Introduction to Deep Learning

Lecture 2

Backpropagation and Modularity

Maria Vakalopoulou & Stergios Christodoulidis

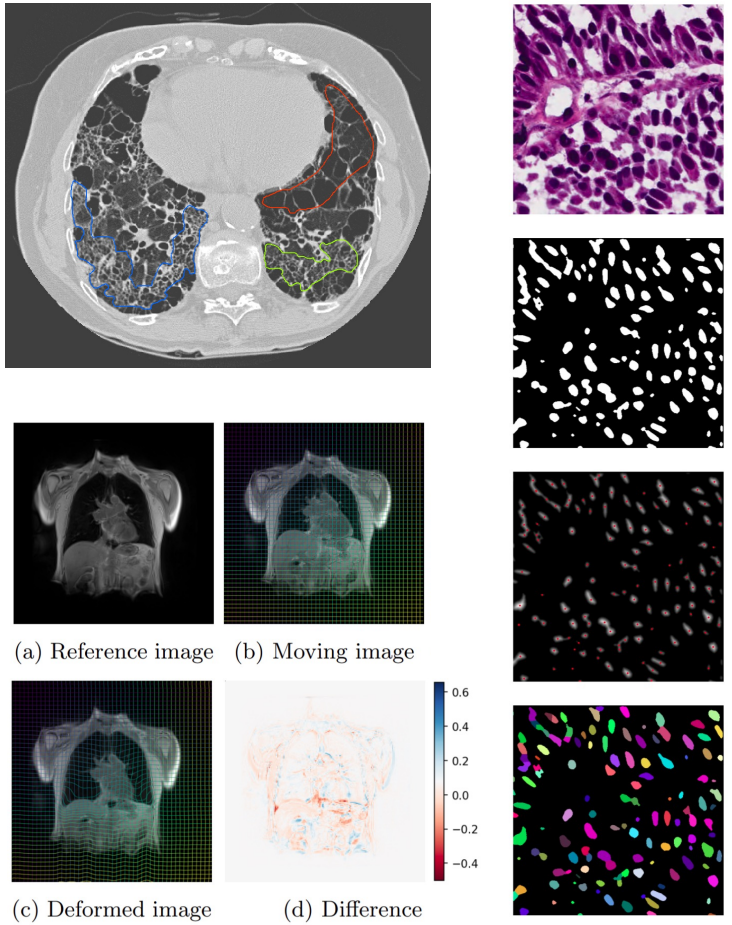
MICS Laboratory

CentraleSupélec
Université Paris-Saclay

Wednesday, November 05, 2021

Short Introduction

- PhD degree in University of Bern
  - Medical Image Analysis
  - Diagnosis Support Systems
- Postdoctoral Researcher at GR
  - AI for Cancer
  - Treatment Decision Support
  - Understanding of biological
- Assistant Prof at CS MICS Lab
  - Machine Learning
  - Healthcare Applications



(a) Reference image (b) Moving image

(c) Deformed image (d) Difference

Last Lecture

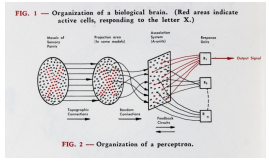
Motivation

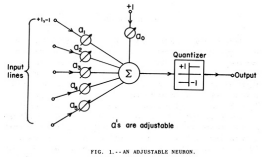
A person riding a motorcycle on a dirt road.

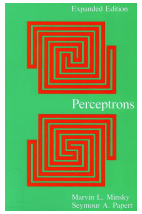
Two dogs play in the grass.

A skateboarder does a trick on a ramp.

Machine Learning Early Years







Perceptrons, Rosenblatt

Adaline, Widrow and Hoff

Perceptrons, Minsky and Papert

Backpropagation, Linnainmaa

Backpropagation, Werbos

Backpropagation, Rumelhart, Hinton and Williams

LSTM, Hochreiter and Schmidhuber

OCR, LeCun, Bottou, Bengio and Haffner

Deep Learning, Hinton, Osindero, Teh

Imagenet, Deng et al.

Alexnet, LeCun, Bottou, Bengio and Haffner

Resnet (154 layers), MSRA

GO, Deepmind

1958

1960

1969

1970

1974

1986

1997

1998

2006

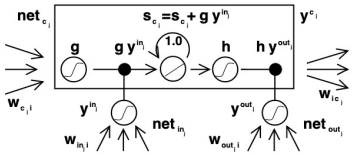
2009

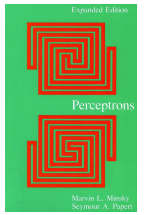
2013

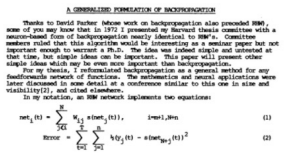
2015

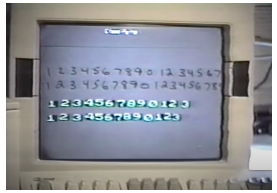
today

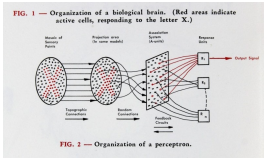
The AI winter

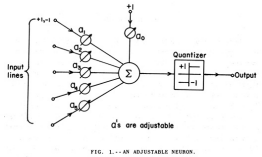












Perceptrons, Rosenblatt

Adaline, Widrow and Hoff

Perceptrons, Minsky and Papert

Backpropagation, Linnainmaa

Backpropagation, Werbos

Backpropagation, Rumelhart, Hinton and Williams

LSTM, Hochreiter and Schmidhuber

OCR, LeCun, Bottou, Bengio and Haffner

Deep Learning, Hinton, Osindero, Teh

Imagenet, Deng et al.

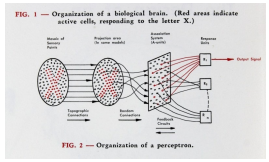
Alexnet, LeCun, Bottou, Bengio and Haffner

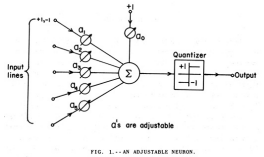
Resnet (154 layers), MSRA

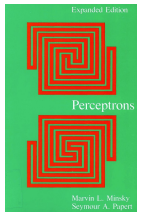
GO, Deepmind

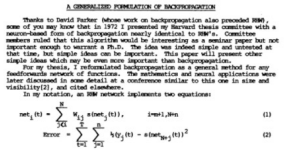
1958 1960 1969 1970 1974 1986 1997 1998 2006 2009 2013 2015 today

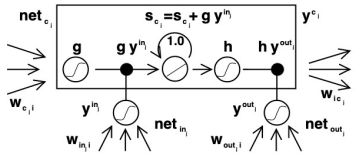
Deep Learning Golden Era

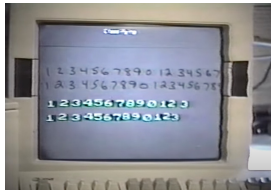


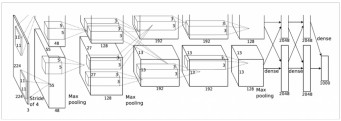












Perceptrons, Rosenblatt 1958

Adaline, Widrow and Hoff 1960

Perceptrons, Minsky and Papert 1969

Backpropagation, Linnainmaa 1970

Backpropagation, Werbos 1974

Backpropagation, Rumelhart, Hinton and Williams 1986

LSTM, Hochreiter and Schmidhuber 1997

OCR, LeCun, Bottou, Bengio and Haffner 1998

Deep Learning, Hinton, Osindero, Teh 2006

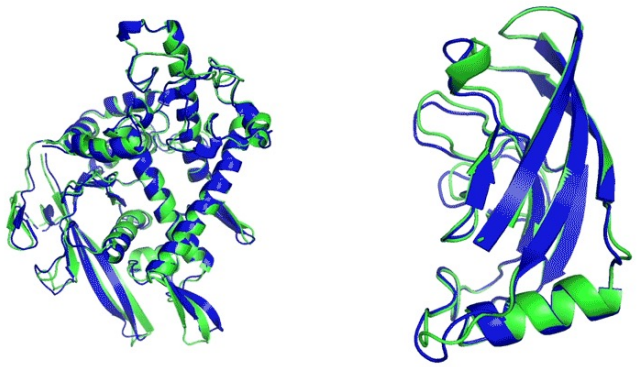
Imagenet, Deng et al. 2009

Alexnet, LeCun, Bottou, Bengio and Haffner 2013

Resnet (154 layers), MSRA 2015

GO, Deepmind 2017

Latest Advances

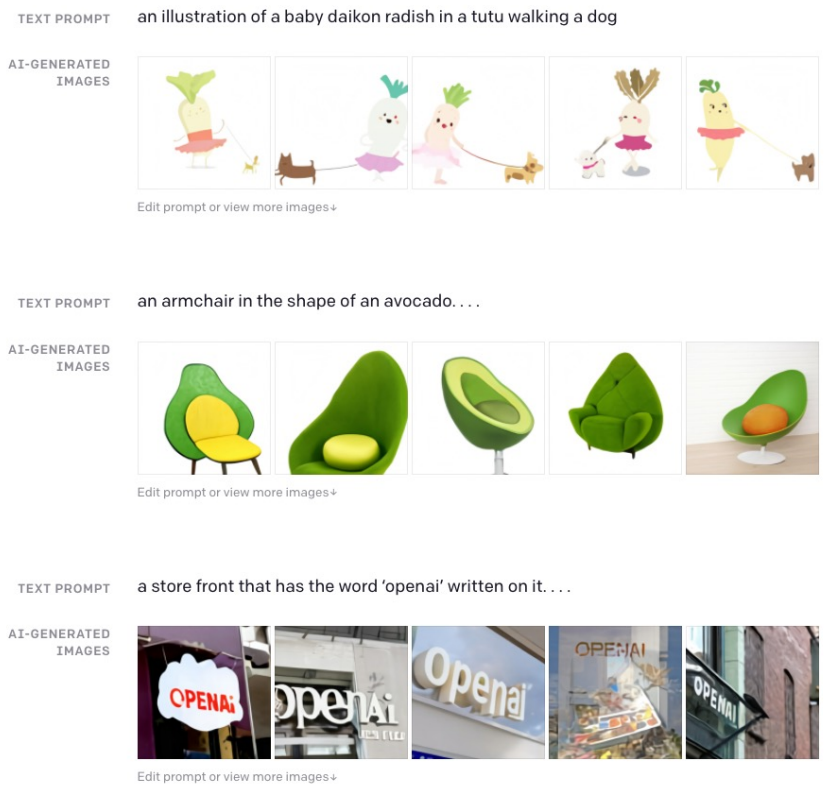


T1037 / 6vr4
90.7 GDT
(RNA polymerase domain)

T1049 / 6y4f
93.3 GDT
(adhesin tip)

Experimental result
Computational prediction

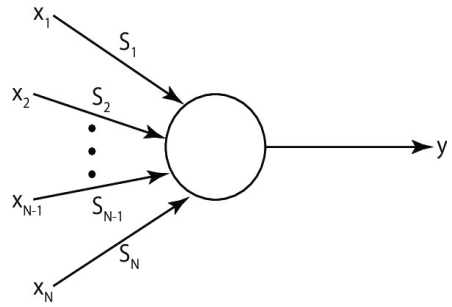
(AlphaFold, DeepMind, 2020)

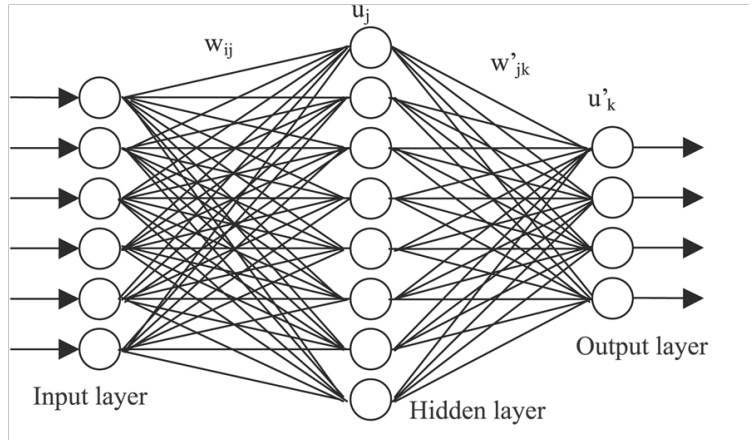


(DALL-E, OpenAI, 2021)

Perceptrons

- 1 perceptron = 1 decision (Neuron)
- What about multiple decisions?
  - E.g. digit classification
- Stacks as many outputs as the possible outcomes into a layer
  - Neural Networks
- Use one layer as input to the next layer
  - Add nonlinearities between layers
  - Multi-layer perceptron (MLP)



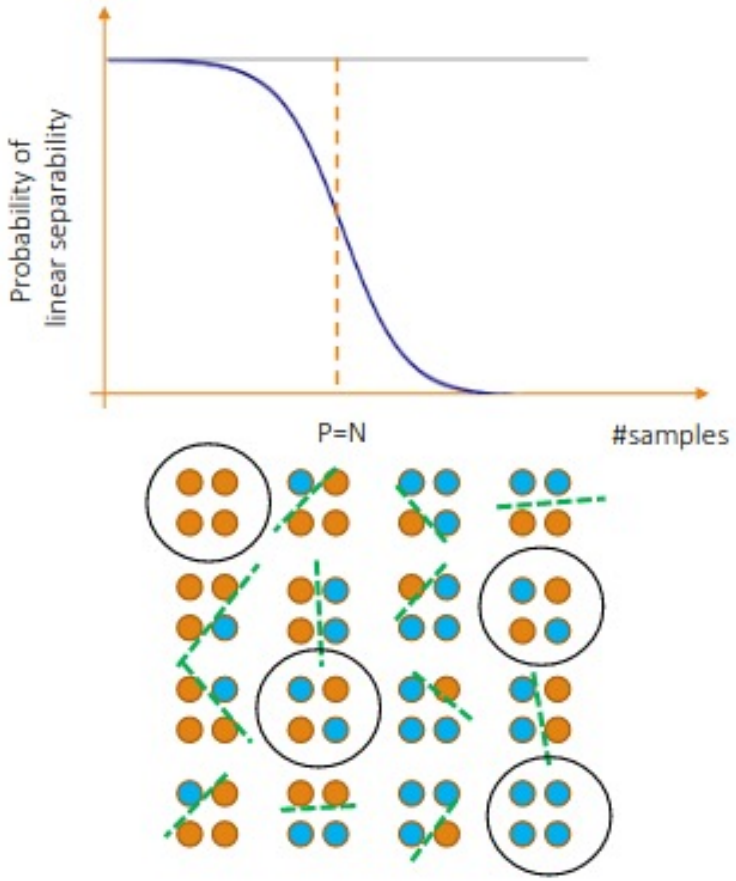


Linear Separability

- Given a set of input data:

$$X = \{x^{(1)}, x^{(2)}, \dots, x^{(N)}\}, x \in \mathcal{R}^P$$

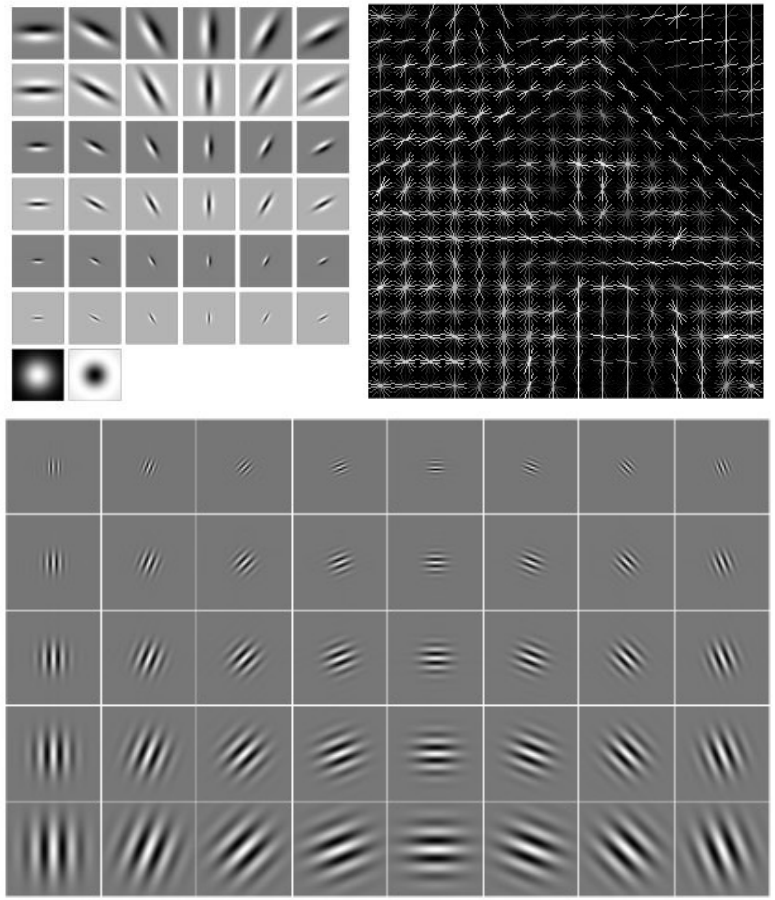
- There are 2^N dichotomies
- Only about P are linearly separable
- With N $>$ P the probability X is linear separable converges to 0 very fast.
- The chances that a dichotomy is linearly separable is very small



Non-Linear Models

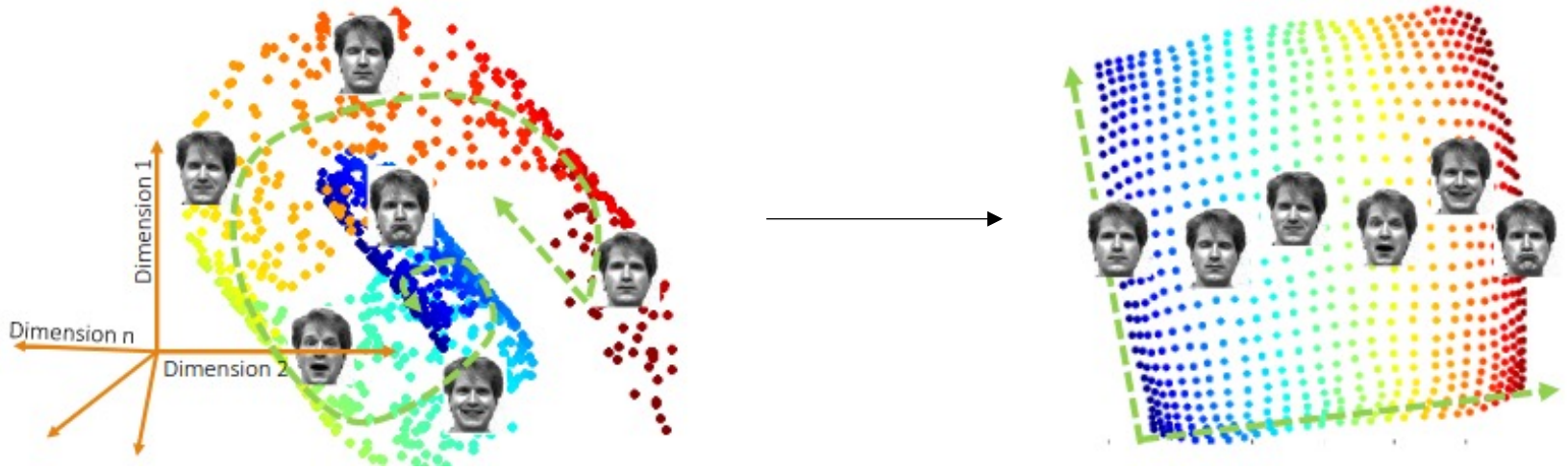
- Most data distributions and tasks are non-linear
- A linear assumption is often convenient, but not necessarily truthful
- **Problem:** How to get non-linear machines without too much effort?

- **Solution:** Make features non-linear
- What is a good non-linear feature?
  - Non-linear kernels, e.g., polynomial. RBF, etc.
  - Explicit design of features (SIFT, HOG)



Manifolds

- Raw data live in huge dimensionalities
- But effectively lie in lower dimensional manifolds → Euclidean properties in local regions
- Can we discover this manifold to embed our data on?



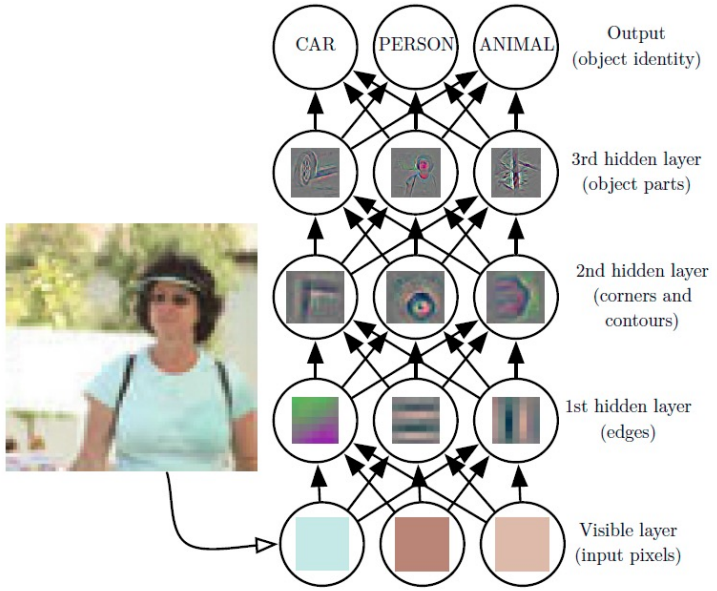
Deep Learning

- A set of parametric, nonlinear and hierarchical representation learning models, which are optimized with stochastic gradient descent to encode domain knowledge, i.e., domain invariances, stationarity.

- Given a training corpus the task is to find the optimal parameters for the model that fit to the data.

- The model is a set of nested parametric linear/non-linear functions:

$$f\left(X;\theta_{1,\ldots,n}\right) = h_n(h_{n-1}(\ \ldots h_1(X;\theta_1);\theta_{n-1});\theta_{\mathrm{n}})$$



(Goodfellow et al., 2016)

Today's Lecture

Today's Lecture

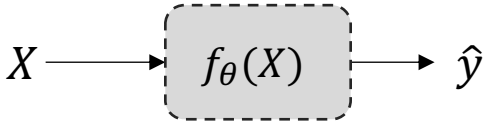
- Gradient Descent
  - E.g., Simple Linear regression
  - E.g., Linear Neuron model
  - Chain Rule and Backpropagation
  - E.g., Multi-layer Perceptron

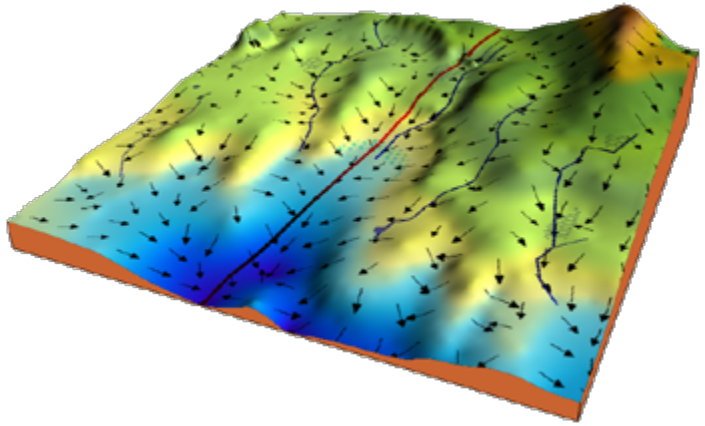
- Modularity in Deep Learning
  - Combining modules
  - Popular Deep Learning modules

Gradient Descent

Gradient Descent

- Optimization algorithm
  - Minimize a function by tuning the model parameters
  - Moving in the direction of the steepest negative gradient $(\frac{d\mathcal{L}}{d\theta})$
- Loss/Cost Function $(\mathcal{L})$
  - The function that is minimized in the context of ML
- Parameter optimization (θ)
  - We start at the top of a hill
  - We iteratively take steps towards the negative gradient
  - We stop when we can no longer move downhill
- Learning Rate (λ)
  - How large will our steps be





$$\theta_{n+1} = \theta_n - \lambda \cdot \frac{d\mathcal{L}}{d\theta}$$

Chain Rule Refresher

- Differentiation in the case of nested functions

$$\mathrm{F}(x) = f(g(x))$$

- We can use the chain rule to calculate the gradient with respect to a variable

$$F'(x) = f'\big(g(x)\big)g'(x) \text{ (Lagrange's notation)}$$

$$\frac{dF}{dx} = \frac{df}{dg}\frac{dg}{dx} \text{ (Leibniz's notation)}$$

Simple Linear Regression Model

- Model Function

$$f(x) = mx + b$$

- Data

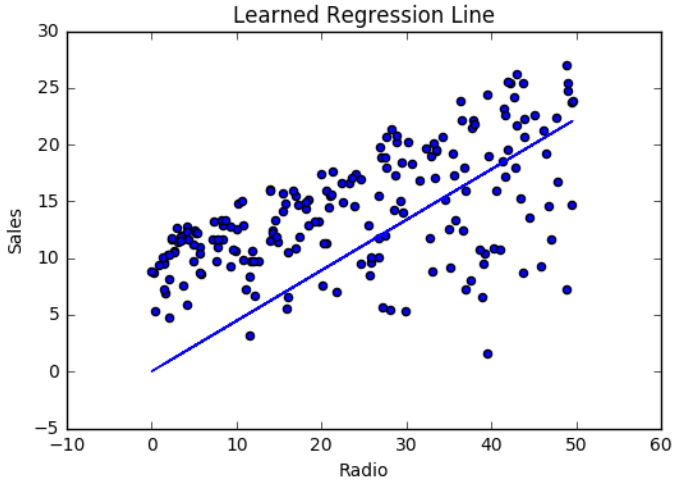
$$\{x^{(i)}, y^{(i)}\}_{i=1}^{n}: \; x \in \mathcal{R}, y \in \mathcal{R}$$

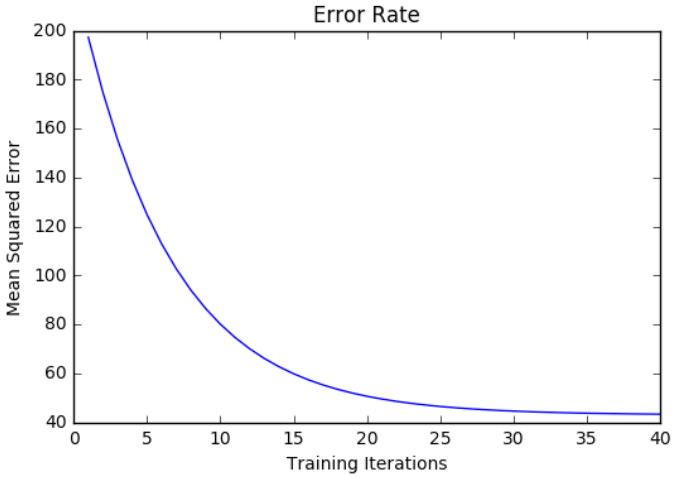
- Loss function (MSE)

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(f(x^{(i)}) - y^{(i)})^2$$

- Gradient Descent

$$\theta_{n+1} = \theta_n - \lambda \frac{d\mathcal{L}}{d\theta}, \qquad \theta = \{m, b\}$$





Linear Neuron Model

- Model Function

$$f(x) = R(\boldsymbol{X}, \boldsymbol{S}) = x_1 S_1 + x_2 S_2 + \ \ldots + \ x_N S_N = \boldsymbol{x}\boldsymbol{S}^T$$

- Data

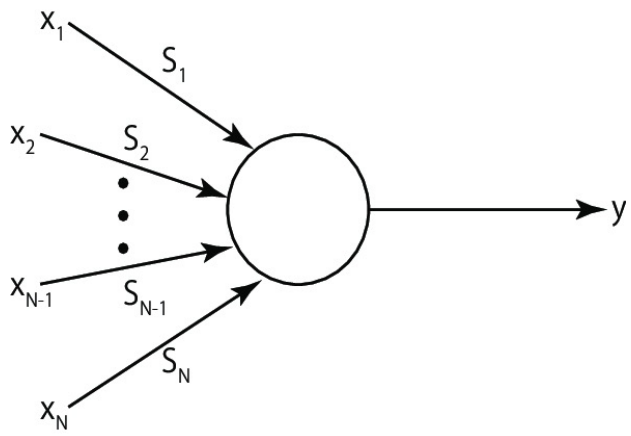
$$\{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^{N} \colon \ x \in \ \mathcal{R}^d, y \in \ \mathcal{R}$$

- Loss function (MSE)

$$\mathcal{L} = \ \frac{1}{N}\sum_{i=1}^{N}(f\left(\boldsymbol{x}^{(i)}\right) - y^{(i)})^2$$

- Gradient Descent

$$\theta_{n+1} = \theta_n - \lambda \frac{d\mathcal{L}}{d\theta}, \qquad \theta = \{S_1, S_2, \ldots, S_n\}$$

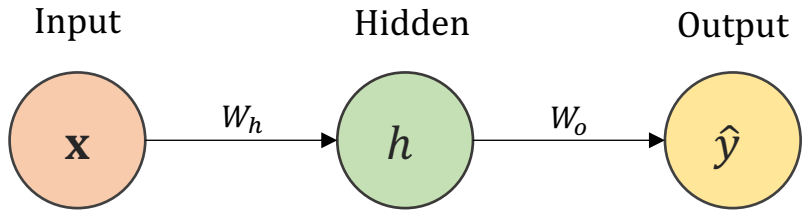


Multi-layer Perceptron Model

- Model Function

$$g_{W_h}(x) = \boldsymbol{h} = \boldsymbol{x}\,\boldsymbol{W_h} \text{ (linear activation function)}$$

$$f_{W_o}(h) = \hat{y} = \boldsymbol{h}\,\boldsymbol{W_o} \text{ (linear activation function)}$$

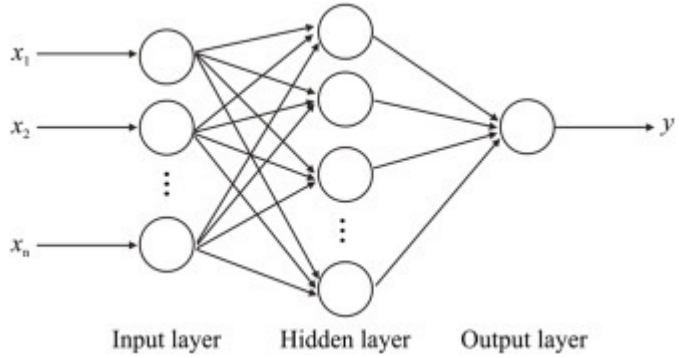


- Data

$$x \in \mathcal{R}^d, y \in \mathcal{R}$$

- Loss function

$$\mathcal{L} = \frac{1}{2}(\hat{y} - y)^2$$



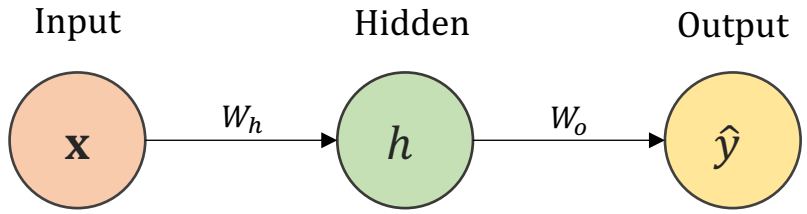
- Gradient Descent

$$\theta_{n+1} = \theta_n - \lambda \frac{d\mathcal{L}}{d\theta}, \qquad \theta = \{W_h, W_o\}$$

Applying the Chain Rule to Multi-layer Perceptron

- Multiple Layers of Computation→ Nested functions

$$min\ \mathcal{L}\left(f_{W_o}\left(g_{W_h}(x)\right)\right)$$



- We can use the chain rule to calculate the gradient:

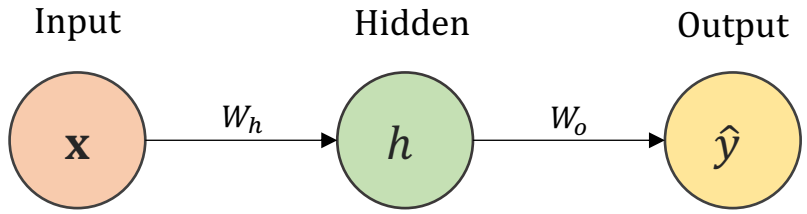
$$\frac{d\mathcal{L}}{dW_o} = \frac{d\mathcal{L}}{df} \cdot \frac{df}{dW_o} = \frac{d\mathcal{L}}{d\hat{y}} \cdot \frac{d\hat{y}}{dW_o}$$

$$\frac{d\mathcal{L}}{dW_h} = \frac{d\mathcal{L}}{df} \cdot \frac{df}{dg} \cdot \frac{dg}{dW_h} = \frac{d\mathcal{L}}{d\hat{y}} \cdot \frac{d\hat{y}}{dh} \cdot \frac{dh}{dW_h}$$

Applying the Chain Rule to Multilayer Perceptron

- Let's calculate the gradients:

Function	Gradient
$\mathcal{L} = \frac{1}{2}[\hat{y} - y]^2$	$\frac{d\mathcal{L}}{d\hat{y}} = (\hat{y} - y)$
$\hat{y} = \boldsymbol{hW_o}$	$\frac{d\hat{y}}{dW_o} = \boldsymbol{h}$, $\frac{d\hat{y}}{dh} = \boldsymbol{W_o}$
$\boldsymbol{h} = \boldsymbol{XW_h}$	$\frac{dh}{dW_h} = \boldsymbol{X}$

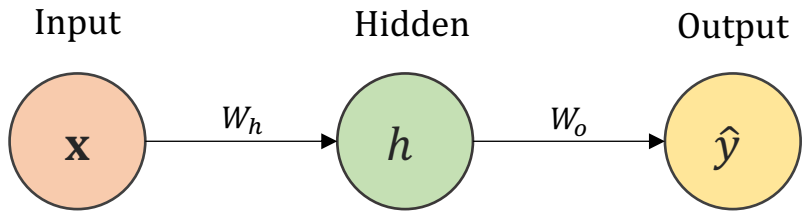


Applying the Chain Rule to Multilayer Perceptron

- Let's calculate the gradients:

$$\frac{d\mathcal{L}}{dW_o} = \frac{d\mathcal{L}}{d\hat{y}} \cdot \frac{d\hat{y}}{dW_o} = (\hat{y} - y)\boldsymbol{h}$$

$$\frac{d\mathcal{L}}{dW_h} = \frac{d\mathcal{L}}{d\hat{y}} \cdot \frac{d\hat{y}}{dh} \cdot \frac{dh}{dW_h} = (\hat{y} - y)\boldsymbol{W_o X}$$



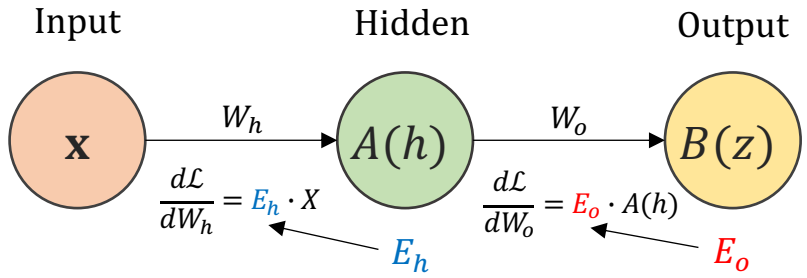
- Gradient Updates:

$$W_o^{(i+1)} = W_o^{(i)} - \lambda(\hat{y} - y)\boldsymbol{h}$$

$$W_h^{(i+1)} = W_h^{(i)} - \lambda(\hat{y} - y)\boldsymbol{W_o X}$$

General Case

- In the general case:
  - Different activation functions are applied in each output
  - Optimization of parameters is performed again using the chain rule
  - The gradient of each layer is the backpropagated error times it's input.



Function	Gradient
Output Layer Error	$E_o = \frac{d\mathcal{L}}{dB}\frac{dB}{dz} = (B(z) - y)\frac{dB}{dz}$
Hidden Layer Error	$E_h = \frac{d\mathcal{L}}{dB}\frac{dB}{dz}\frac{dz}{dA}\frac{dA}{dh} = E_o \cdot W_o \cdot \frac{dA}{dh}$
Gradient	$Layer\ BP\ Error\ *\ Layer\ Input$

Python Code Example

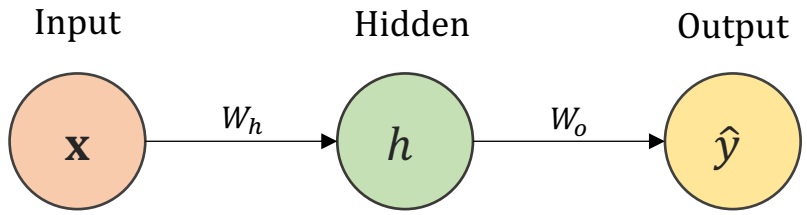
File Edit Selection View Go Run Terminal Help example.py - Visual Studio Code

example.py

C: > example.py > linear

```python
import numpy as np

def linear(z):
    return z

def linear_grad(z):
    return 1

def backprop(x, y, Wh, Wo, lr):
    h = linear(np.dot(x, Wh))              # Hidden layer
    yHat = linear(np.dot(h, Wo))           # Output layer

    # Layer Error
    Eo = (yHat - y) * linear_grad(yHat)
    Eh = Eo * Wo * linear_grad(h)

    # Loss derivative for weights
    dWo = Eo * h
    dWh = Eh * x

    # Update weights
    Wh -= lr * dWh
    Wo -= lr * dWo

```

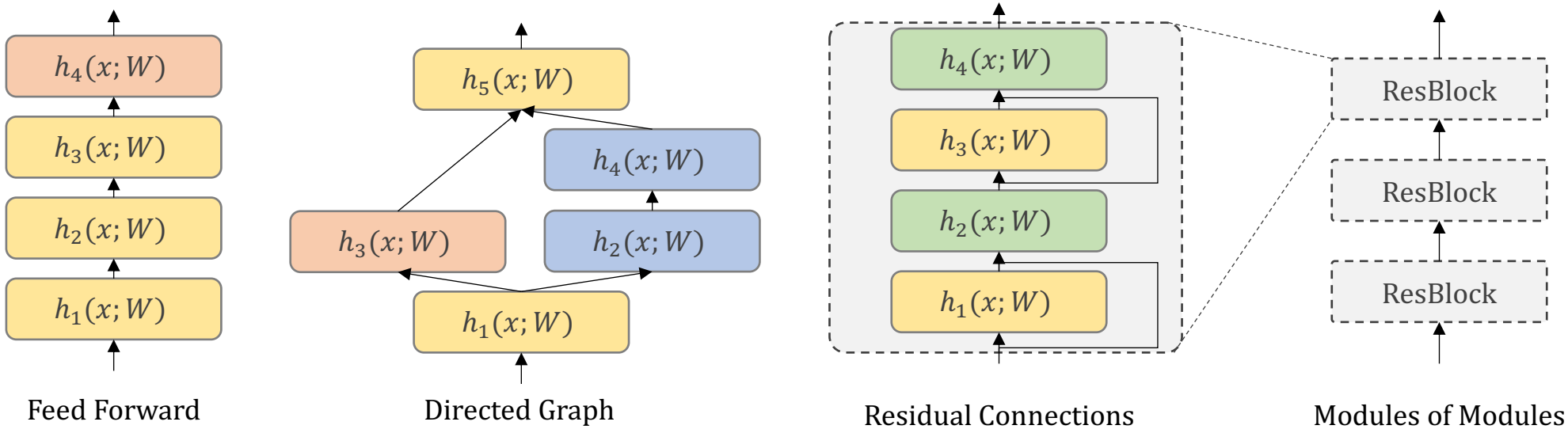
Python 2.7.13 32-bit ⊗ 0 ⚠ 0 Ln 30, Col 1 Spaces: 4 UTF-8 CRLF Python



Modularity in Deep Learning

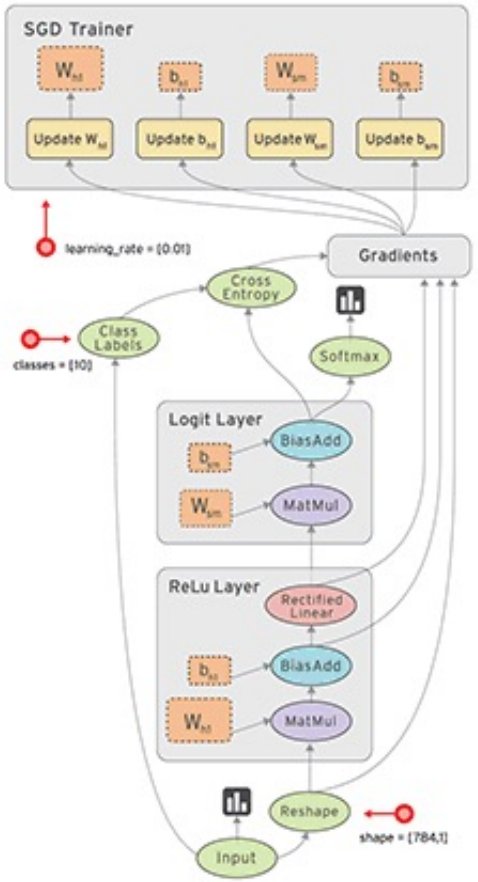
Modularity

- Deep neural networks can be defined as a series of hierarchically connected functions.
- In order to be optimized with gradient descent each of these functions should be differentiable.
- These hierarchies can get very complex!



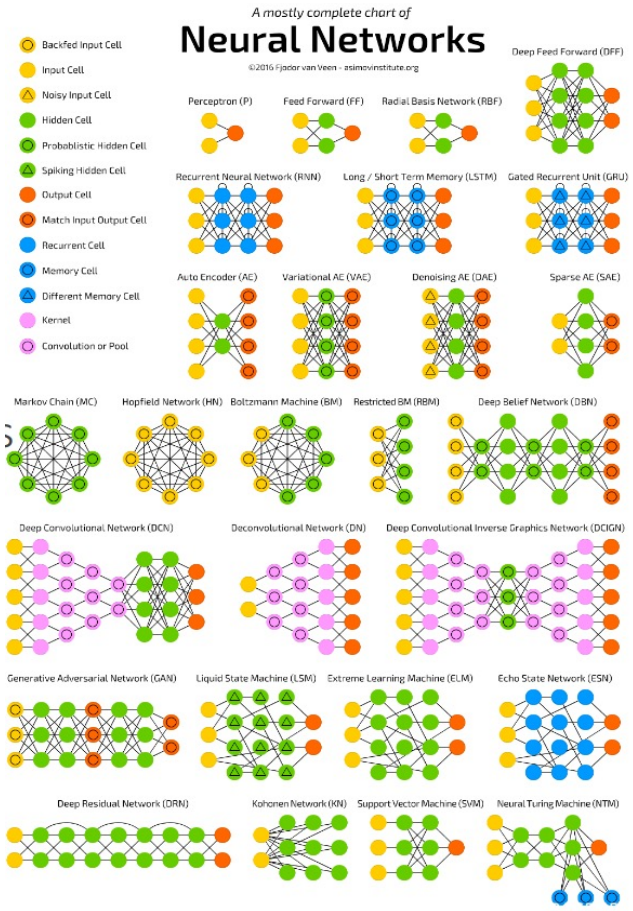
Modules

- A module is a building block for our network
- Each module is a function $a = h_w(x)$ that
  - Contains trainable parameters w
  - Receives as an argument an input x
  - Returns an output a based on the activation function $h(\dots)$
- The activation function should be (at least) first order differentiable (almost) everywhere
- For easier/more efficient backpropagation → store module input
  - Easy to get module output fast
  - Easy to compute derivatives



List of Different Modules/Functions

- Activation Functions
- Loss Functions
- Normalization Layers
- Down-sampling/Up-sampling Layers
- Convolutional Layers
- Concatenation Layers
- Noise Layers
- (Many More)



Task specific modules

- Regression:
  - Target output is a single or multiple continuous values
  - Activation functions for the output layer must be appropriate (e.g., Linear, tanh, sigmoid)
  - Take care of the target range e.g., sigmoid cannot be used for a target variable with range >1 (!!!)
  - Typically used loss functions are MSE (L2), MAE (L1)

- Classification:
  - Target output a single or multiple categorical values.
  - Typically, the different target classes are encoded in one-hot vectors (e.g. [0 0 1 0] -> class3)
  - Activation functions should encode probabilities of the input to belong to a class (e.g. [0.01, 0.002, 0.9, 0.08])
  - Output should sum up to 1 (Softmax)
  - Typically used loss functions are Categorical Cross Entropy (CCE), Kullback-Leibler Divergence (KL)

Activation Functions

Linear Activation Function

- Activation:

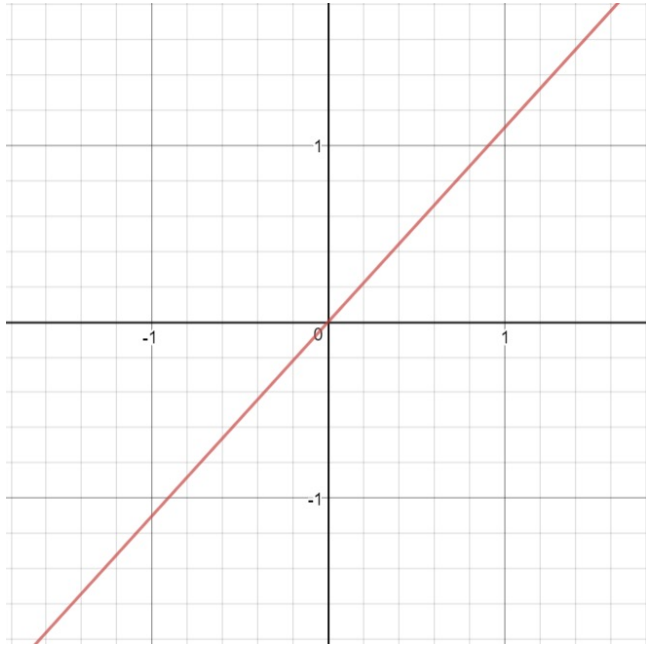
$$\mathrm{R}(z) = z$$

- Gradient:

$$\frac{dR}{dz} = 1$$

- No activation saturation

- Hence, strong & stable gradients
  - Reliable learning with linear modules



Rectified Linear Unit (ReLU)

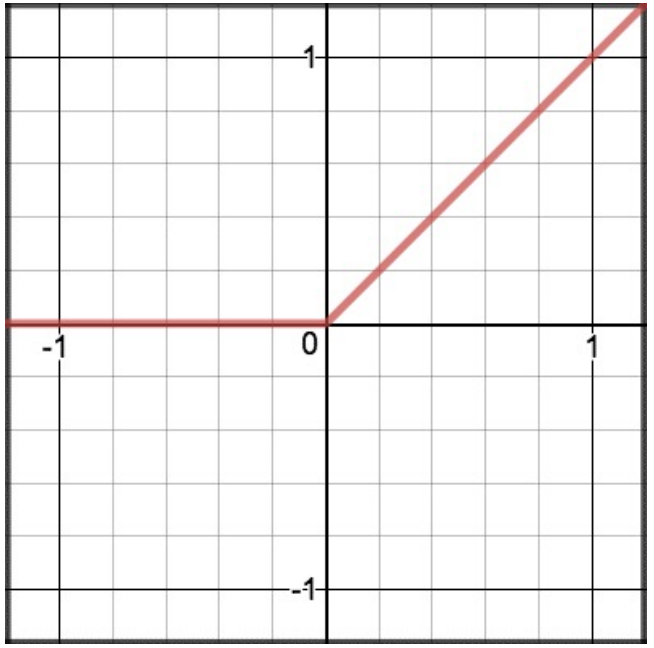
- Activation:

$$R(z) = \begin{cases} z, & z > 0 \\ 0, & z \leq 0 \end{cases}$$

- Gradient:

$$\frac{dR}{dz} = \begin{cases} 1, & z > 0 \\ 0, & z < 0 \end{cases}$$

- Strong gradients: either 0 or 1
- Fast gradients: just a binary comparison
- It is not differentiable at 0, but not a big problem
  - An activation of precisely 0 rarely happens with non-zero weights, and if it happens, we choose a convention
- Nowadays ReLU is the default non-linearity



Sigmoid Function

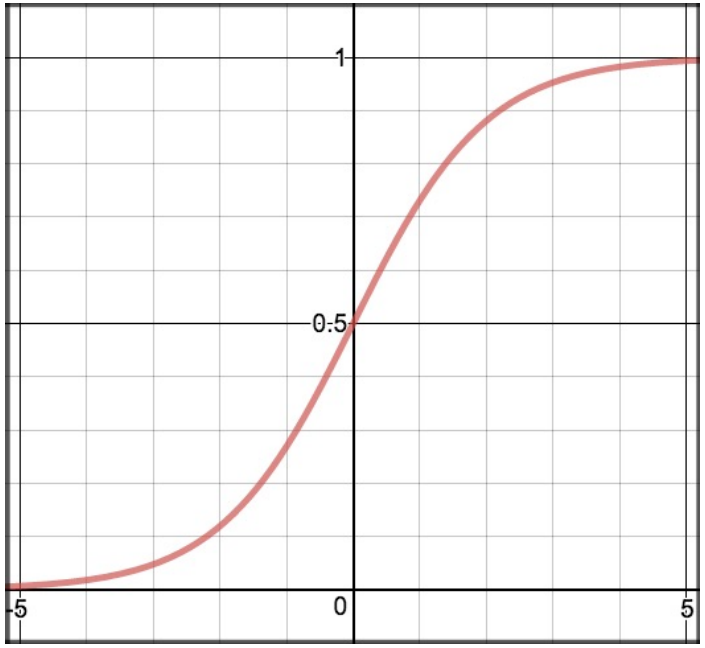
- Activation:

$$S(z) = \frac{1}{1 + e^{-z}}$$

- Gradient:

$$\frac{dS}{dz} = S(z) \cdot (1 - S(z))$$

- Smooth gradients
- Outputs in the range [0,1]



Tanh Function

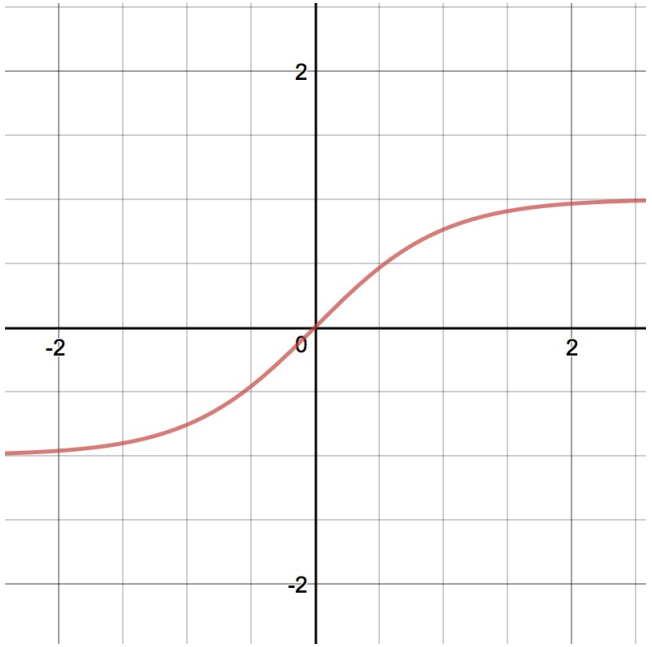
- Activation:

$$T(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- Gradient:

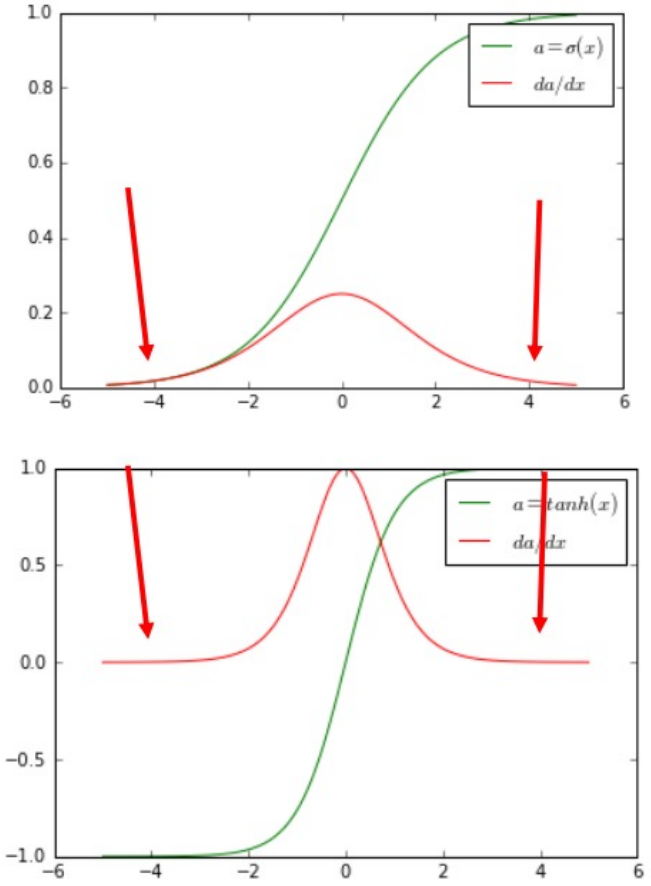
$$\frac{dT}{dz} = 1 - T(z)^2$$

- Smooth gradients
- output in the range [-1,1]



Sigmoid vs Tanh

- Functional form is very similar
- $\tanh(x)$ due to the output range [-1,+1]:
  - Larger gradient ranges, [0,1] instead of [0, 0.5]
- Both saturate at the extreme values →0 gradients
  - "Overconfident", without necessarily being correct
  - Especially bad when in the middle layers: why should a neuron be overconfident, when it represents a latent variable
- The gradients for both are <1 so we can have small gradient values if propagating errors in multiple layers
- From the two, $\tanh(x)$ enables better learning
  - But still, not a great choice



Softmax

- Activation:

$$S(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

- Outputs probability distribution for N classes

$$\sum_{j=1}^{N} S(z_j) = 1$$

- Avoid exponenting too large/small numbers → better stability

Loss Functions

Loss functions for different Tasks

- Regression Tasks
  - Mean Absolute Error (MAE, L1)

$$\mathcal{L}_{MAE} = \frac{1}{n}\sum_{i=1}^{n} \|f(x_i) - y_i\|$$

  - Mean Squared Error (MSE, L2)

$$\mathcal{L}_{MSE} = \frac{1}{n}\sum_{i=1}^{n} (f(x_i) - y_i)^2$$

- Classification Tasks
  - Categorical Cross Entropy Loss (CCE)

$$\mathcal{L}_{CCE} = -\sum_{j=1}^{k} y_j \cdot \log(\hat{y}_j)$$

Take Home Messages

- Deep learning and neural networks are based on simple mathematical functions.
- Different modules can be used and combined depending on the data and the problems we need to address.
- Depending on the task at hand the appropriate modules should be selected!
  - Regression: Linear, Sigmoid, L1, L2
  - Classification: Softmax, CCE
- Training is happening with gradient descend (gradient should be available almost everywhere).
- The parameters are optimized by backpropagating errors.
- Backpropagating errors is the heart of optimizing parametrical models like Deep Neural Networks.