Module 1 :

Machine Learning Review

Build a ML algorithm

Géraldine Conti, August 2020

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, inertia, K-means++, silhouette score, Gaussian mixtures, covariance comparison, BIC and AIC

Bibliography

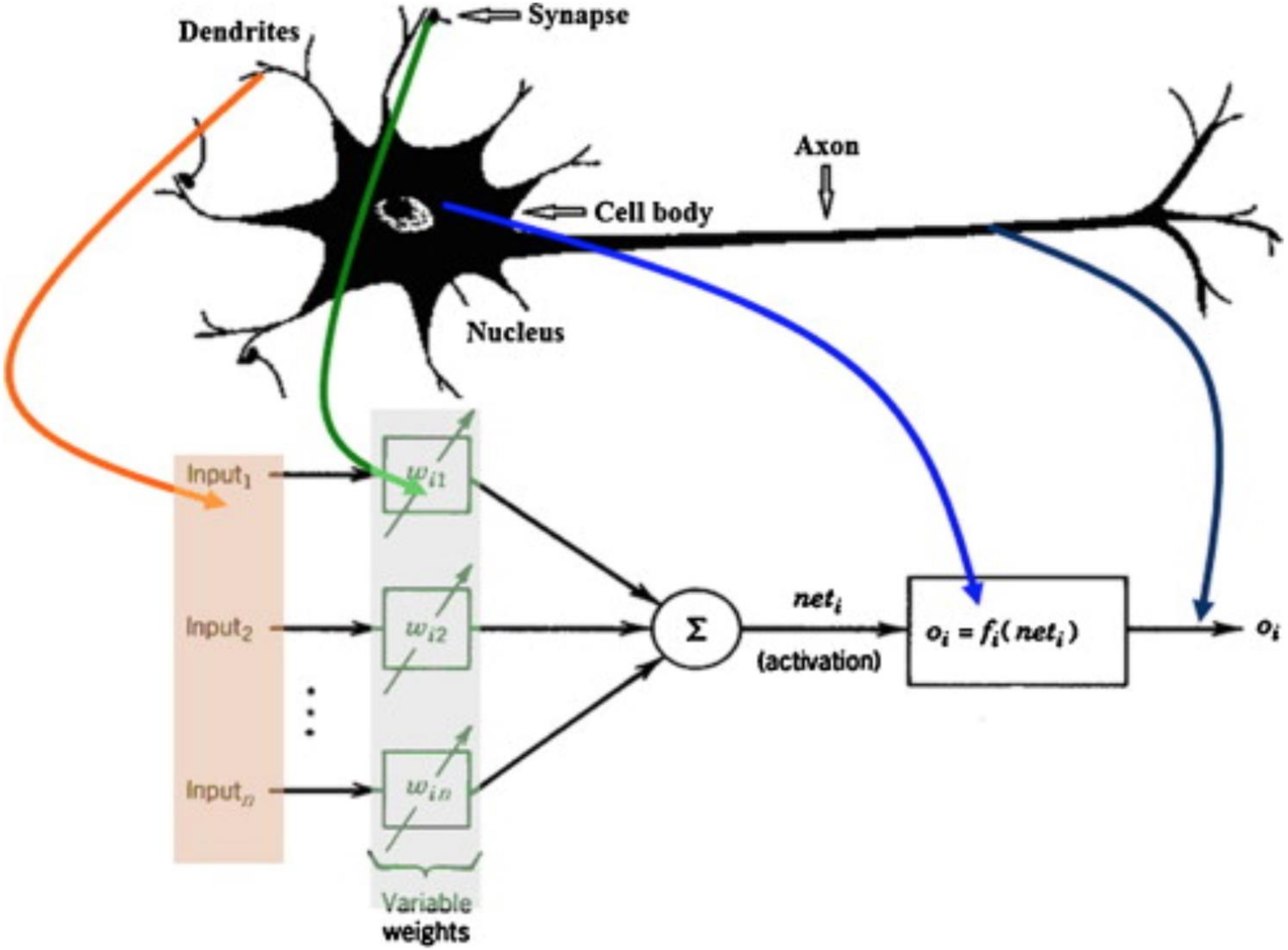
- Deep Learning book (Goodfellow, Bengio, Courville)
- Machine Learning @ Stanford (Prof Andrew Ng)
- Hands-On Machine Learning with Scikit-Learn & Tensorflow (Aurélien Géron)

Learning Objectives

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

Neural Networks

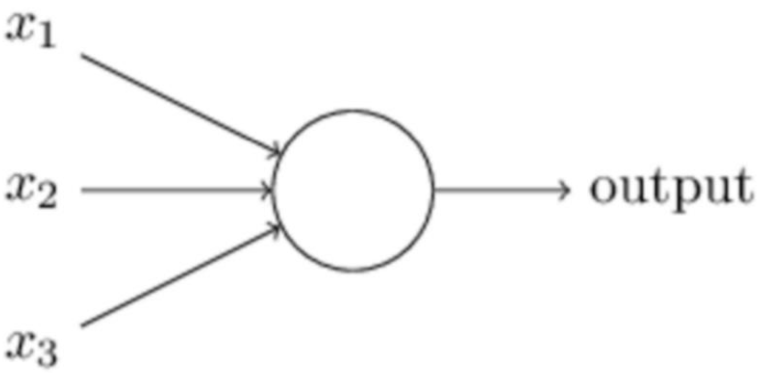
Neural Network (NN)



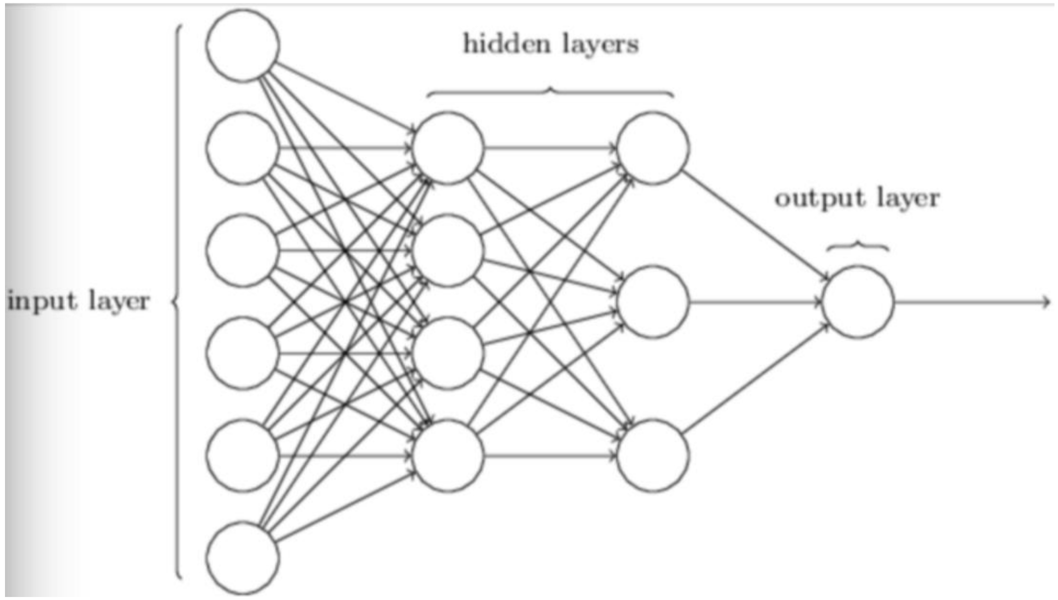
Learning algorithm inspired by *how the brain works*

History

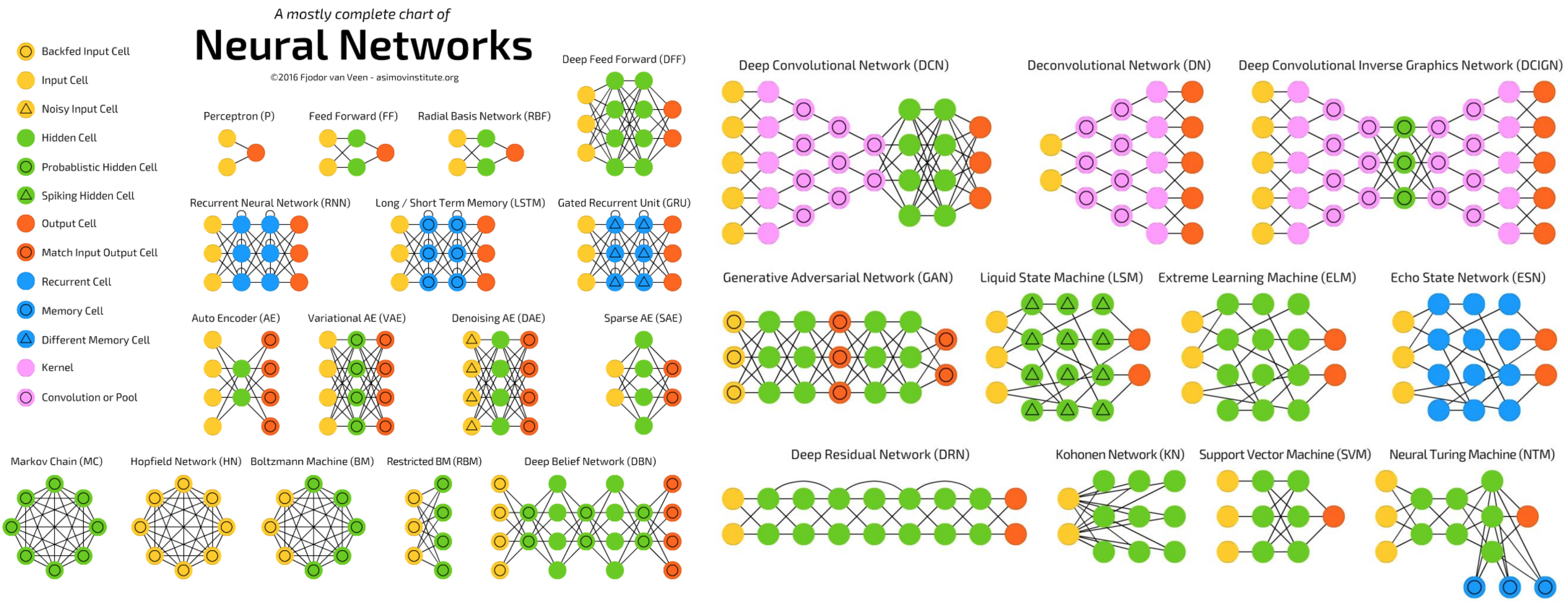
- The first single-neuron network called perceptron was proposed already in 1958 by AI pioneer Frank Rosenblatt



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

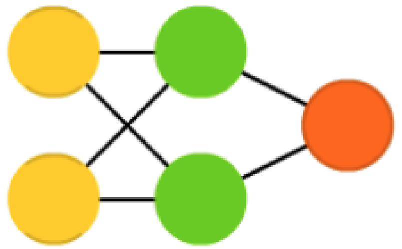


Nowadays

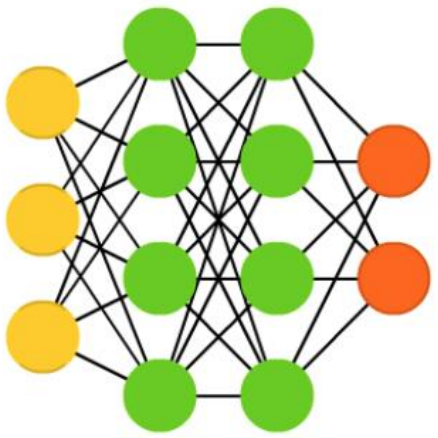

A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
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

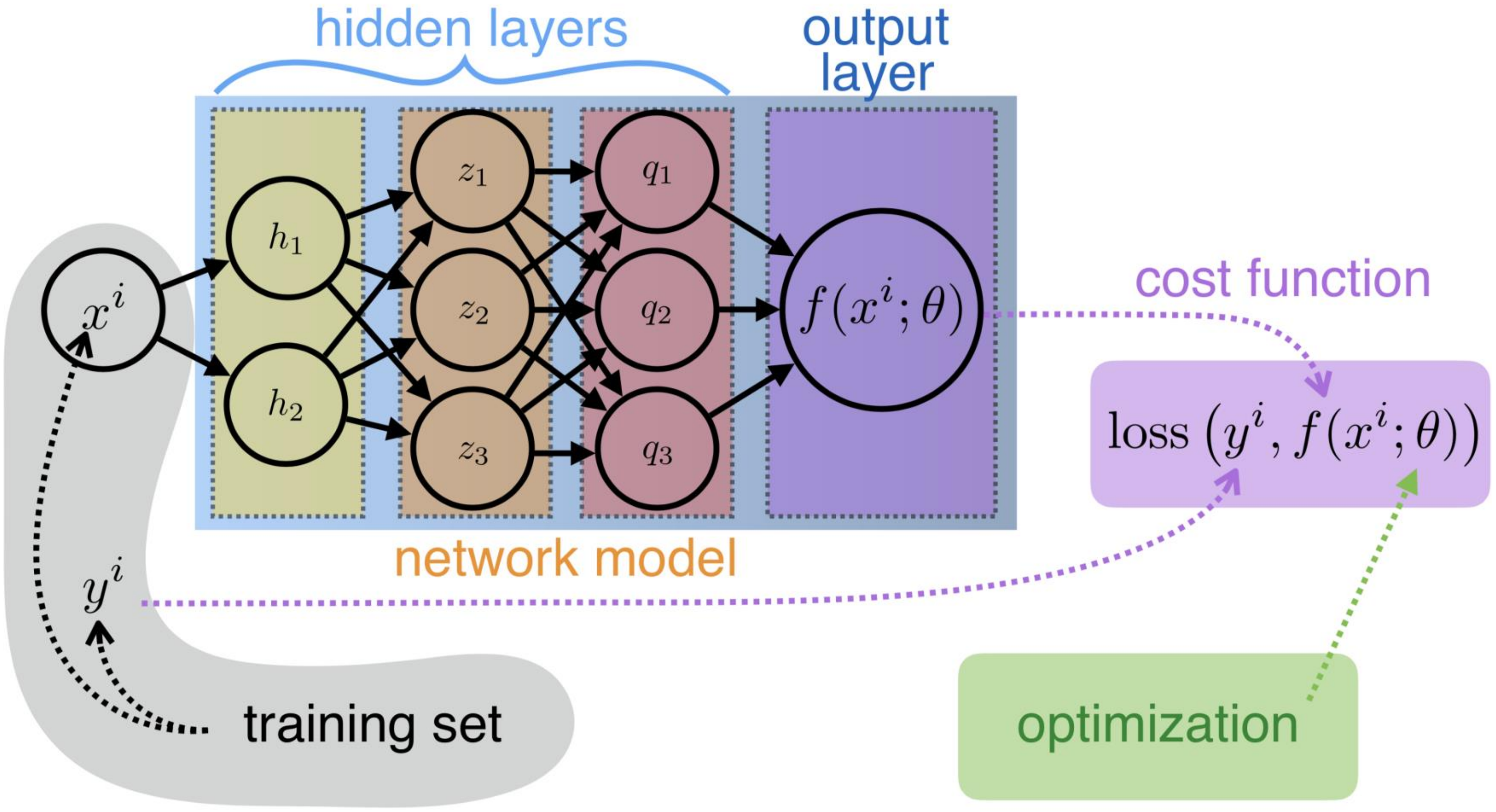


- **feedforward** = information flows from **input** to **output** layer **without feedback loops**

Can you name a NN type with feedback loops ?

- **Deep** for “more than 1 **hidden layer**”

Deploying a Neural Network



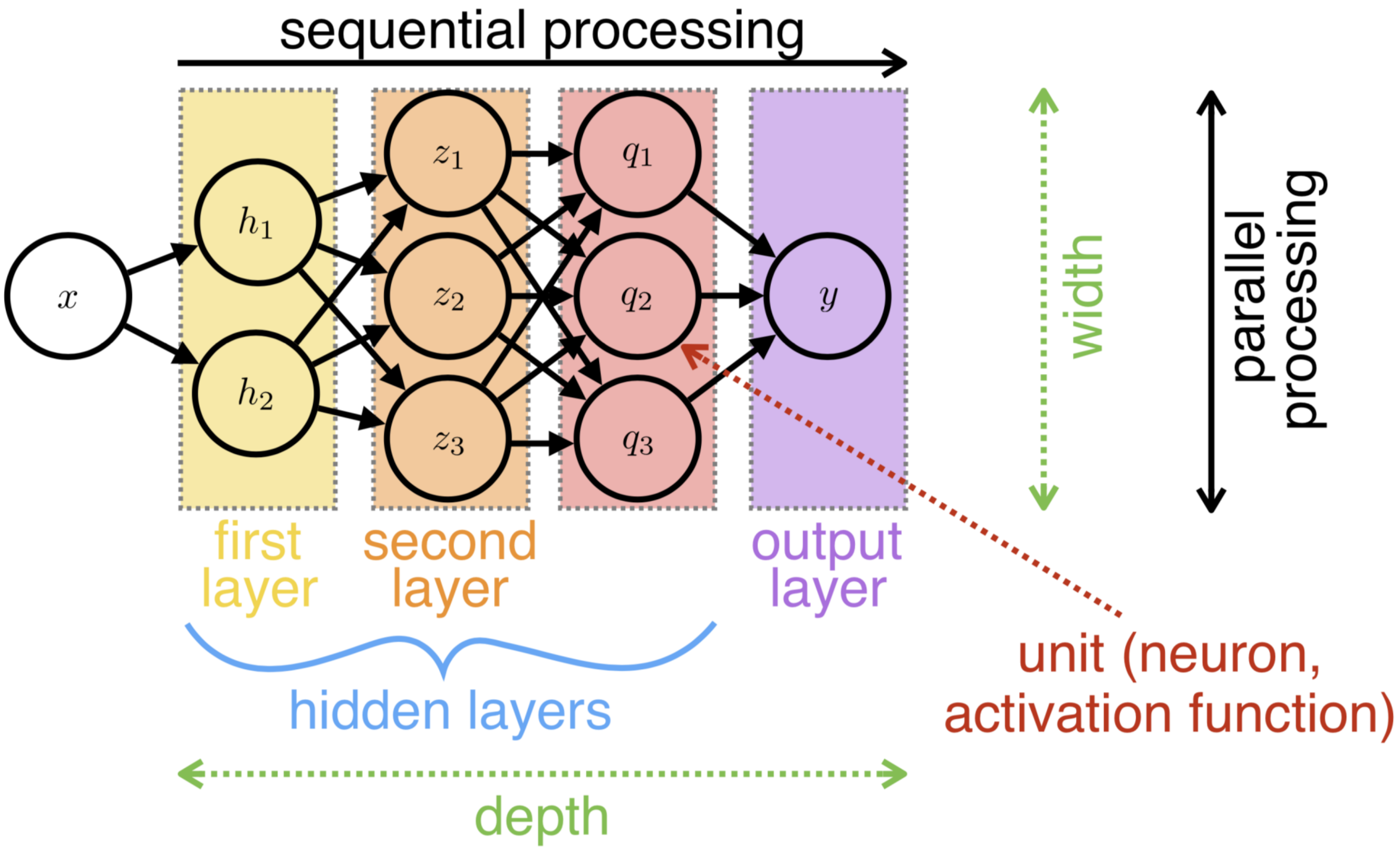
Given a task (in terms of I/O mappings), we need :

1) Network model

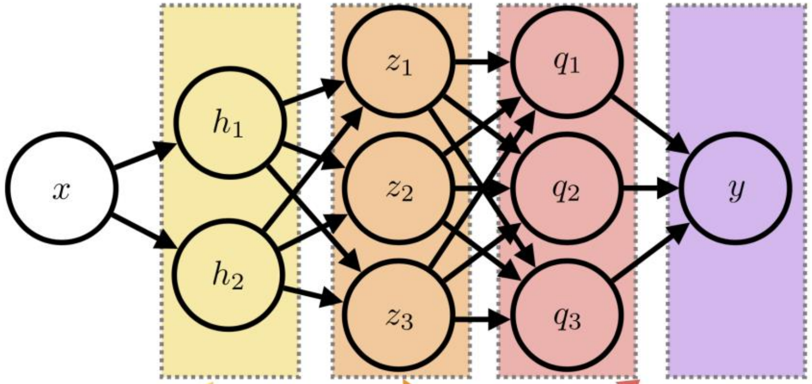
2) Cost function

3) Optimization

Network Model



Activation Functions



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

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

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

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

Hierarchical representation

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

Fully connected

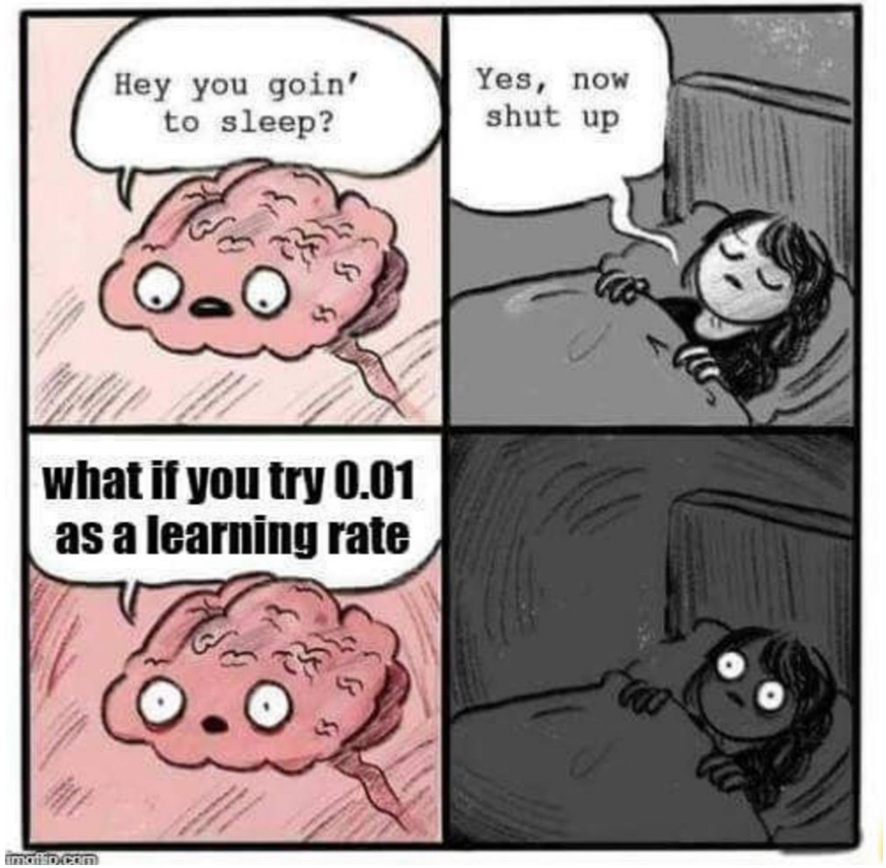
$$\boxed{f_{2,2}(h_1, h_2) = w_1 h_1 + w_2 h_2 + b_{2,2}}$$

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

- Maximum Likelihood
- Training
- Backpropagation
  Activation function
  - Saturating
  - Non-saturating
- Loss functions
- Faster optimizers

reminder

Loss and Cost functions

- Loss function $L\left(\hat{y}^{(i)}, y^{(i)}\right)$, also called error function, measures how different the prediction $\hat{y} = f(x)$ and the desired output y are

- Cost function $J(w, b)$ is the average of the loss function on the entire training set

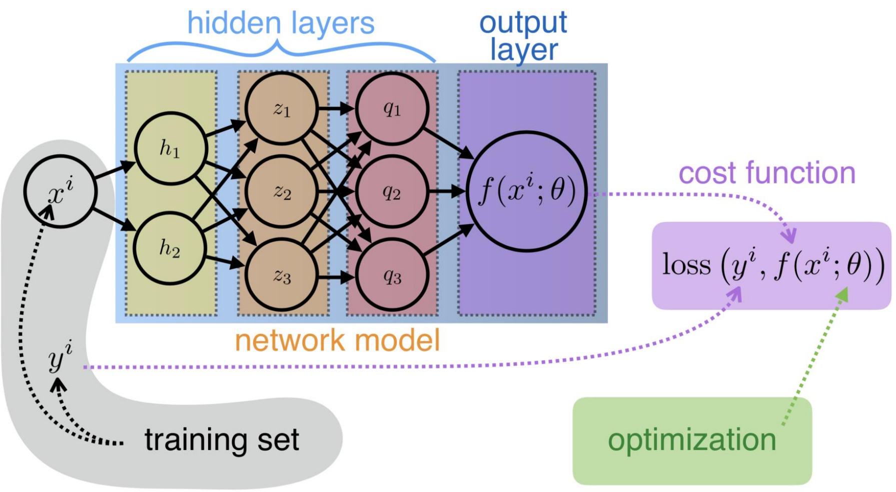
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

- Goal of the optimization is to find the parameters $\theta = (\mathrm{w}, \mathrm{b})$ that minimize the cost function

- Choice of loss function determined by the output representation (regression, classification)

Optimization

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



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

- In addition, we have to set values for hyperparameters

Maximum Likelihood

- Given IID input/output samples :

- Conditional Maximum Likelihood estimate (between model pdf and data pdf):

$$\theta_{\mathrm{ML}} = \arg\max_{\theta} \prod_{i=1}^{m} p_{\mathrm{data}}(y^i | x^i; \theta)$$

- Mathematical tricks :

$$= \arg\max_{\theta} \sum_{i=1}^{m} \log p_{\mathrm{data}}(y^i | x^i; \theta)$$

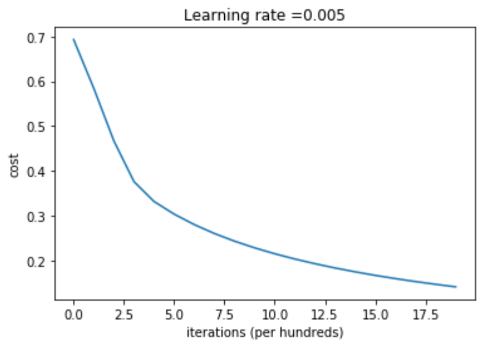
$$\min_{\theta} -E_{x,y \sim \hat{p}_{\mathrm{data}}} [\log p_{\mathrm{model}}(y | x; \theta)]$$

Maximize *the* ***likelihood*** *==* ***Minimize the negative log-likelihood***

Training

- *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**)

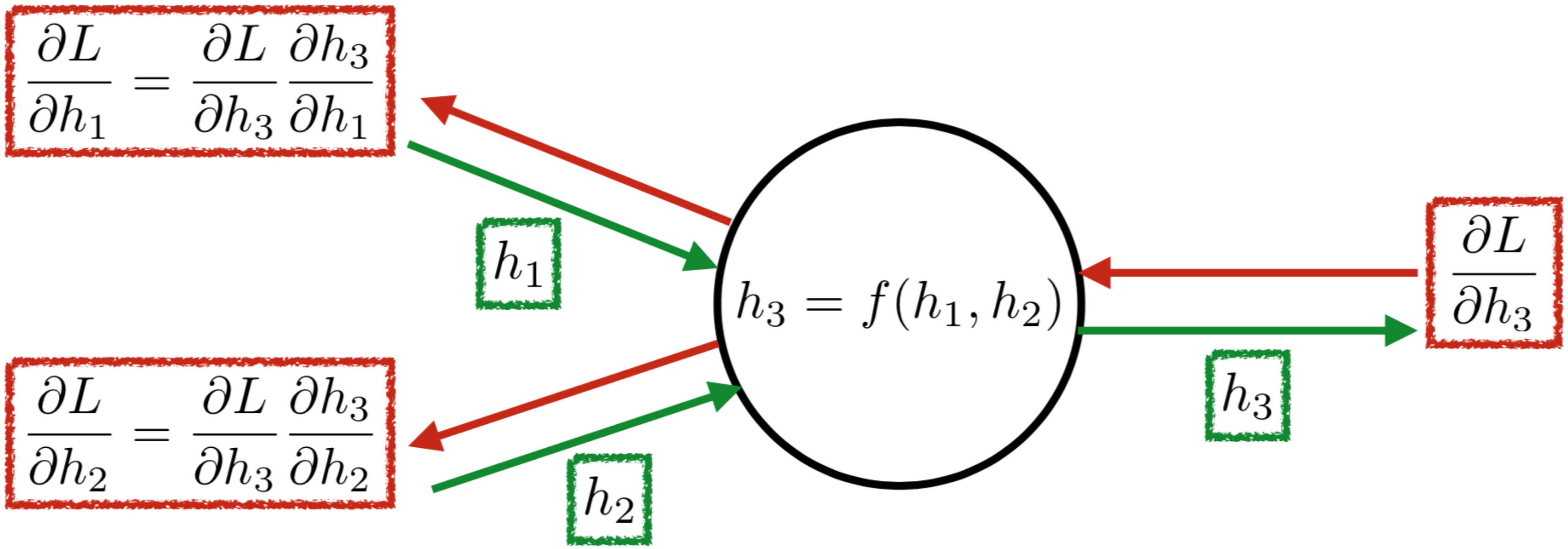
Tweaks the connection weights to reduce the error

learning rate α

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

Backpropagation

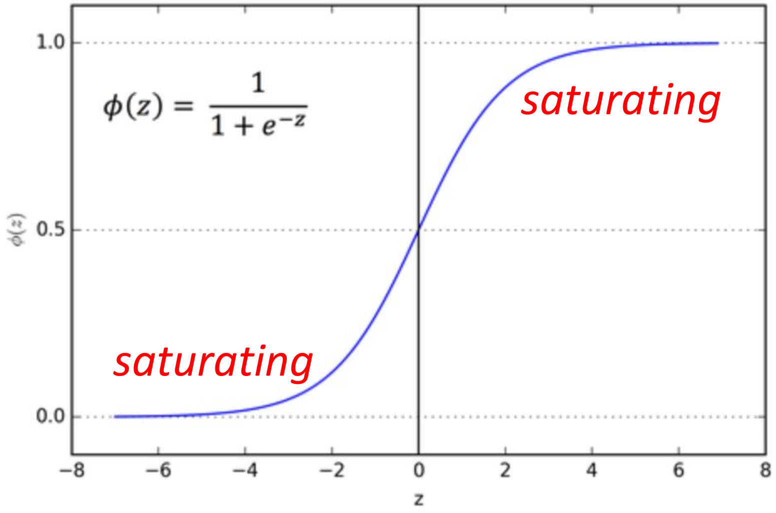
- Efficient implementation of the chain-rule to compute derivatives with respect to network weights



Activation Functions

Sigmoid Function

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



Softmax function used for multiclass classification

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

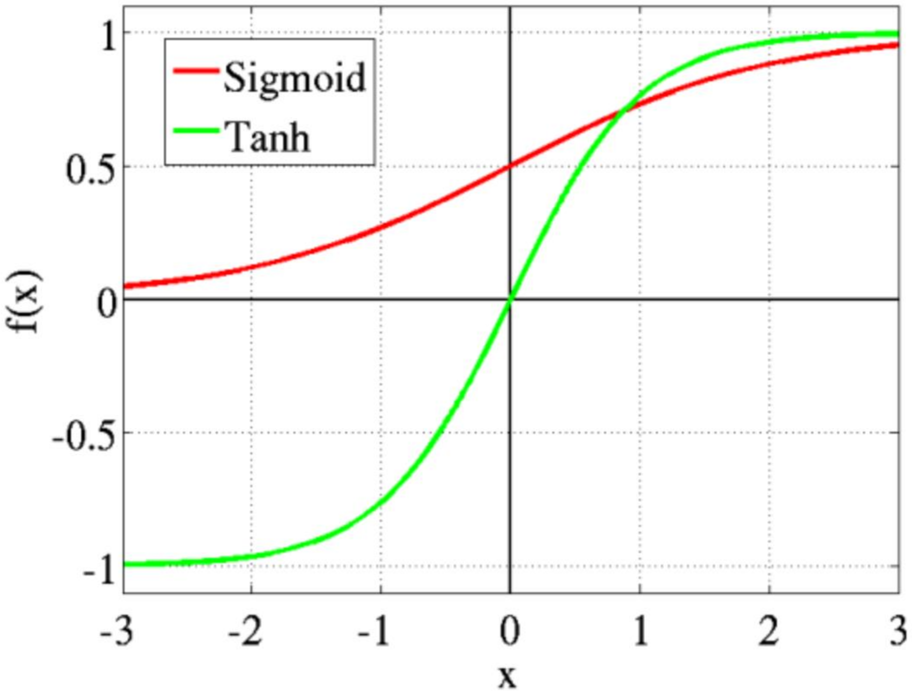
- Used for models where we have to predict the probability as an output

- Differentiable, monotonic but not its derivative

Is this problematic ?

Tanh Function

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



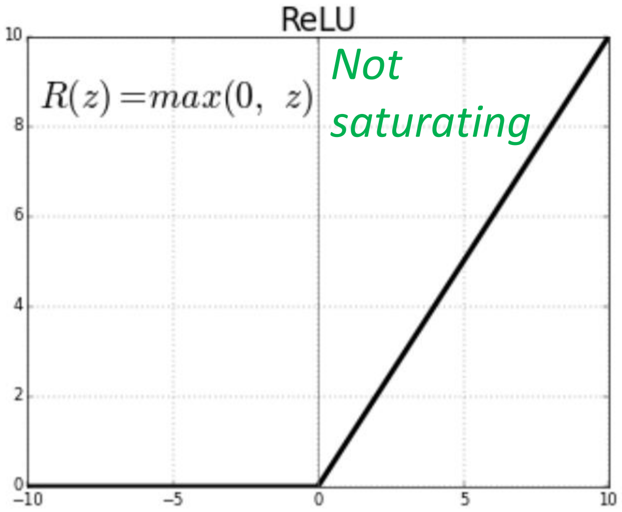
What are the advantages over sigmoid ?

- Used for classification between two classes

- Differentiable, monotonic but not its derivative

ReLU Function

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



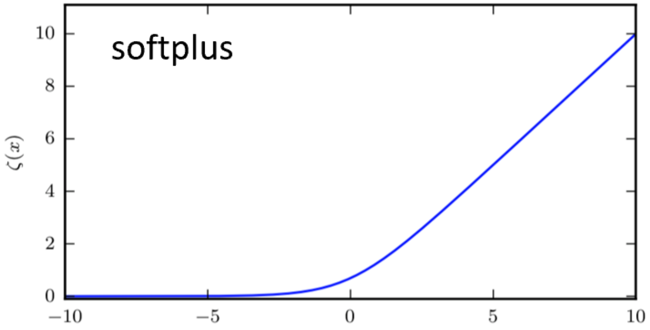
What are the limitations ?

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

- Function and its derivative are both monotonic

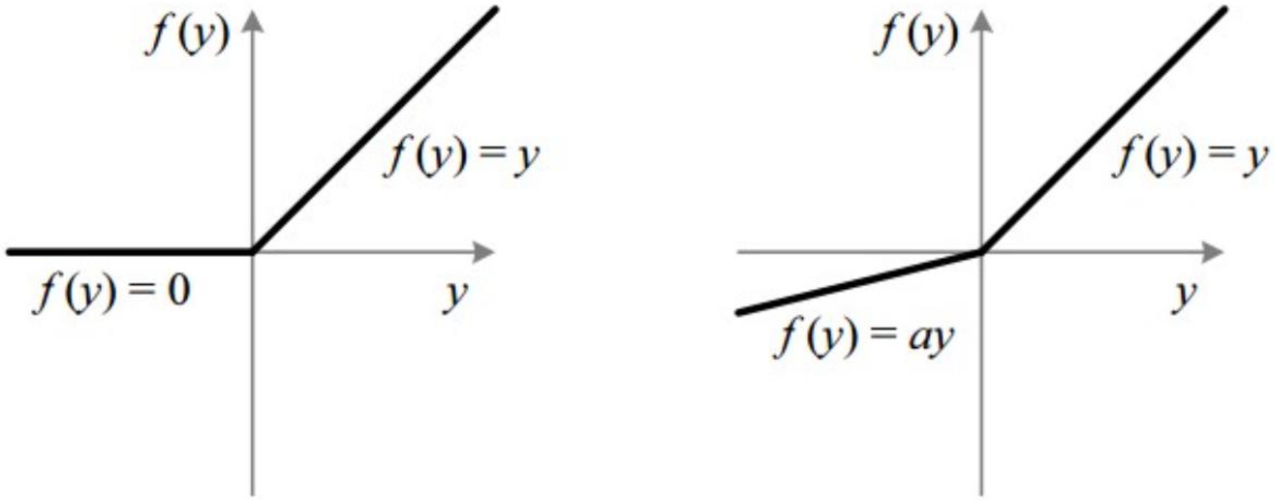
- Softplus : Smooth approximation

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



Leaky ReLU Function

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

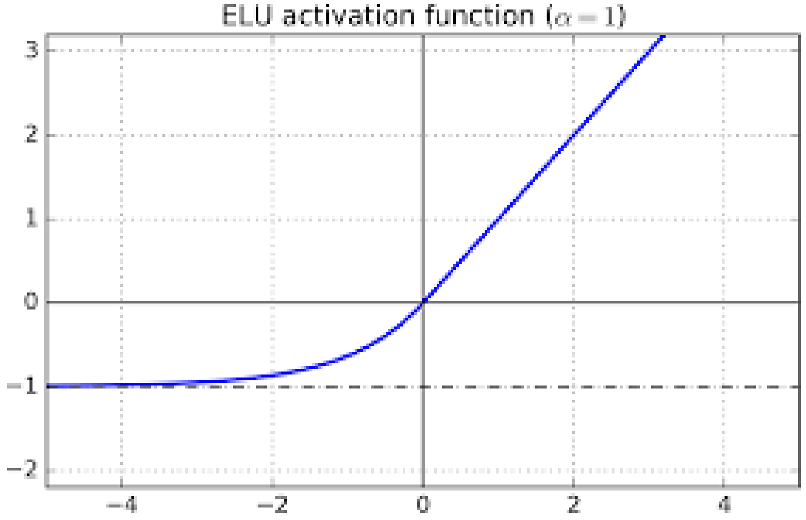


- The leak α = 0.2 seems to lead to better performance than α=0.01

- Alternative is to use randomized ReLU

- Function and its derivative are both monotonic

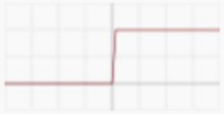
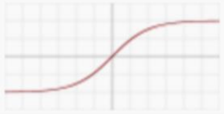
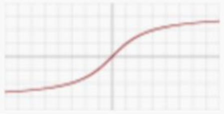
ELU Function



- Takes on negative values when z<0 (solves vanishing gradients problem)
- Non-zero gradient for z<0 (avoids the dying units issue)
- Smooth everywhere, including around z=0 (speed up Gradient Descent)

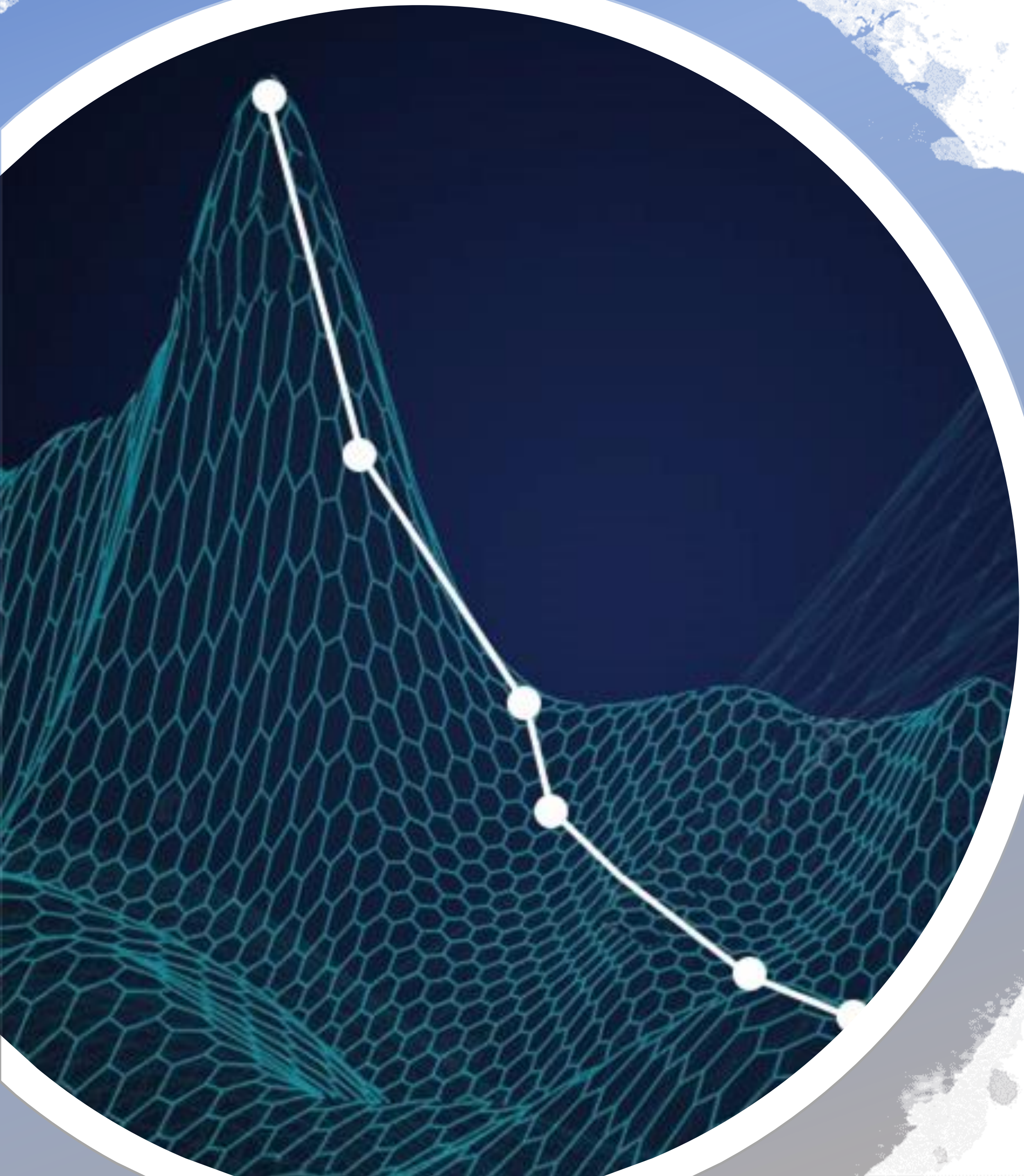
- Main drawback : slower to compute than RELU
  - During training : compensated by faster convergence rate
  - During testing : slower

Activation Functions 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		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Activation Functions: In practice (Keras)

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



Loss functions

Regression Loss Functions

- 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

$$\hat{y} = W^\top h + b$$

$$p(y|\hat{y}) = \mathrm{N}(y;\hat{y})$$

$$\boxed{L_2(\hat{y}, y) = -\log p(y|\hat{y}) = \sum_{i=0}^{m} \left(y^i - \hat{y}^i\right)^2}$$

Regression Loss Functions

- Mean Squared Logarithmic Error Loss (MSLE)
  - 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
  - First calculate the natural log of each predicted values, then MSE
- Mean Absolute Error Loss (MAE)
  - Target variable may be mostly Gaussian, but with outliers

Binary Classification Loss Functions

- Binary Cross-Entropy Loss

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

$$\hat{y} = \sigma(w^\top h + b)$$

$$p(y|\hat{y}) = \hat{y}^{y}(1-\hat{y})^{(1-y)}$$

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

Binary Classification Loss Functions

- Hinge Loss
  - Primarily developed for use with SVM models
  - Binary classification where the target values are in the set {-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

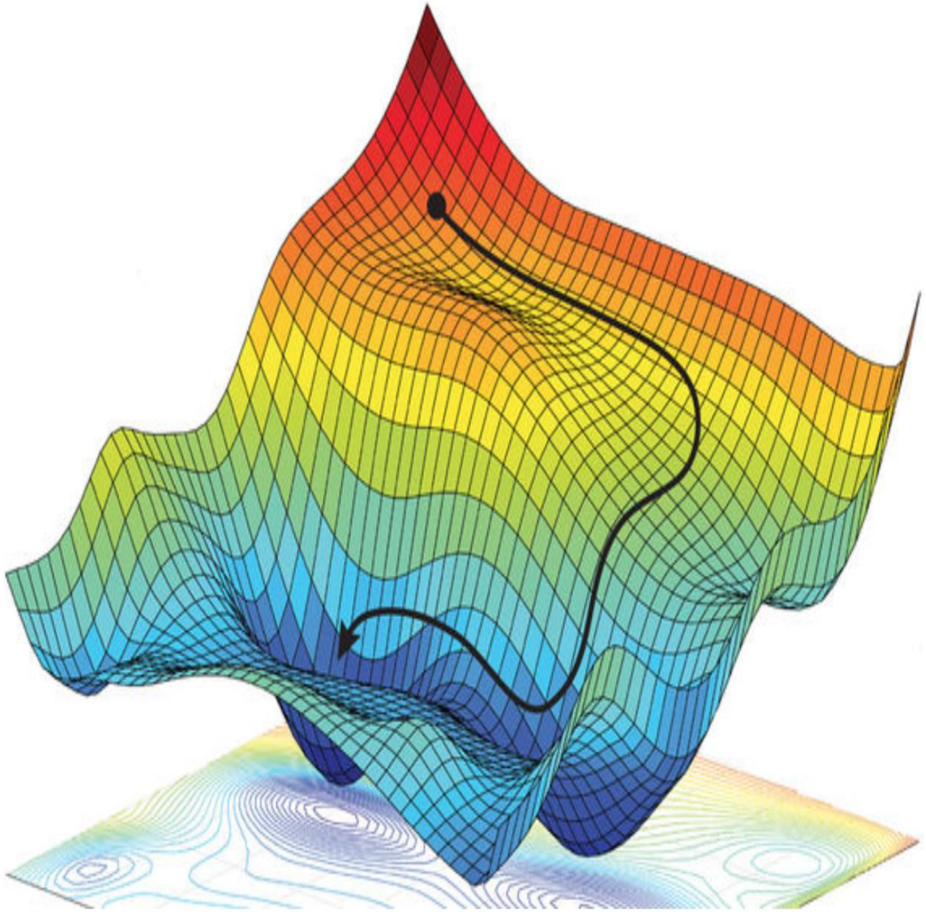
- Multi-Class Cross-Entropy Loss
  - Target set {0,1,2,…,n} : need for one-hot encoding
  - Generalization of binary cross-entropy loss to n classes
- Sparse Multiclass Cross-Entropy Loss
  - No need to have the target variable be one-hot encoded
    - Tackles the problem of one-hot encoding when too 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			
Geometry of output layer			
Activation of output layer			
Loss function			



Faster Optimizers than Gradient Descent

- Momentum optimization
- RMSprop
- Adaptative Moment (Adam)

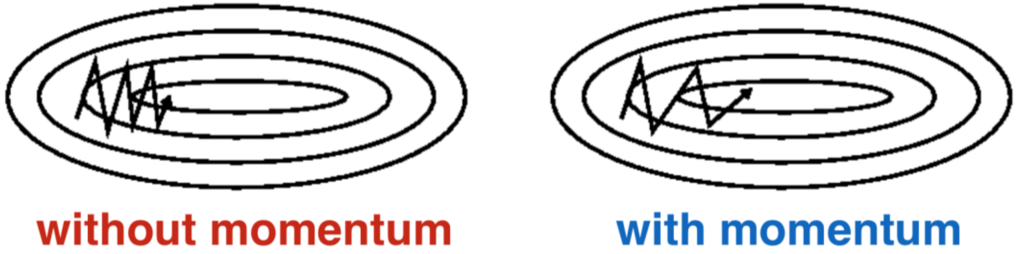
Momentum Optimization

- Momentum optimization takes past gradients into account

- Hyperparameter momentum β accelerates search in direction of minima (update rule) :

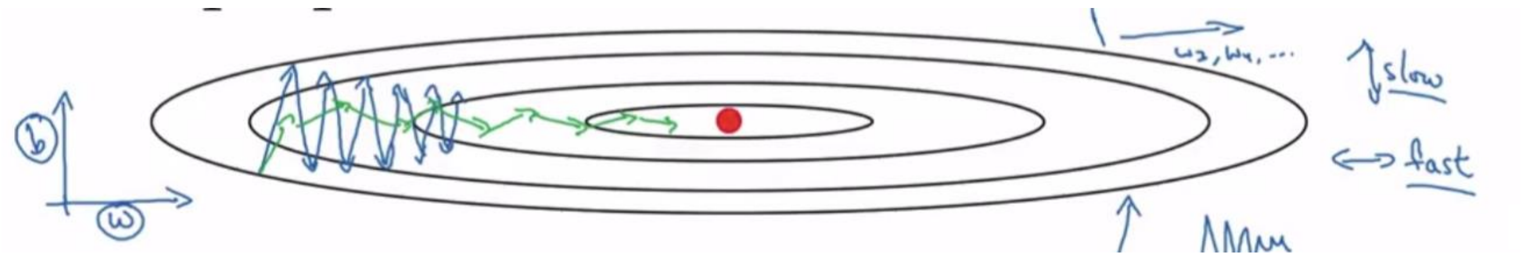
$$\boxed{\begin{gathered}\mathbf{m}_{t+1} \leftarrow \beta \mathbf{m}_t - \alpha \nabla_\theta J(\theta_t) \\ \theta_{t+1} \leftarrow \theta_t + \mathbf{m}_{t+1}\end{gathered}}$$

- 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

- Similar to SGD+β, difference in how the gradients are computed
- impedes search in direction of oscillations (vertical direction) : allows to increase α



$$v_{dw} = \beta \cdot v_{dw} + (1-\beta) \cdot dw$$

$$v_{db} = \beta \cdot v_{dw} + (1-\beta) \cdot db$$

$$W = W - \alpha \cdot v_{dw}$$

$$b = b - \alpha \cdot v_{db}$$

SGD+β

$$v_{dw} = \beta \cdot v_{dw} + (1-\beta) \cdot dw^2$$

$$v_{db} = \beta \cdot v_{dw} + (1-\beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

RMSProp

Adam Algorithm

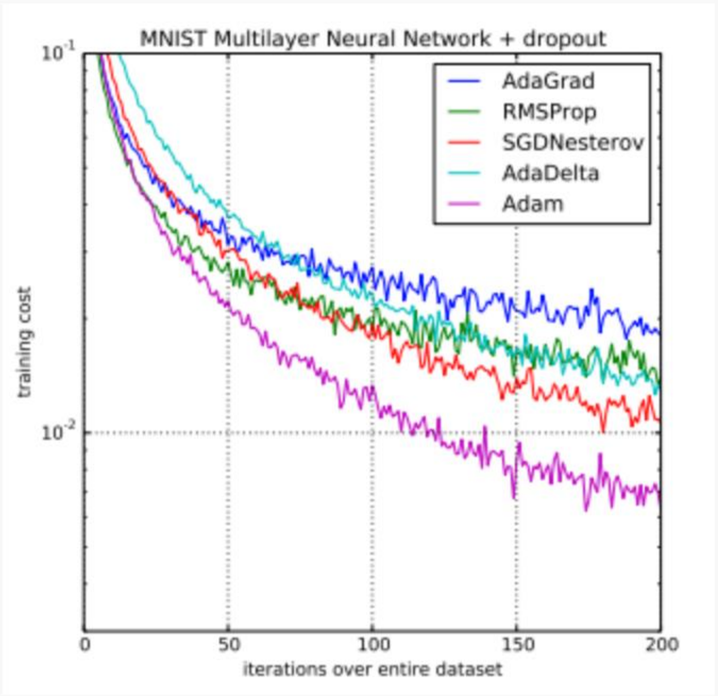
- Combine ideas from RMSProp (α increase) and Momentum (acceleration), with parameters :
  - Momentum decay : β_1 for dw (*usually 0.9*)
  - Scaling decay : β_2 for dw^2 (*usually 0.99*)
  - Smoothing term : ε (*usually* 1^{-10})

$$\boxed{\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned}}$$

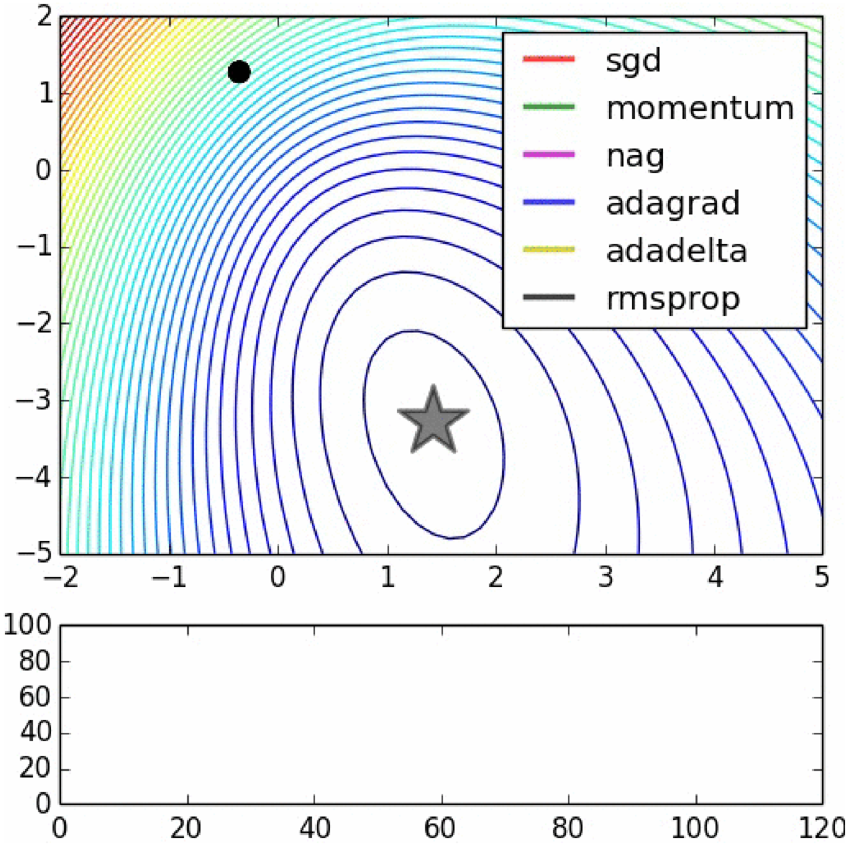
$$\boxed{\begin{aligned} \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \end{aligned}}$$

$$\boxed{w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}}$$

Comparison



Comparison of Adam to Other Optimization Algorithms Training a Multilayer Perceptron
Taken from Adam: A Method for Stochastic Optimization, 2015.



45

Optimizer: In practice (Keras)

- optimizer=‘SGD’
- optimizer=‘RMSprop’
- optimizer=‘Adam’
- …

Learning Rate Scheduling

- Start with high learning rate and reduce it once it stops making fast progress

- Good solution reached faster than with the optimal constant learning rate

- Learning schedules examples:
  - Performance scheduling
    - Measure the validation error every N steps and reduce the learning rate when the error stops dropping
  - Exponential scheduling
    - Set the learning rate to a function of the iteration number t

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

Neural Networks as an alternative to other ML algorithms

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'])
```

Exercise

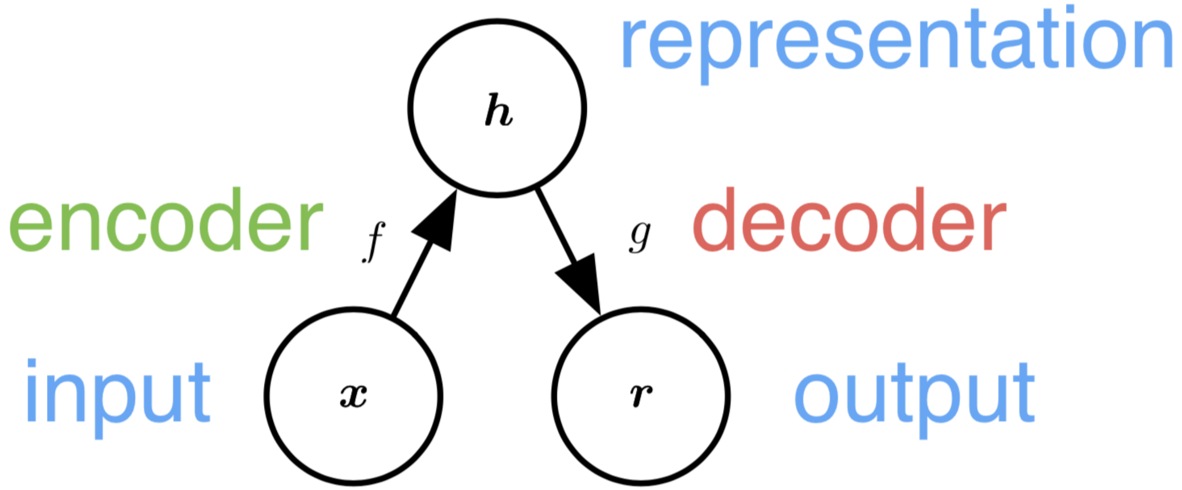
What do these codes do ?

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)

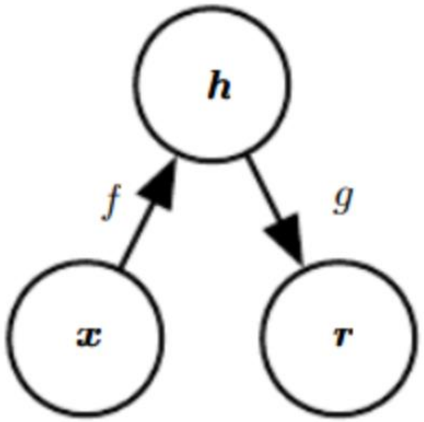
- Network that replicates the input
  - Internally, it builds a representation of the input
  - Network before the internal representation : **encoder**
  - Network following the representation : **decoder**



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

- The network is an unbiased estimator that is minimizing the variance between two distributions

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)

Two-Minute Papers

Quiz

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

Room : CONTI6128