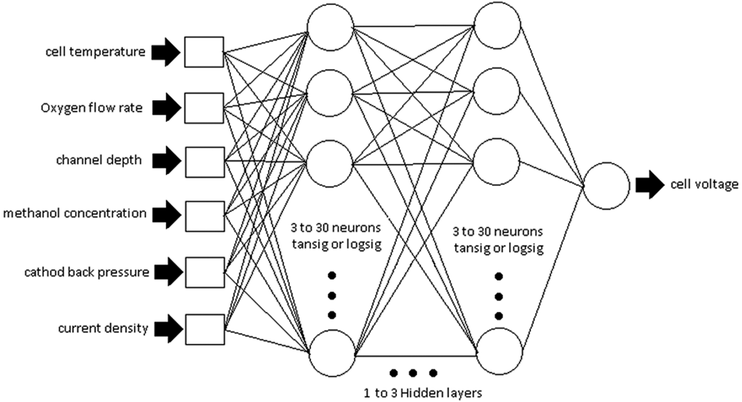


(Source)

The Multilayer Perceptron

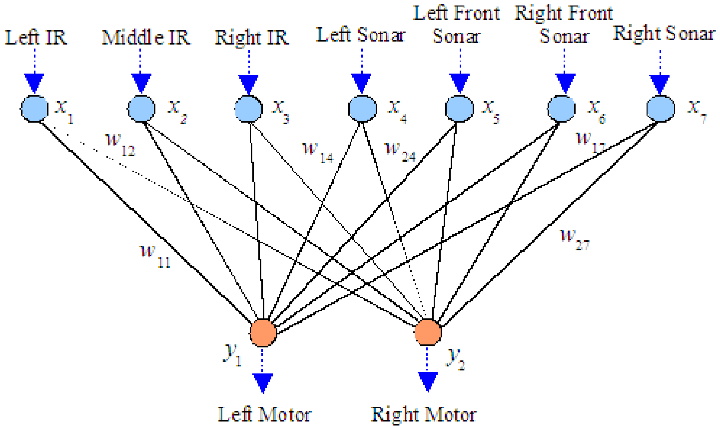
MLP for regression (I)

One output neuron for each output dimension



The Multilayer Perceptron

MLP for regression (II)



The Multilayer Perceptron

MLP for regression (III)

You usually do not want to limit the output

- Output layer with linear activation
- ReLu allowed for strictly positive output

Loss function as in any regression

- Mean Squared Error (MSE) by default
- Mean Absolute Error (MAE) is less sensitive to outlayers

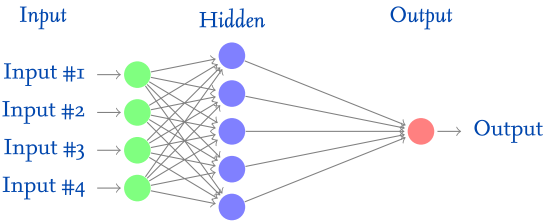
A loss function is an error function used to train / evaluate a network

The Multilayer Perceptron

MLP for classification (I)

Binary classification

- One output neuron
- Sigmoid activation
- The output can be interpreted as a positive class probability

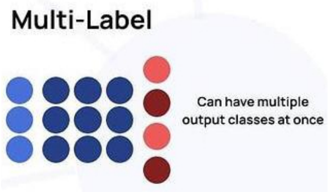


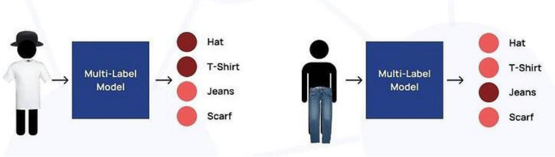
The Multilayer Perceptron

MLP for classification (II)

Multilabel classification

- One output neuron per label
- Labels are not mutually exclusive
- Sigmoid activation





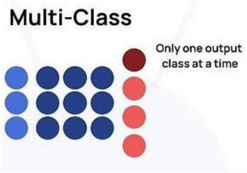
(Source)

The Multilayer Perceptron

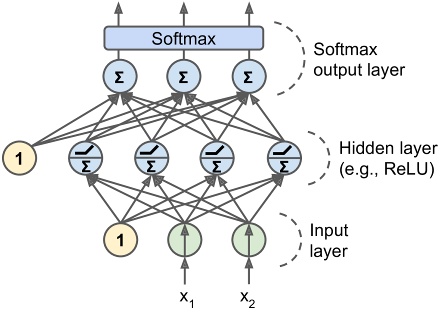
MLP for classification (III)

Multi-class classification

- Mutually exclusive label ⇒ Probabilities are not independent
- One output neuron per label
- Softmax activation



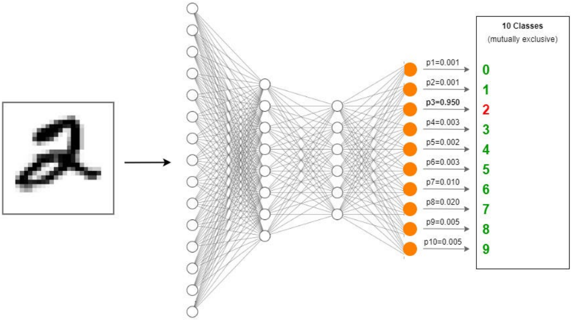
(Source)



(Source)

The Multilayer Perceptron

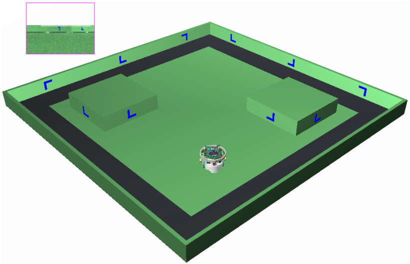
MLP for classification (IV): Example 1

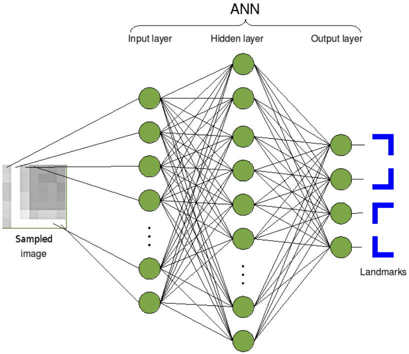


(Source)

The Multilayer Perceptron

MLP for classification (IV): Example 2

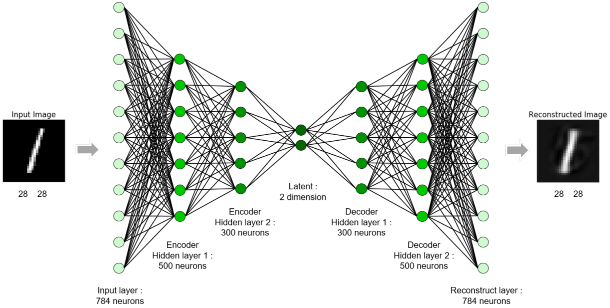


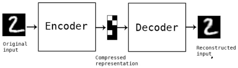


(Source)

The Multilayer Perceptron

Autoencoders





(Source)

The Multilayer Perceptron

Demo

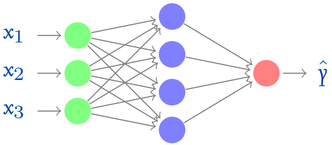
(Online demo)

Gradient Descent

Motivation (I)

In supervised learning we have the target outputs ...

- ... so we can compare them with the observed one



x_1	x_2	x_3	$\hat{y}$	y
1,1	2,5	4,5	0,2	−0,1
0,9	2,4	1,2	0,5	0,4
1,0	2,0	9,9	0,4	1,2

Loss function (*función de pérdida*): Measure of the error

- Any error measure can be used, there are many available
- Usually MSE or MAE (among others ...)

$$\text{MSE} = \frac{1}{n}\sum_i (y_i - \hat{y}_i)^2 \Rightarrow \text{MSE} = f(\vec{\theta})$$

where $\vec{\theta}$ is our network (its weights and biases)

Gradient Descent

Motivation (II)

Problem: Determine $\vec{\theta}$ that minimizes $f(\vec{\theta})$

- This is a classical optimization problem
- Any optimization algorithm can be used
- ... in AI, optimization means search

In DL, our network may have millions of parameters

- We do know analytically $f(\vec{\theta})$
- Optimization based on gradients: Gradient Descent

Gradient Descent is a general optimization algorithm

- Not limited to train neural networks
- Widely used, for instance, to fit lineal models

Gradient Descent

Overview

(Source)

Gradient Descent

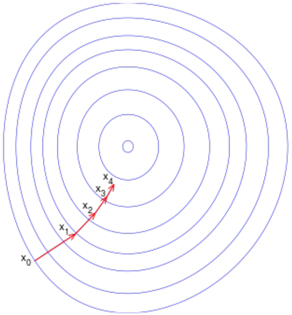
Gradient Descent (I)

Calculate the gradient of the loss function with respect weights

- Adjust weights along gradient direction
- Gradient provides the direction
- η is the learning rate

Gradient descent

> $\vec{\theta} \leftarrow random()$
> **while** Not converged **do**
>   **for all** $\theta_i \in \vec{\theta}$ **do**
> $\theta_i \leftarrow \theta_i - \eta \frac{\partial}{\partial \theta_i} f(\vec{\theta})$
>   **end for**
> **end while**

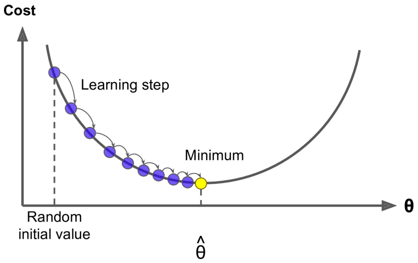


Gradient Descent

Gradient Descent (II)

Alternative notation:

$$\vec{\theta} = \vec{\theta} - \eta \nabla_{\theta} f(\vec{\theta})$$



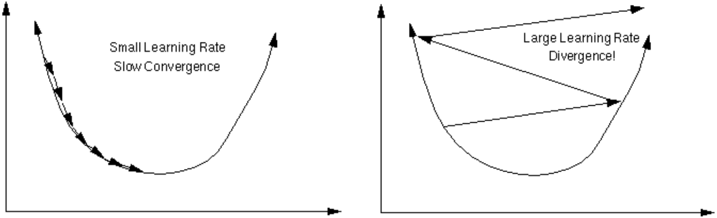
(Source)

Gradient Descent

Learning rate

The learning rate is a an important hyperparameter

- Small learning rate ⇒ Slow convergence
- Large learning rate ⇒ Jump across the valley

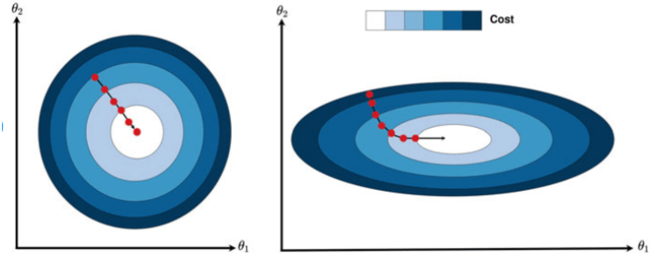


(Source)

Gradient Descent

Gradient Descent problems (I)

GD is sensitive to features scaling



(Source)

Gradient Descent

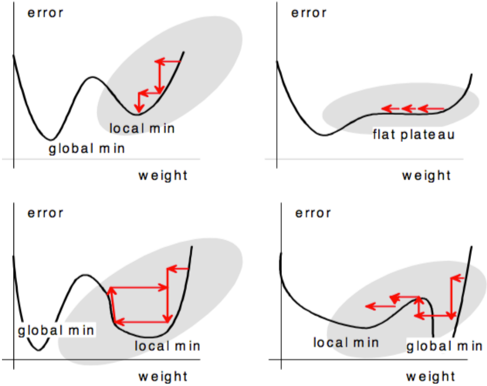
Gradient Descent problems (II)

Potential problems

- Local minima
- Flat plateau
- Oscillation
- Missing good minima

GD uses the whole dataset

- It is slow ...
- ... inviable in practice



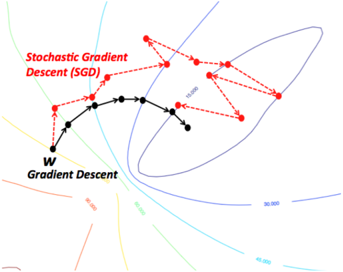
Gradient Descent

Stochastic Gradient Descent (I)

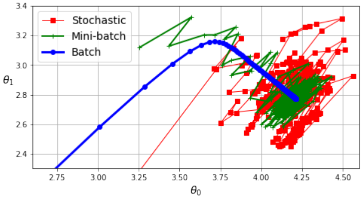
SGD approximates the gradient sampling the dataset

On-line	One sample (Stochastic Gradient Descent, SGD)
Mini-batch	Several samples (named mini-batches)
Batch	All the samples (Gradient Descent)

Computations are faster but gradient computations looses accuracy



(Source)



(Source)

Gradient Descent

Stochastic Gradient Descent(II)

In practice we use mini-batch SGD

- GPUs reduce dramatically computation time

The batch size is one of the most important hyperparameters

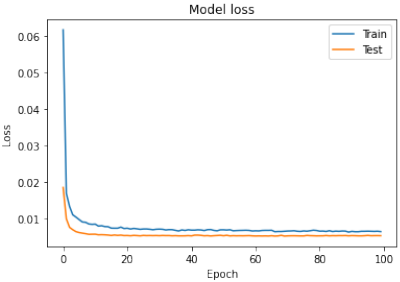
- Best performance with large batch sizes
- Batch size limited by VRAM (GPU RAM)
- Erratic gratients (or event randomness) could help to scape from local minima

Gradient Descent

Stochastic Gradient Descent (III)

Each iteration is named epoch

- Usually, an epoch involves the algorithm to visit the whole training set
- We really want to visualize the *learning curve*



(Source)

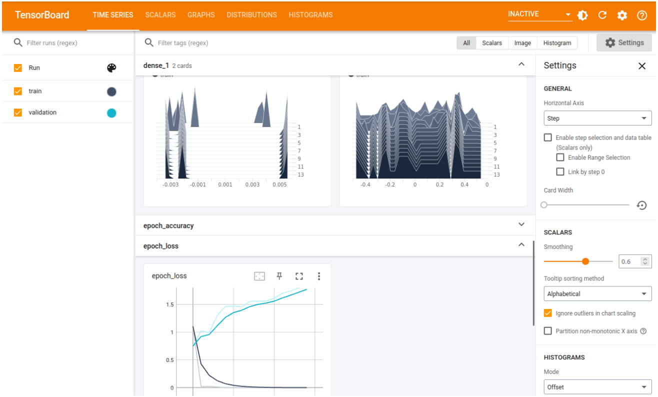
```
Epoch 1/20
363/363 [==============================] - 1s 3ms/step - loss: 0.7382 - val_loss: 2.8659
Epoch 2/20
363/363 [==============================] - 1s 2ms/step - loss: 1.3813 - val_loss: 0.8254
Epoch 3/20
363/363 [==============================] - 1s 2ms/step - loss: 0.5059 - val_loss: 0.4566
Epoch 4/20
363/363 [==============================] - 1s 3ms/step - loss: 0.4412 - val_loss: 0.4302
Epoch 5/20
363/363 [==============================] - 1s 3ms/step - loss: 0.4046 - val_loss: 0.4248
Epoch 6/20
363/363 [==============================] - 1s 2ms/step - loss: 0.4017 - val_loss: 0.4140
Epoch 7/20
363/363 [==============================] - 1s 2ms/step - loss: 0.3903 - val_loss: 0.4104
Epoch 8/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3777 - val_loss: 0.4060
Epoch 9/20
363/363 [==============================] - 1s 1ms/step - loss: 0.3720 - val_loss: 0.3976
Epoch 10/20
363/363 [==============================] - 1s 2ms/step - loss: 0.3658 - val_loss: 0.3987
Epoch 11/20
363/363 [==============================] - 1s 2ms/step - loss: 0.3620 - val_loss: 0.3878
Epoch 12/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3586 - val_loss: 0.3882
Epoch 13/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3557 - val_loss: 0.3816
Epoch 14/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3525 - val_loss: 0.3859
Epoch 15/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3498 - val_loss: 0.3860
Epoch 16/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3488 - val_loss: 0.3752
Epoch 17/20
363/363 [==============================] - 1s 2ms/step - loss: 0.3462 - val_loss: 0.3772
Epoch 18/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3442 - val_loss: 0.3755
Epoch 19/20
363/363 [==============================] - 1s 2ms/step - loss: 0.3429 - val_loss: 0.3765
Epoch 20/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3409 - val_loss: 0.3707
```

Gradient Descent

Stochastic Gradient Descent (IV)

There are advanced tools to visualize learning curves, among other cool metrics

- The most widely known is TensorBoard



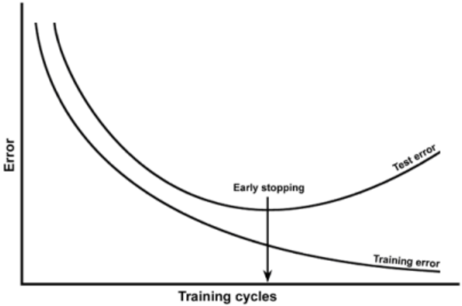
Net50

Gradient Descent

Early stopping

Early stopping: Stop training when the network begins to overfit

- Compare the loss in train and test to detect overfitting



(Source)

Backpropagation

Efficient algorithm to compute gradients

- Proposed in 1986
- First practical ANN training algorithm
- It uses the chain rule to propagate errors
- Automatic computation of gradients
- It creates a computation graph (TensorFlow takes its name from this)

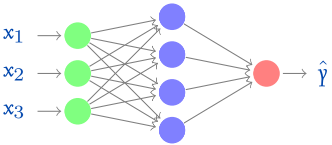
Implicit in ANN/DL packages

- You will find it as SGD
- ANN/DL packages name the optimization algorithm as 'optimizers'

Backpropagation

Backpropagation

1. **Feed-forward step**. Feed input, one mini-batch at a time. Compute output and error
2. **Feed-backward step**. Compute individual contribution to error (gradients) using the chain rule
3. **Adjust weights**. Perform a Gradient Descent step using the computed gradients



Other optimization algorithms

Momentum optimization

SGD does not take care about past gradients

$$\vec{\theta} \leftarrow \vec{\theta} - \eta \nabla_\theta J(\vec{\theta})$$

Usually, a momentum vector is introduced as

$$\vec{m} \leftarrow \beta \vec{m} - \eta \nabla_\theta J(\vec{\theta})$$

$$\vec{\theta} \leftarrow \vec{\theta} + \vec{m}$$

where ...

- η is the learning rate
- β is the momentum strength
    - If $\beta = 0$ then gradient descent
    - $\beta = 0{,}9$ uses to be a good default

(On-line demo)

Training algorithms

Other optimization algorithms

Learning rate / momentum adaptative methods

- Nesterov Accelerated Gradient - Modified momentum
- AdaGrad - Adaptative Gradient Algorithm
- RMSProp - Root Mean Square Propagation
- Adam - Adaptive Moment Estimation
  - Adaptative learning rate
  - Default choice in real-word problems

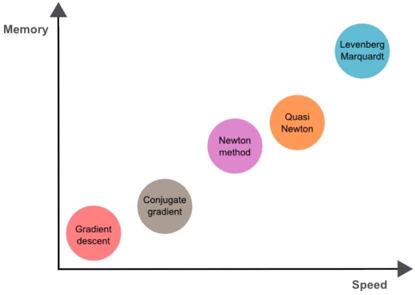
Training algorithms

Second order optimization algorithms

Other second derivative-based optimization algorithms

- Newton's method
- Quasi-Newton's method
- Levenberg-Marquardt method
- Conjugate Gradient

They are never used in DL



(Source)