Module 1 :

Machine Learning Review

Build a ML algorithm with Neural Networks

Géraldine Schaller-Conti, August 2024

Discussion Session

- Review of Notebooks 3.1 and 3.2 :
- **Dimensionality reduction** : PCA, MINST compression example, Elbow method, Kernel PCA, grid search optimization, LLE, MDS, Isomap, t-SNE
- **Clustering** : k-means

Bibliography

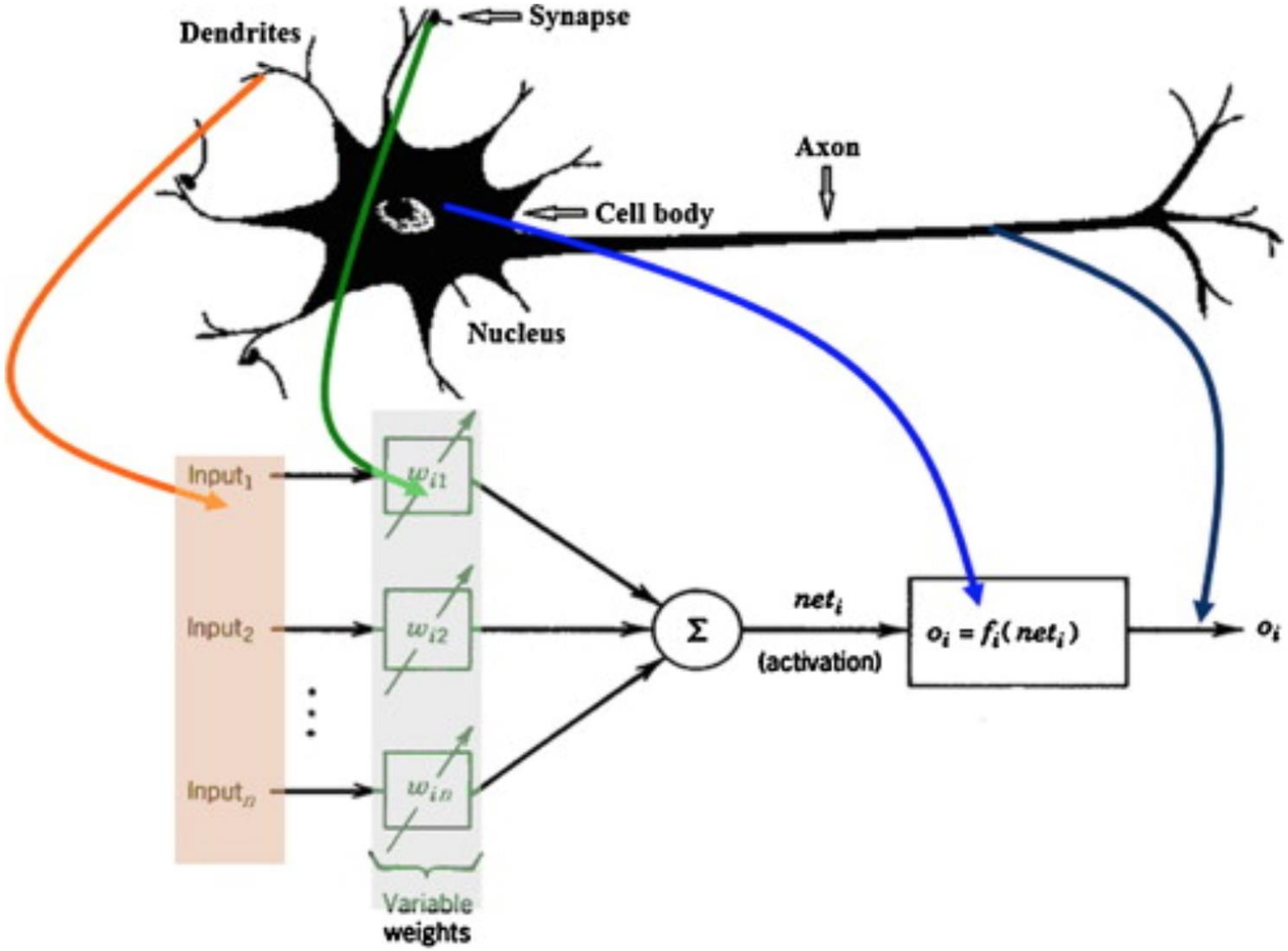
- Deep Learning book (Goodfellow, Bengio, Courville)
- Machine Learning @ Stanford (Prof Andrew Ng)
- Hands-On Machine Learning with Scikit-Learn & Tensorflow (Aurélien Géron)

Learning Objectives

- Neural Networks
- Training the NN
- Activation functions
- Loss functions
- Faster optimizers
- Neural Network as alternative

Neural Networks

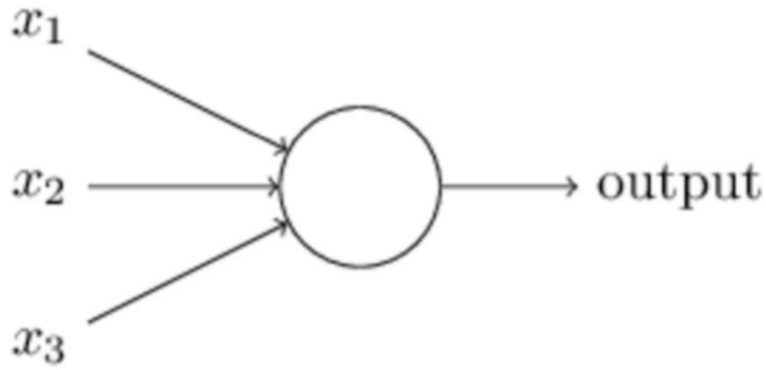
Neural Network (NN)



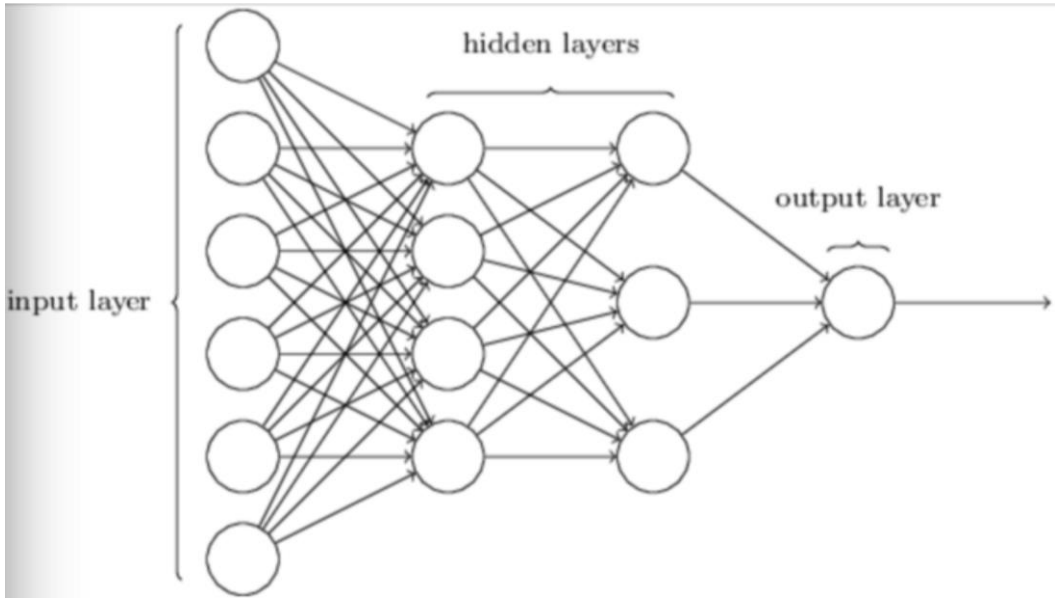
Learning algorithm inspired by *how the brain works*

History

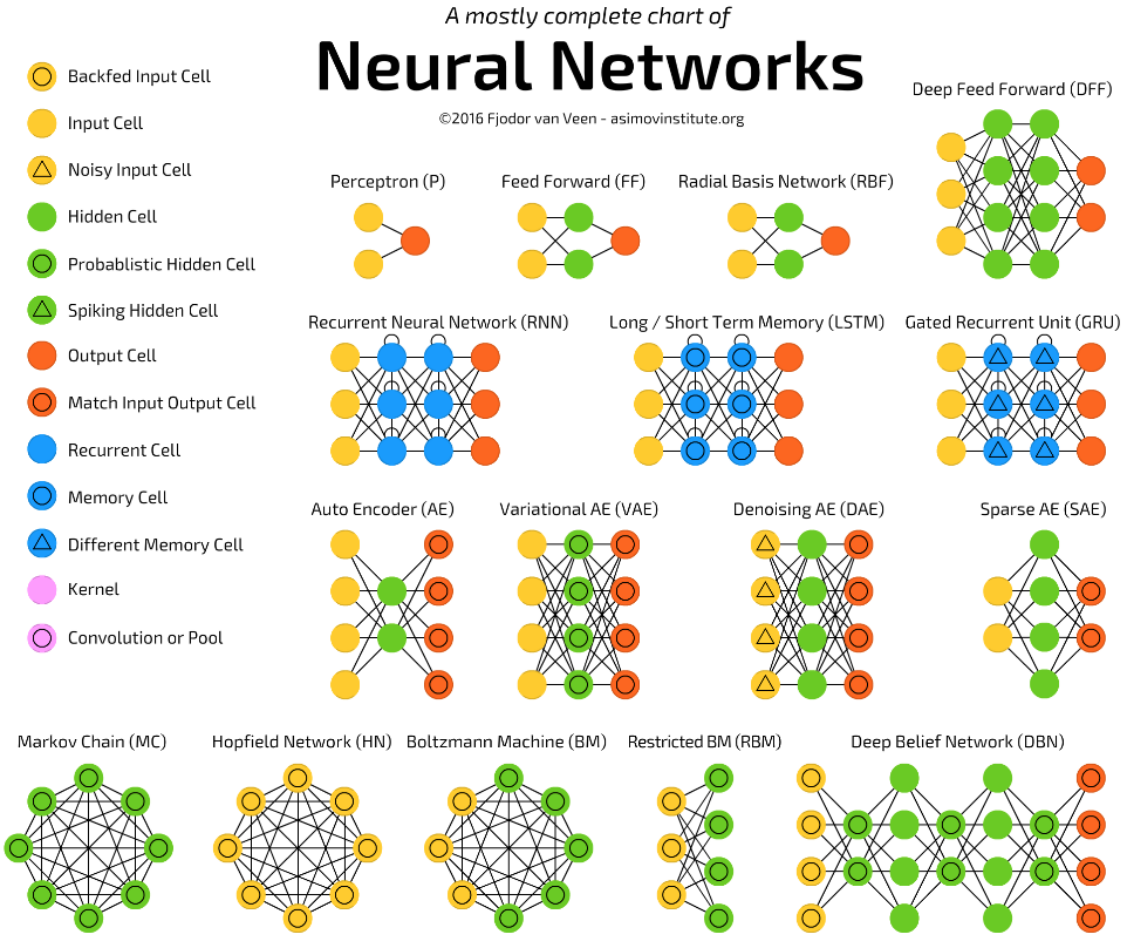
- The first single-neuron network called perceptron was proposed already in 1958 by AI pioneer Frank Rosenblatt

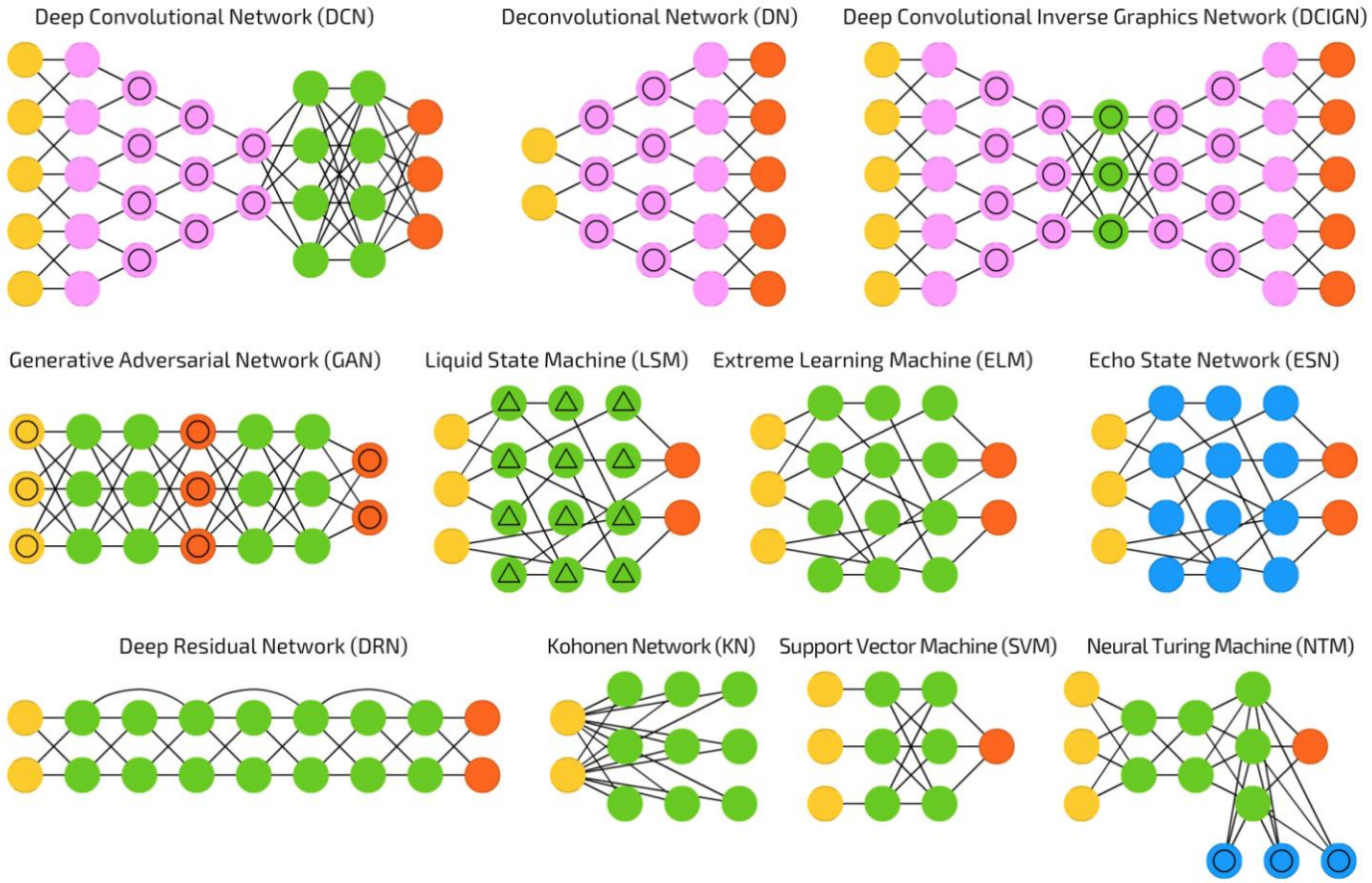


- Combining many layers of perceptrons is known as multilayer perceptrons (or FNN)



Nowadays


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)

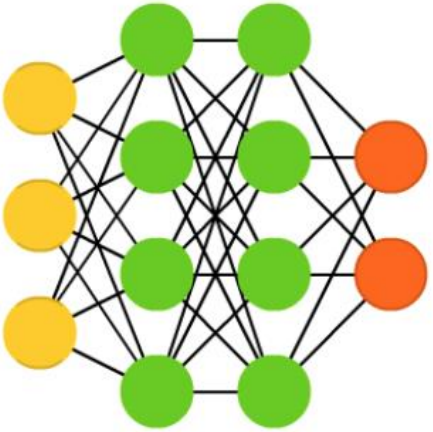

Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Feedforward Neural Networks

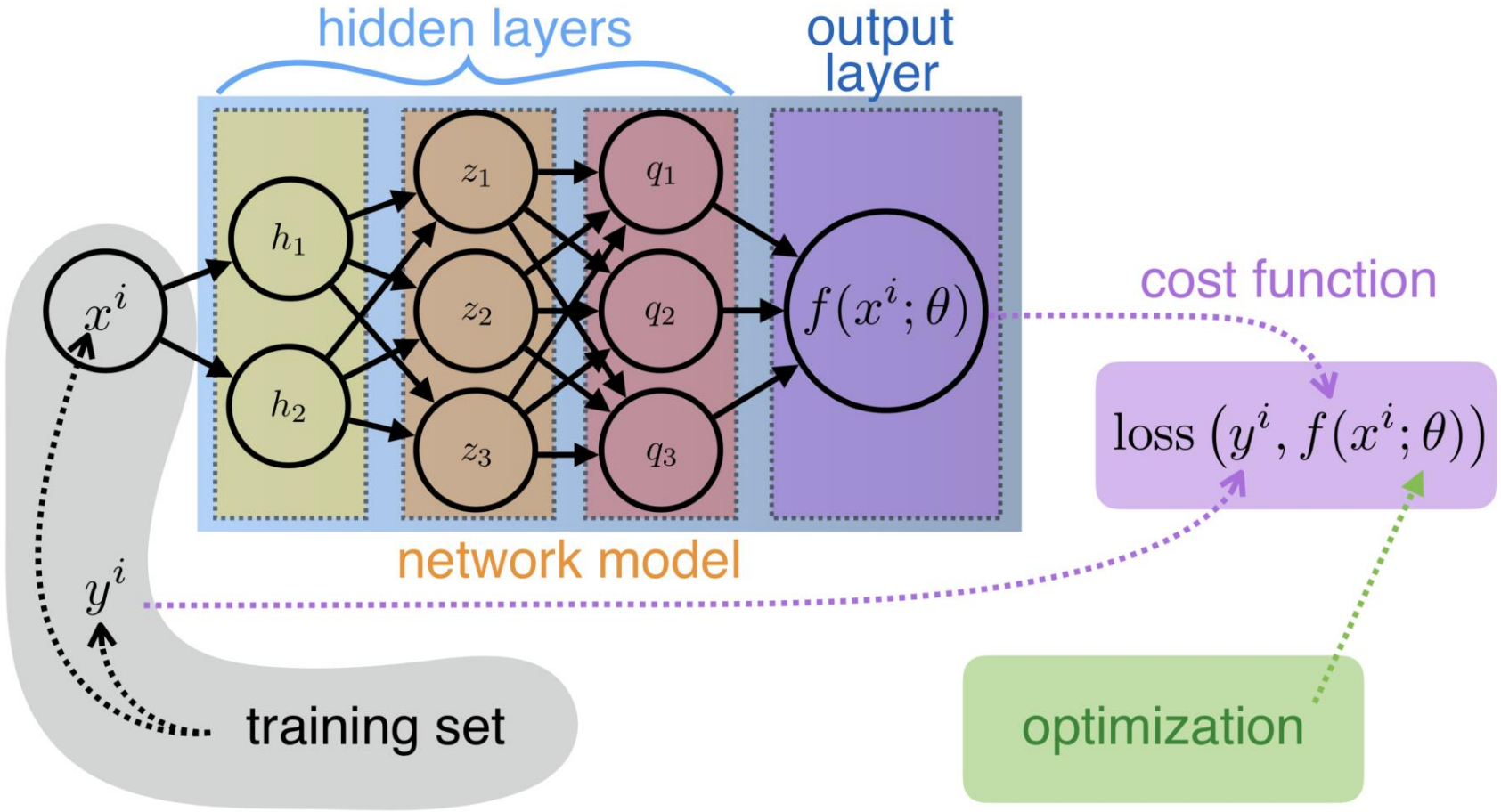
- Naming :
  - Deep feedforward networks (DFNN)
  - multilayer perceptrons(MLPs)
- Goal : approximate some function f
- feedforward = information flows from input to output layer without feedback loops
- Deep for “more than 1 hidden layer”

Feed Forward (FF)

Deep Feed Forward (DFF)



Deploying a Neural Network



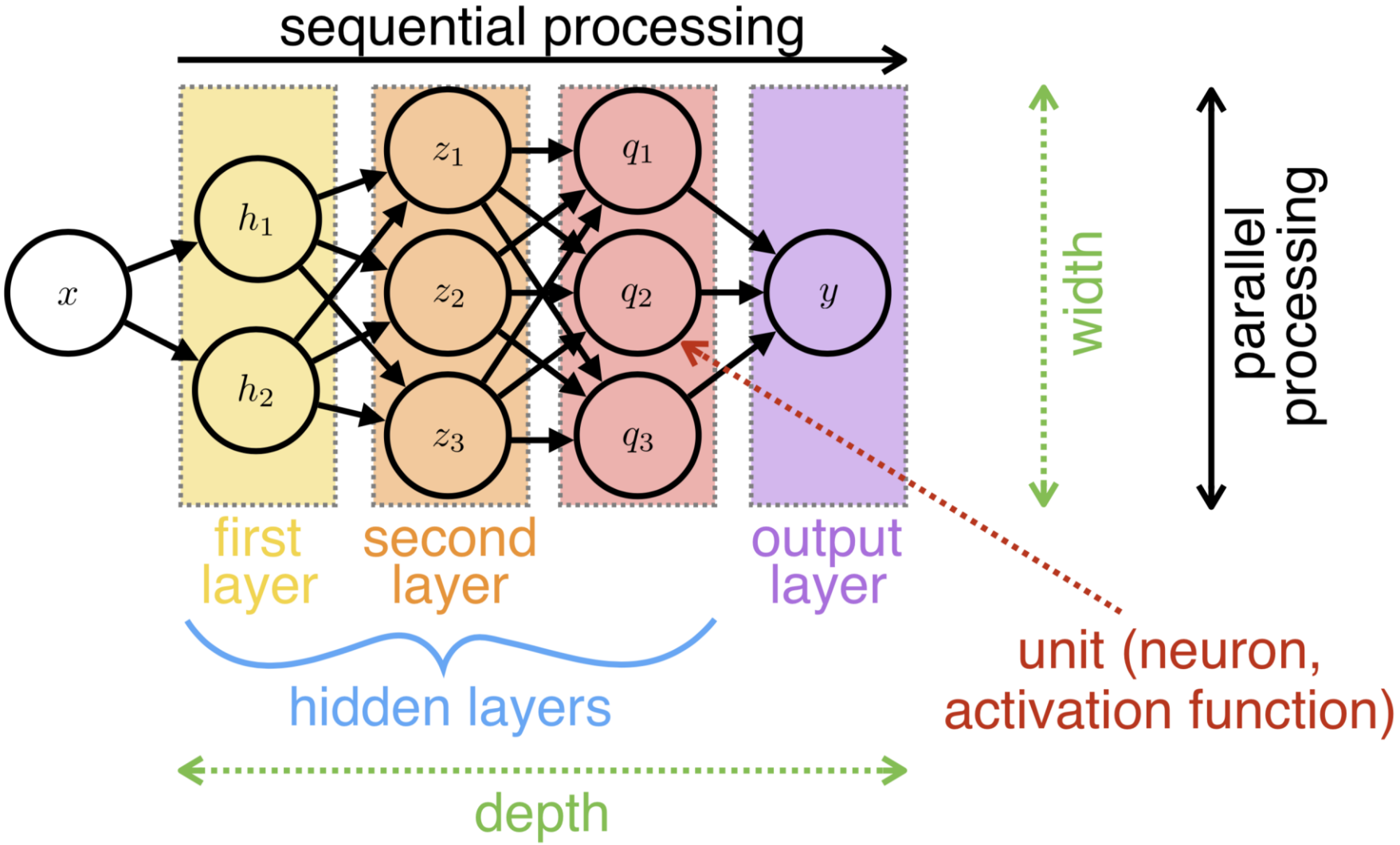
Given a task (in terms of I/O mappings), we need :

1) Network model

2) Cost function

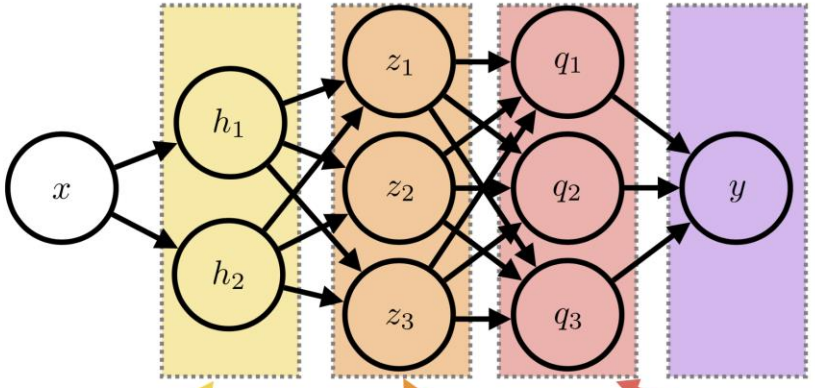
3) Optimization

Network Model



Dense layer : fully-connected layer

Activation Functions



$$y = f_4(q_1, q_2, q_3)$$

$$h_1 = f_{1,1}(x)$$
$$h_2 = f_{1,2}(x)$$

$$z_1 = f_{2,1}(h_1, h_2)$$
$$z_2 = f_{2,2}(h_1, h_2)$$
$$z_3 = f_{2,3}(h_1, h_2)$$

$$q_1 = f_{3,1}(z_1, z_2, z_3)$$
$$q_2 = f_{3,2}(z_1, z_2, z_3)$$
$$q_3 = f_{3,3}(z_1, z_2, z_3)$$

Hierarchical representation

$$y = f_4(f_{3,1}(f_{2,1}(f_{1,1}(x), f_{1,2}(x)), \dots), \dots)$$

Fully connected

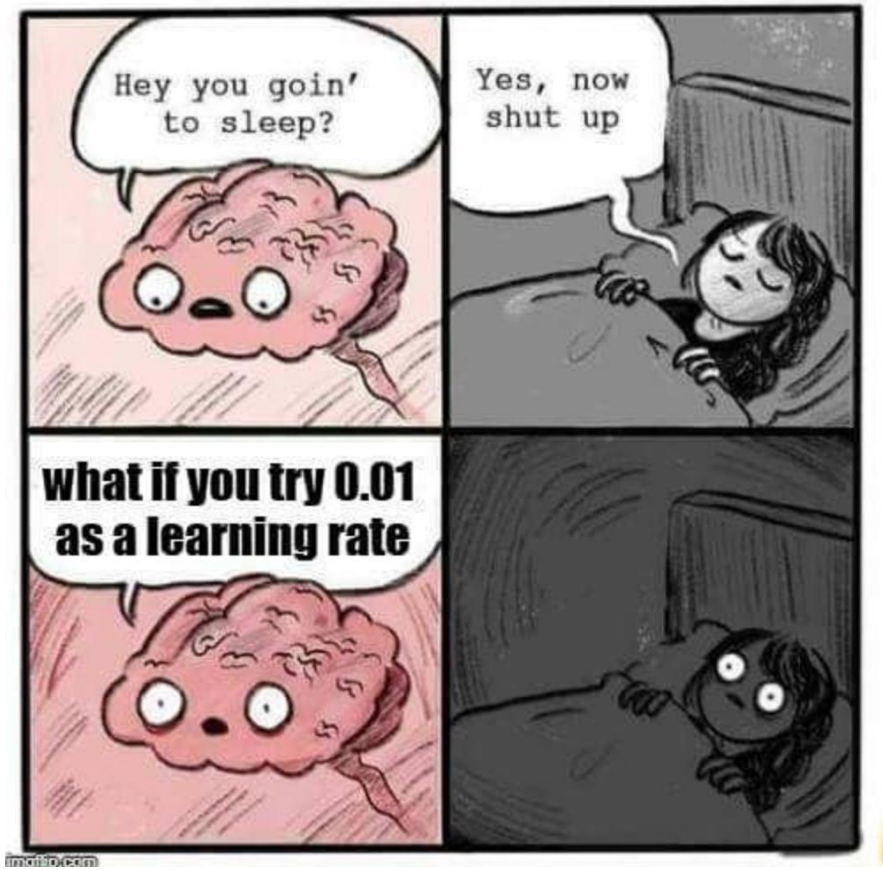
$$f_{2,2}(h_1, h_2) = w_1 h_1 + w_2 h_2 + b_2$$

Weights w and bias b parameters to optimize

- Non-linear activation functions used in hidden layers
  - Help model to generalize or adapt with variety of data

Hyperparameters

Parameters that cannot be learnt directly from training data



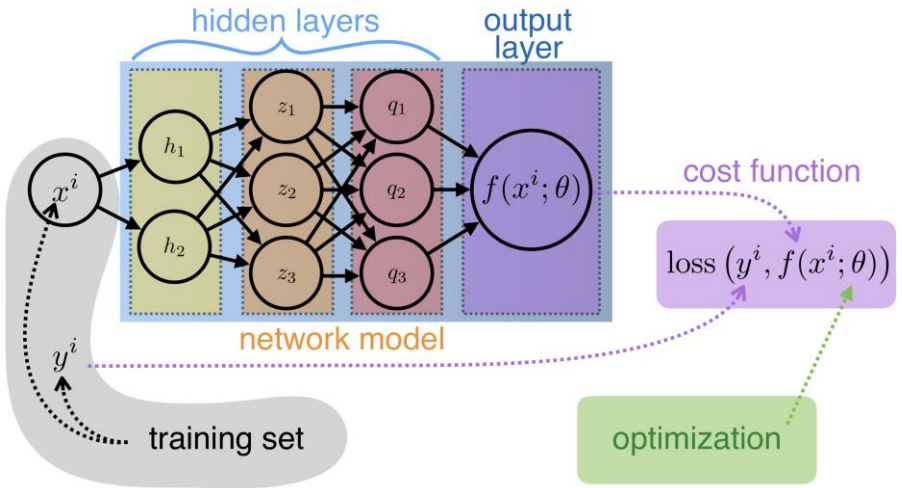
- A long list…
  - Number of hidden layers
  - Number of hidden units
  - …

Training the NN

- Training loop
- Backpropagation
- Activation functions
- Loss functions
- Faster optimizers

Optimization

- Given a task we define
  - Training data $\{x^i, y^i\}_{i=1,\ldots,m}$
  - Network $f(x;\theta)$
  - Cost function $J(\theta) = \sum_{i=1}^{m} \text{loss}\left(y^i, f(x^i;\theta)\right)$
  - Parameter initialization (weights, biases)



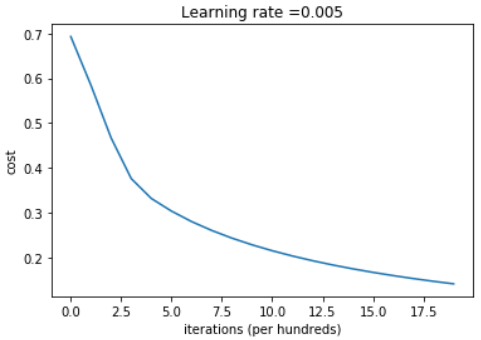
- Next, we **optimize the network parameters θ** (training)

- In addition, we have to set values for hyperparameters

Training Loop

- *Iterative* process

Learning curve



Forward propagation

$$Z = w^T x + b$$

$$A = \sigma(Z)$$

Make a prediction

Cost function

$$J(w, b) = J(\theta)$$

Define the error function

epochs

Backward propagation
(dJ/dw, dJ/db)

Measure the error contribution from each connection

Parameter update
(**gradient descent**)

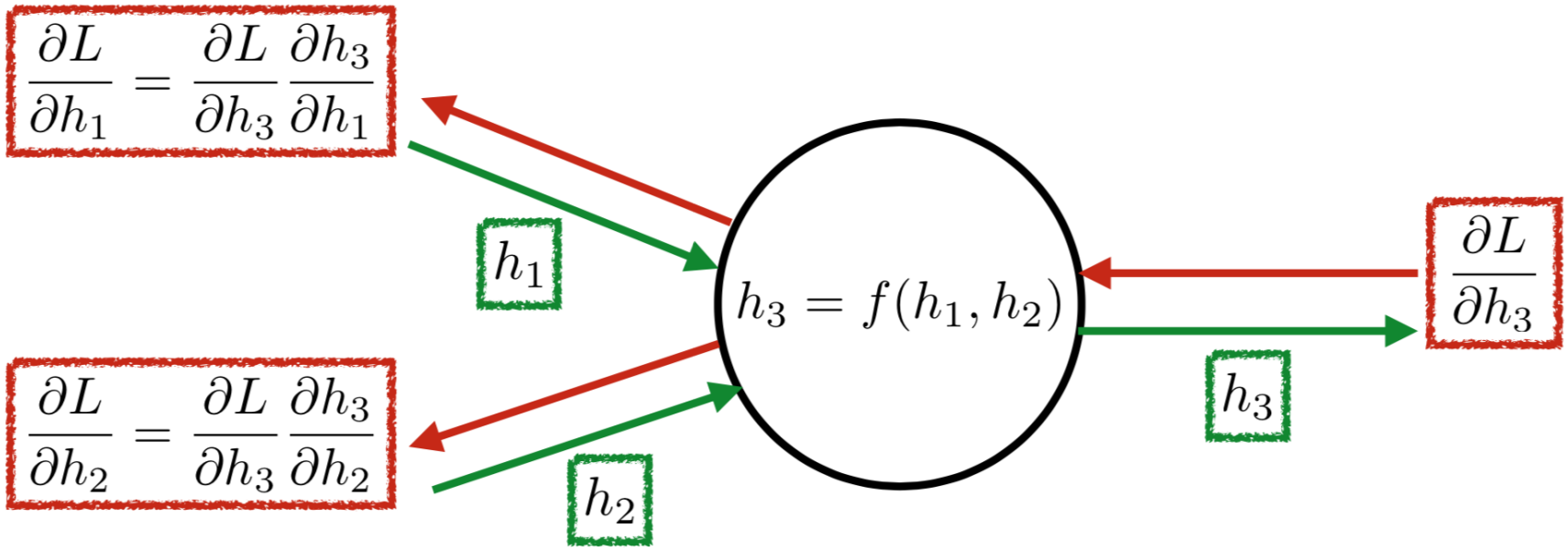
Tweaks the connection weights to reduce the error

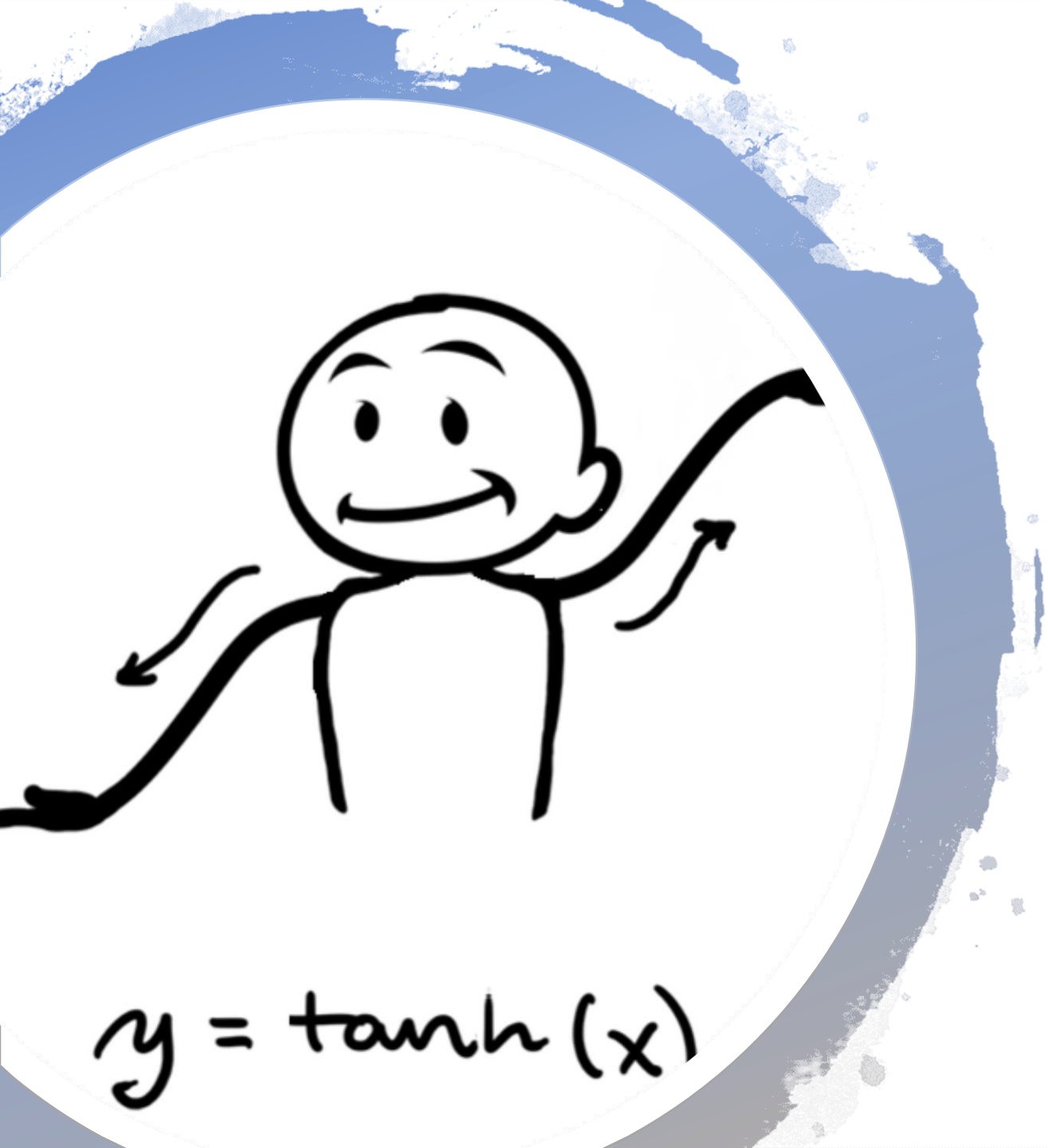
learning rate α

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t)$$

Backpropagation

- Efficient implementation of the chain-rule to compute derivatives with respect to network weights



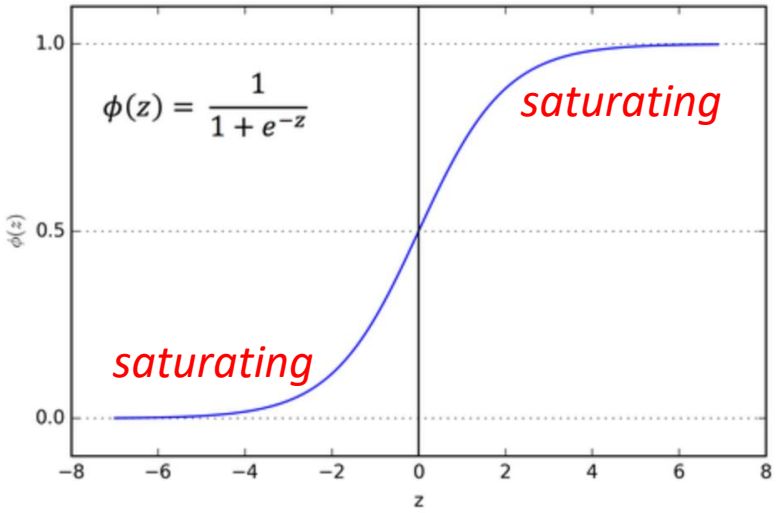


Activation Functions

Used to introduce non-linearity into the neural network. Non-linearity is crucial for a NN to model complex, non-linear relationships in the data

Sigmoid Function

- Looks like a S-shape in the [0,1] range



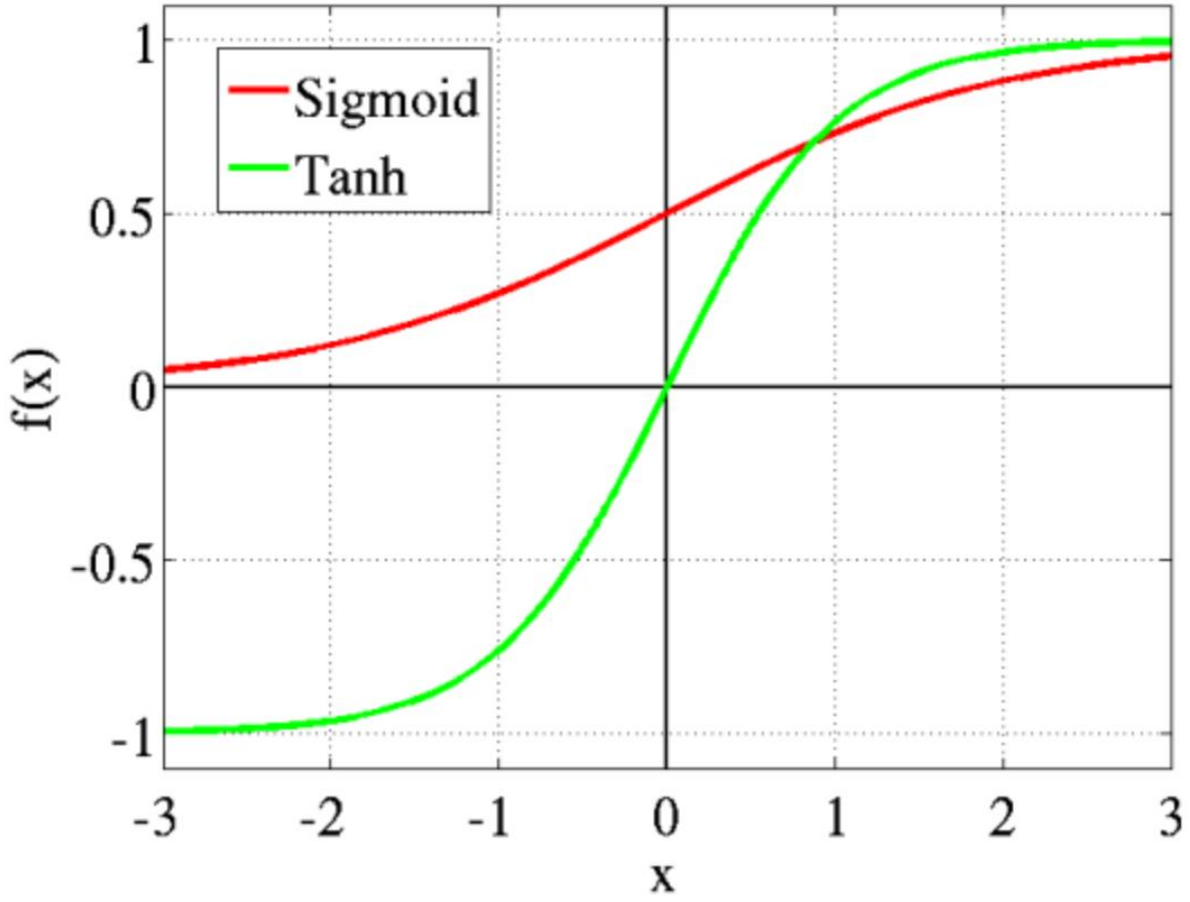
- Softmax function used for multiclass classification

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

- Used for models where we have to predict the probability as an output

Tanh Function

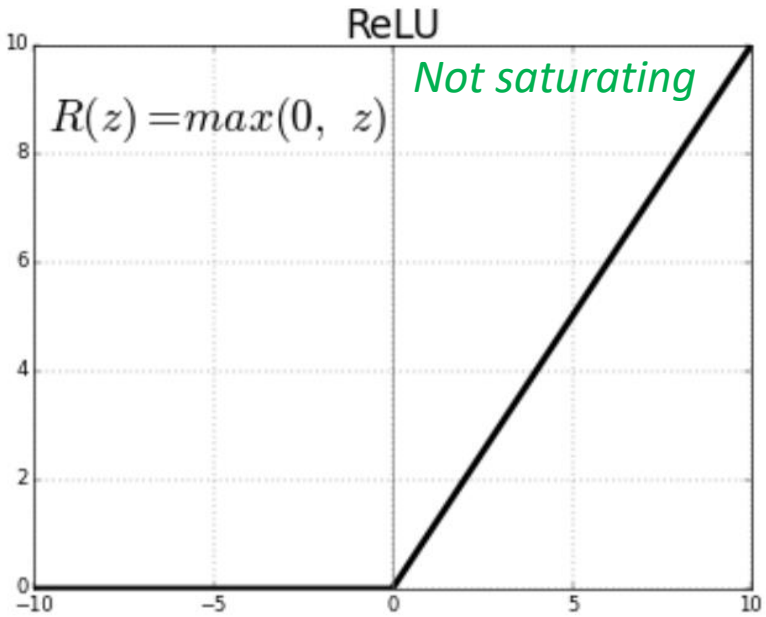
- S-shape in the [-1,1] range



- Used for classification between two classes

ReLU Function

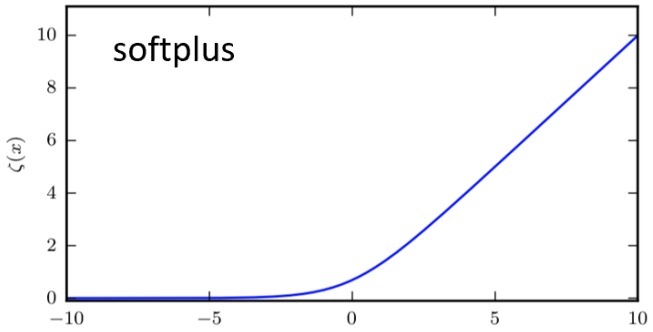
- Rectified Linear Unit (ReLU) in the [0, infinity) range



- Most used activation function right now
  - Faster to compute than other activation functions

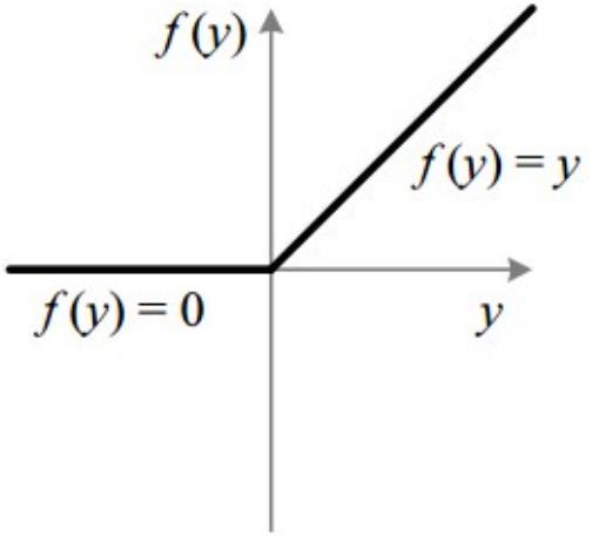
- Softplus : Smooth approximation

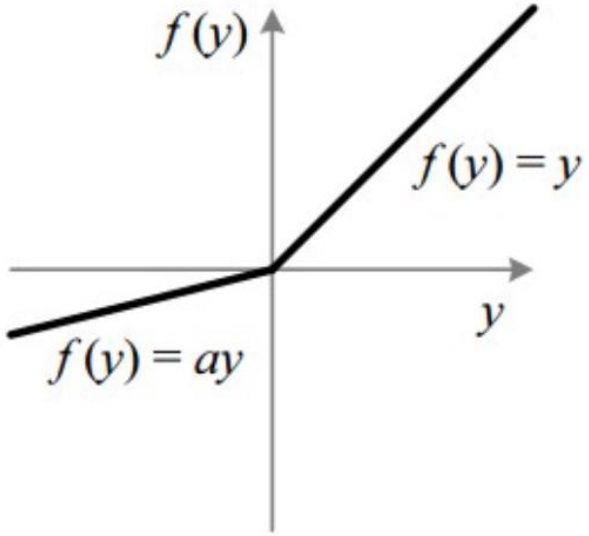
$$\zeta(x) = \log(1 + \exp(x))$$



Leaky ReLU Function

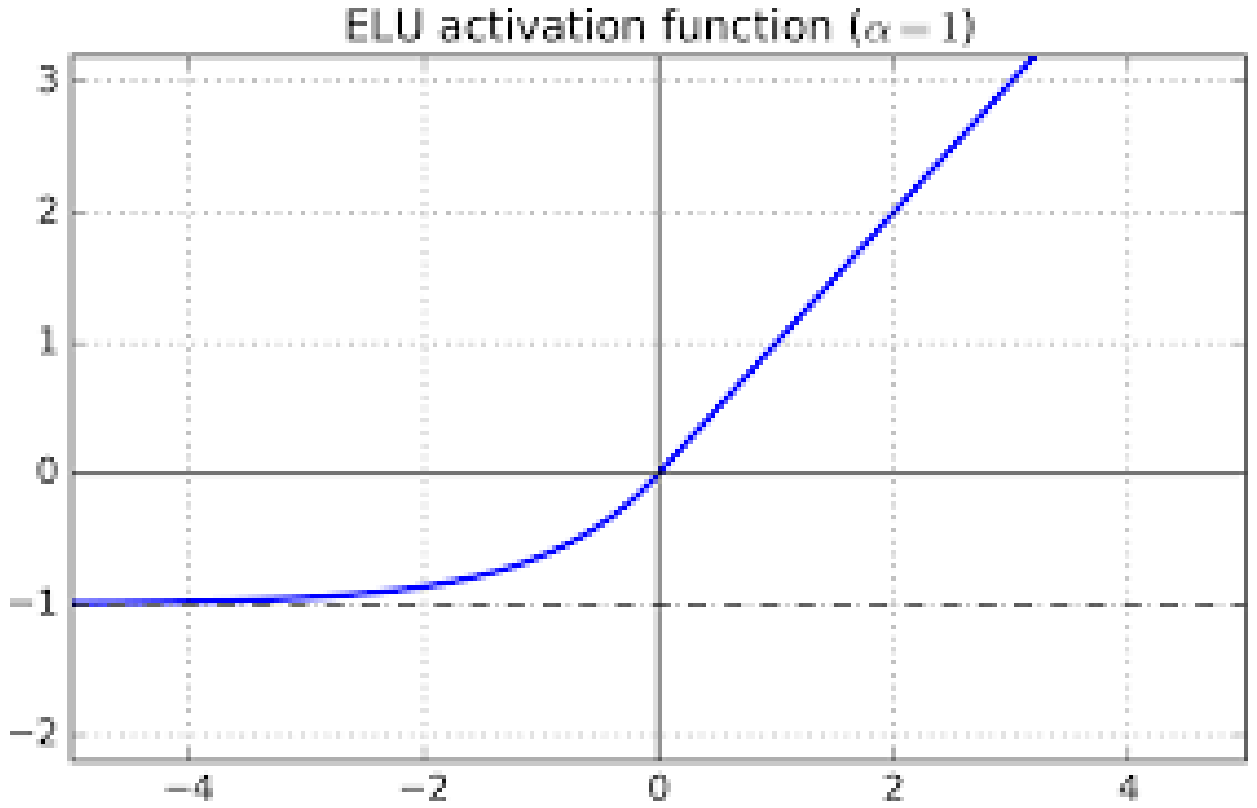
- Attempt to solve the dying ReLU problem in the (-infinity, +infinity) range





- The leak α = 0.2 seems to lead to a good performance

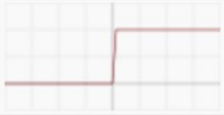
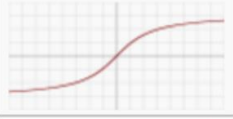
ELU Function



- Smooth everywhere, including around z=0

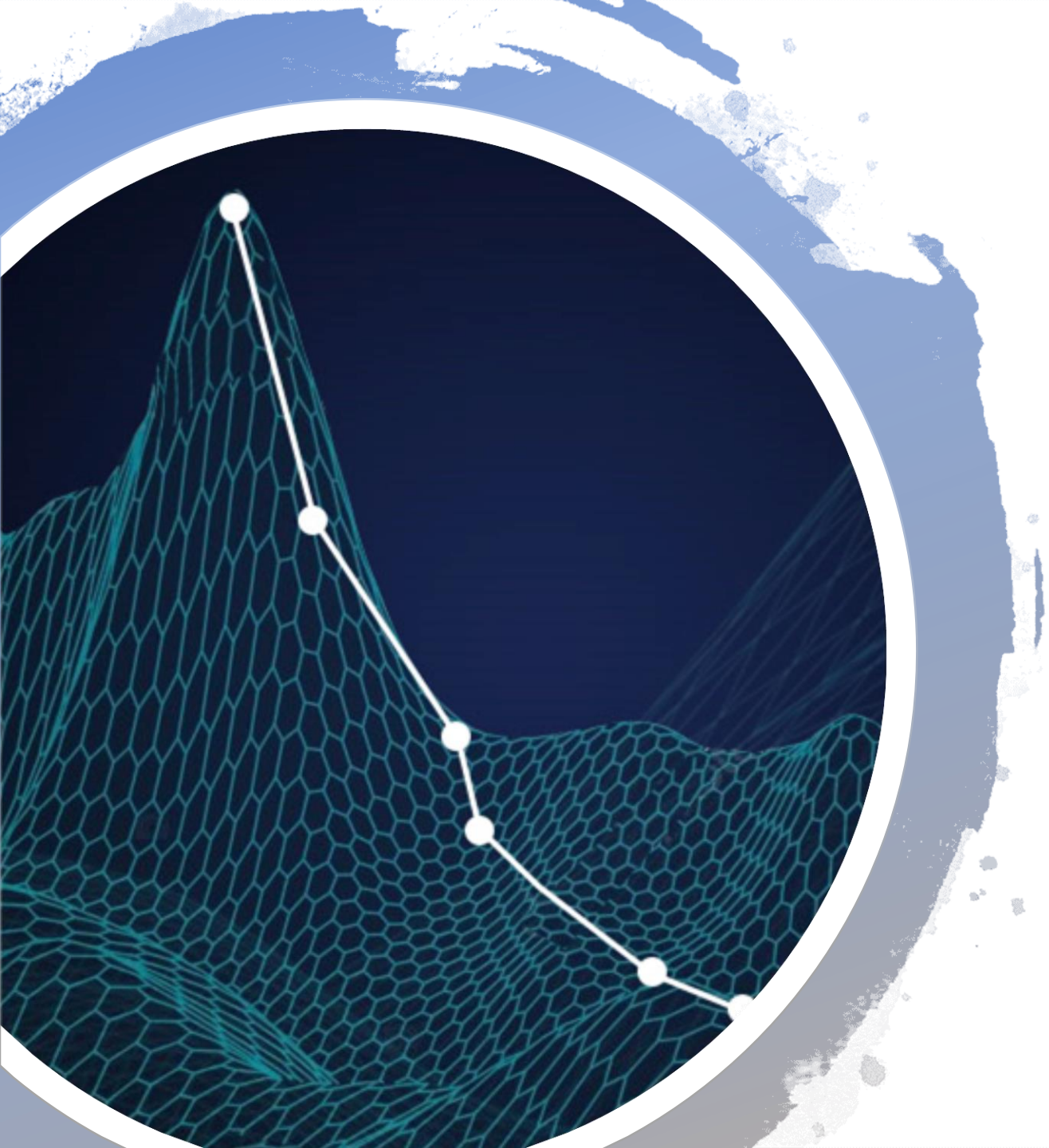
- slower to compute than RELU
  - During training : compensated by faster convergence rate
  - During testing : slower

Summary

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

In practice (Keras)

- activation='linear’
- activation='sigmoid’
- activation='tanh’
- activation='softmax’



Loss functions

Depend on the task to be performed

Regression Loss Functions

1) Mean Squared Error Loss (MSE), L2 Loss

- Distribution of target variable is a standard Gaussian
- Average of the squared differences between predicted and true values

$$L_2(\hat{y}, y) = \frac{1}{m}\sum_{i=0}^{m}\left(y^i - \hat{y}^i\right)^2$$

2) Mean Squared Logarithmic Error Loss (MSLE)

- calculate the natural log of predicted values, then MSE
- **Use case**: if target value has a spread of values, and when predicting a large value one does NOT want to punish the model as heavily as MSE

3) Mean Absolute Error Loss (MAE)

- **Use case** : target variable may be mostly Gaussian, but with outliers

Binary Classification Loss Functions

1) Binary Cross-Entropy Loss

- Score that summarizes the average difference between the actual and predicted probability distributions for predicting class 1

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

2) Hinge Loss

- **Use case :** Binary classification where the target values are {-1, 1}
- Assign more error when there is a difference in the sign between the actual and predicted class values

- Squared Hinge Loss : calculates the square of the score hinge loss to smoothen the surface of the error function

Multi-Class Classification Loss Functions

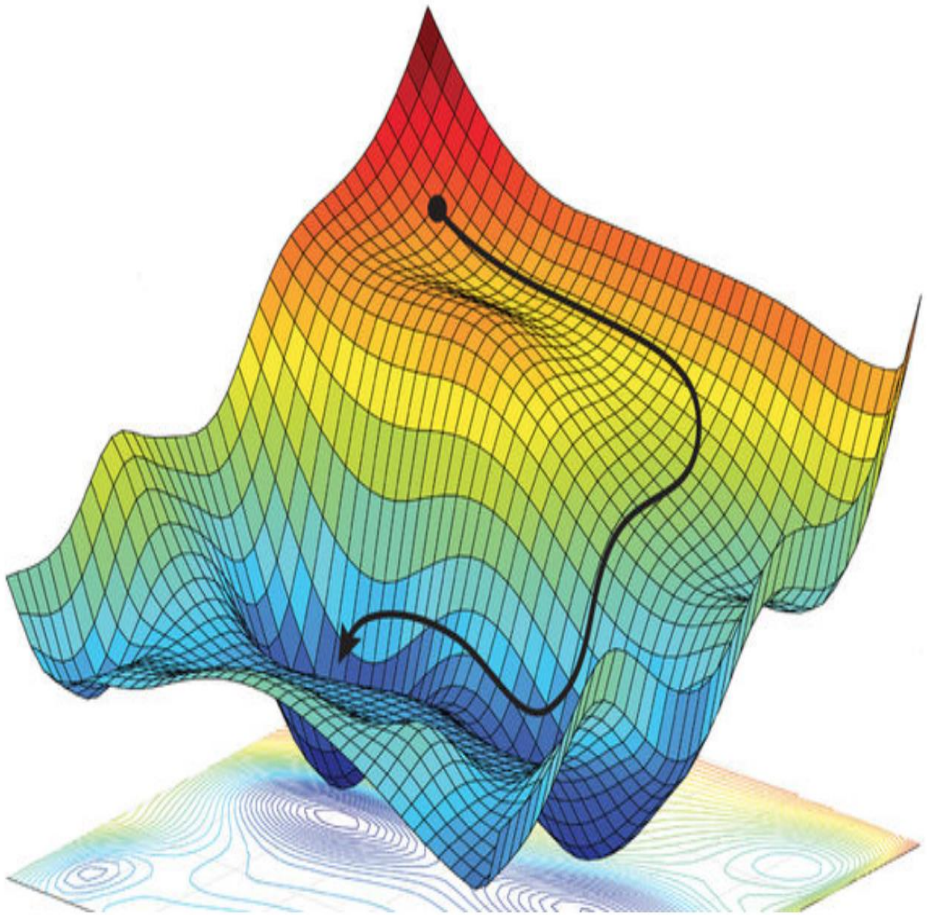
- Multi-Class Cross-Entropy Loss
  - Target set {0,1,2,…,n} : need for one-hot encoding
  - Generalization of binary cross-entropy loss to n classes
- Sparse Multiclass Cross-Entropy Loss
  - No need to have the target variable be one-hot encoded
    - Tackles the problem of one-hot encoding when too many categories
- Kullback Leibler Divergence Loss (KL)
  - Measure of how one probability distribution differs from a baseline distribution
  - Mainly used when using models that learn to approximate a more complex function that multi-class classification
    - Autoencoders

Loss Functions : In Practice (Keras)

- Regression :
  - loss='mean_squared_error’ or loss=mse’
  - loss='mean_squared_logarithmic_error’
  - loss='mean_absolute_error’
- Binary classification :
  - loss='binary_crossentropy’
  - loss='hinge’
  - loss='squared_hinge’
- Multi-Class classification:
  - loss= 'categorical_crossentropy’
  - loss= 'sparse_categorical_crossentropy’
  - loss= 'kullback_leibler_divergence’

Ingredient Summary

	Regression	Binary classification	Multi-class classification
Hidden layer activation	RELU	RELU	RELU
Geometry of output layer	Dense (1)	Dense (1)	Dense (n)
Activation of output layer	Linear	Sigmoid	Softmax
Loss function	MSE, MSLE, MAE	Binary cross-entropy	(sparse) categorical cross-entropy, KL divergence



Optimization : make it faster than vanilla GD

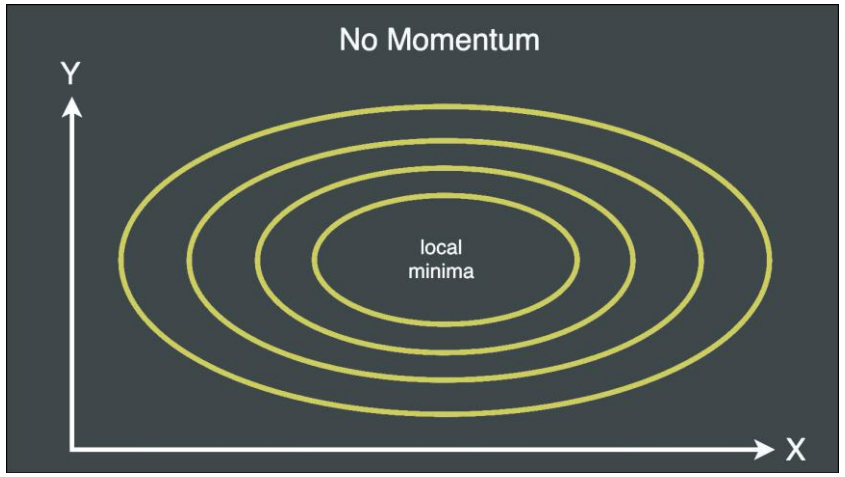
- Momentum

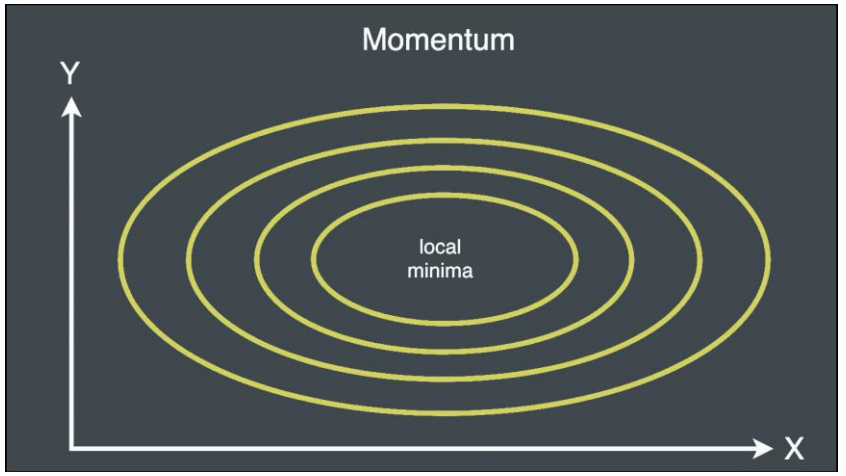
- RMSprop

- Adaptative Moment (Adam)

SGD + Momentum

- Optimization with momentum β takes past gradients into account
  - it accelerates search in direction of minima

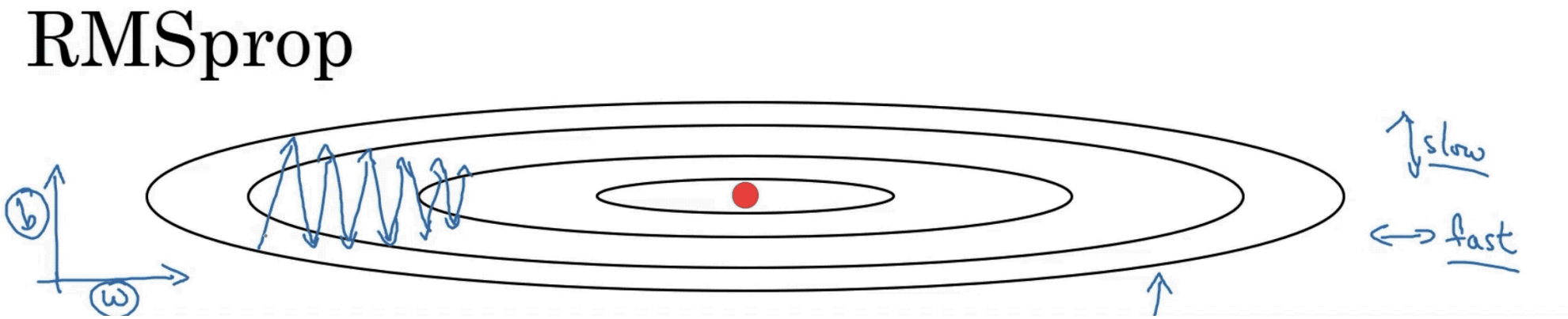




- The larger β, the smoother the update because the more we take past gradients into account
  - *$\beta = 0.9$ is a good choice (between 0.8 and 0.999)*

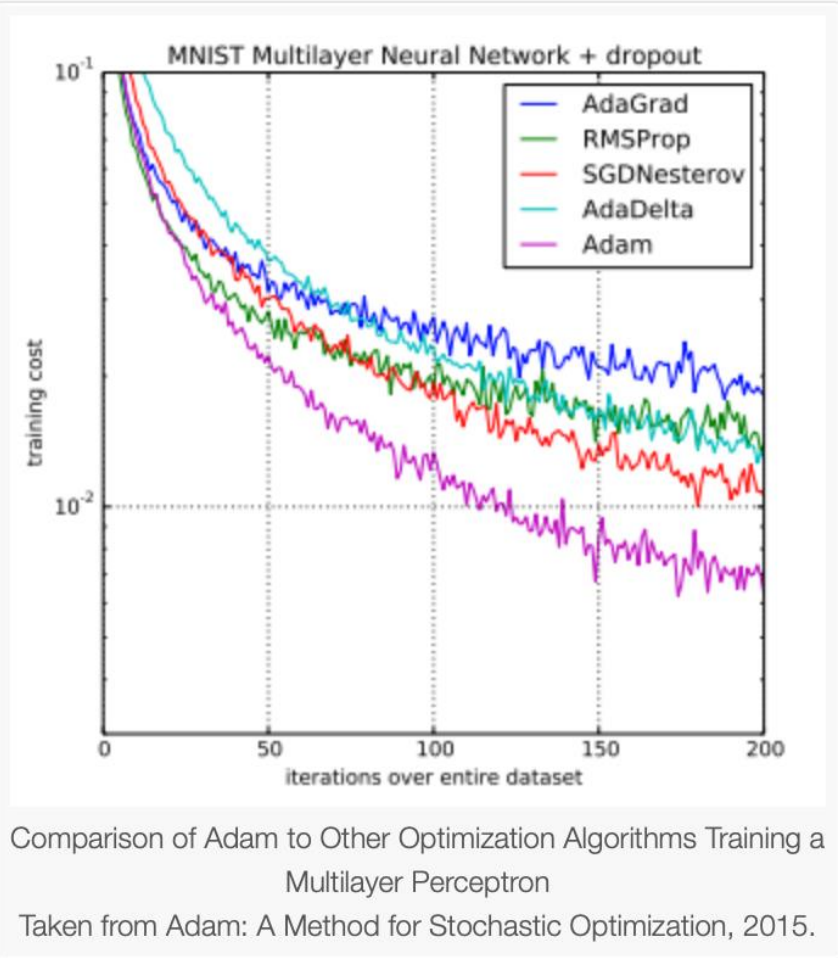
RMSProp Algorithm

- Difference in how the gradients are computed
- impedes search in direction of oscillations (vertical direction) : allows to increase α



Adam Algorithm

- Adam combines ideas from RMSProp and momentum
  - The recommended one



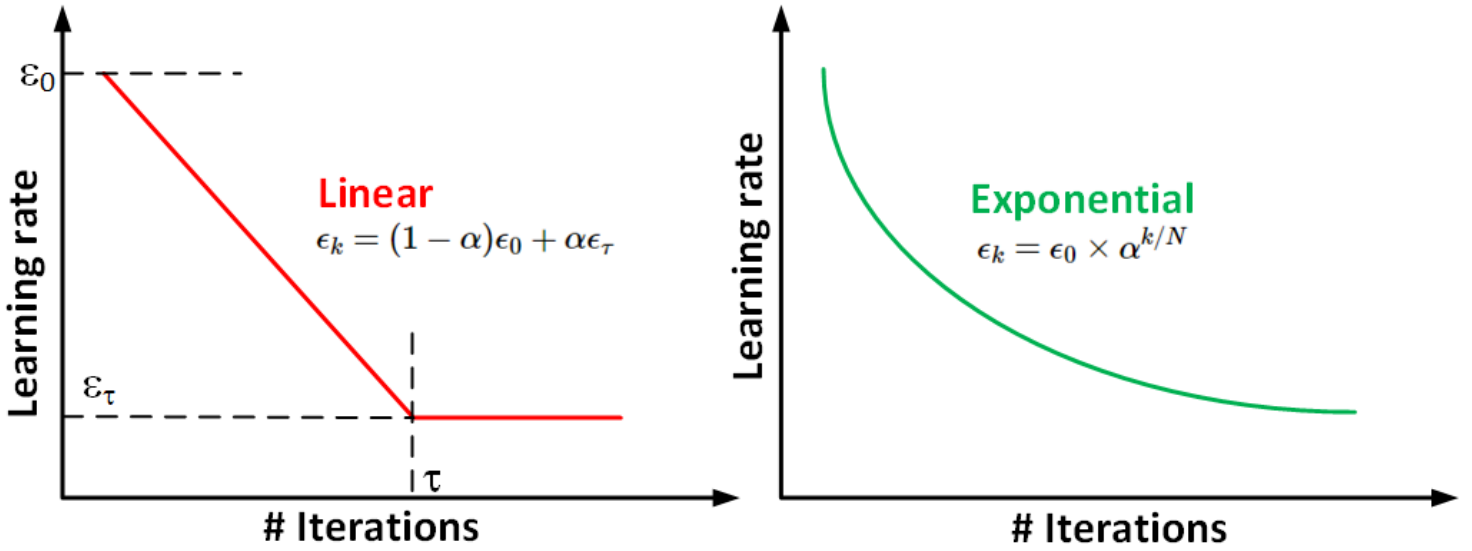
Comparison of Adam to Other Optimization Algorithms Training a Multilayer Perceptron

Taken from Adam: A Method for Stochastic Optimization, 2015.

Learning Rate Scheduling

- Start with high learning rate and reduce it once it stops making fast progress
  - solution reached faster than with the optimal constant learning rate

- Learning schedules examples:



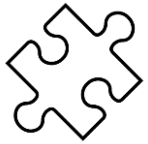
- RMSProp and Adam optimization algorithms automatically reduce the learning rate during training

Neural Networks as an alternative to other ML algorithms

Exercise

1

```
model = keras.Sequential([keras.layers.Flatten(input_shape (28,28)),
                          keras.layers.Dense(128,activation = tf.nn.sigmoid),
                          keras.layers.Dense(10,activation = tf.nn.softmax)])
model.compile(optimizer =
'adam',loss='sparse_categorical_crossentropy',metrics =['accuracy'])
```



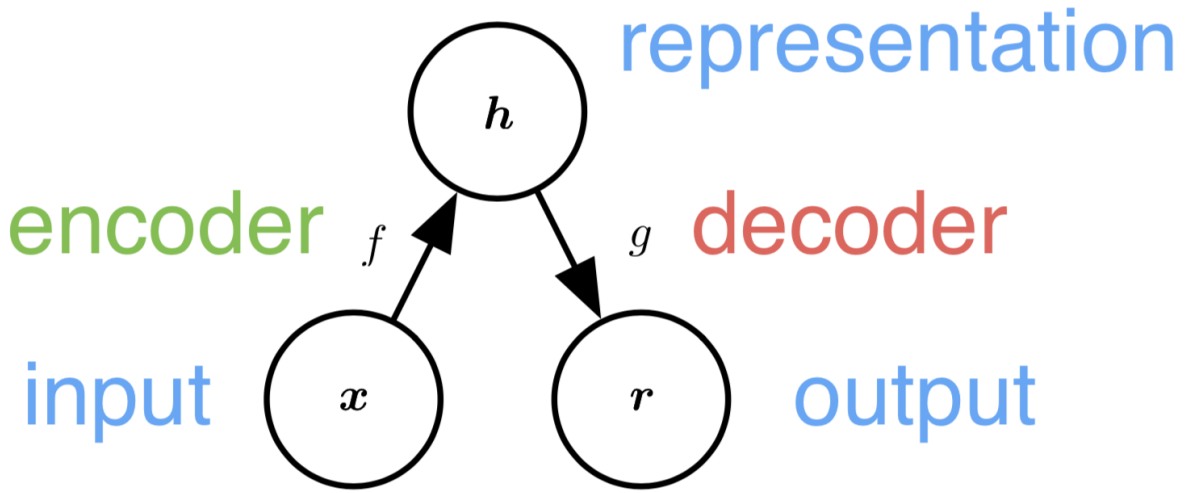
Which tasks these codes are meant for ?

2

```
NN_model = Sequential()

# The Input Layer :
NN_model.add(Dense(128, kernel_initializer='normal',input_dim = train.shape[1], activation='relu'))

# The Hidden Layers :
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))

# The Output Layer :
NN_model.add(Dense(1, kernel_initializer='normal',activation='linear'))

# Compile the network :
NN_model.compile(loss='mean_absolute_error', optimizer='adam', metrics=['mean_absolute_error'])
NN_model.summary()
```

Autoencoders (AE)

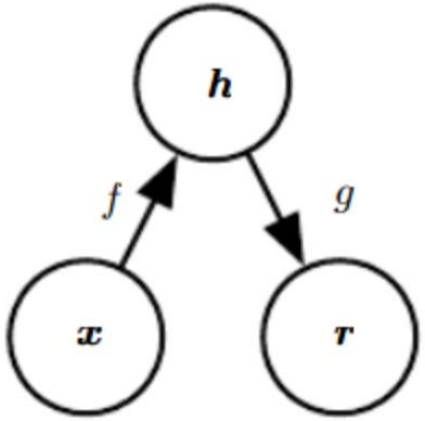
- Network that replicates the input
  - Internally, it builds a representation of the input
  - Network before the internal representation : **encoder**
  - Network following the representation : **decoder**



What tasks can autoencoders perform as an alternative to other ML algorithms ?

AE as an alternative to other ML algorithms

$h=f(x)=W^Tx+b$ is the latent variable representation of the input in a low dimensional space (linear activation function)



Loss function : L(x,g(f(x)))

r=g(h)=g(f(x)) is the reconstruction of the input from the latent representation

What should be changed for the non-linear case ?

AE versus PCA

	PCA	Autoencoders
Transformation of data	Linear	(non)-linear
Speed	Fast	Slower (gradient descent)
Transformed data	Orthogonal dimensions	Not guaranteed
Complexity	Simple transformation	can model complex relationships
Data size	Small datasets	Larger datasets
Hyperparameter	k (number of dimensions)	Architecture of the NN

- AE with single layer and linear activation has similar performance as PCA.
- AE with multiple layers and non-linear activation functions prone to overfitting (need for regularization)

https://www.youtube.com/watch?v=Rdpbnd0pCiI

Two-Minute Papers

Quiz

https://b.socrative.com/login/student/

Room : CONTI6128