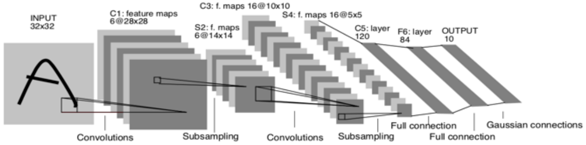
[Deep] Neural Networks

Deep Learning
Computer Science Master Degree

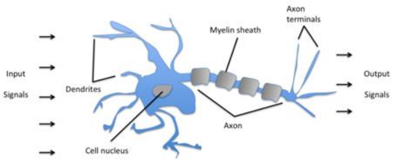
Nicoletta Noceti
Nicoletta.noceti@unige.it

Some definitions



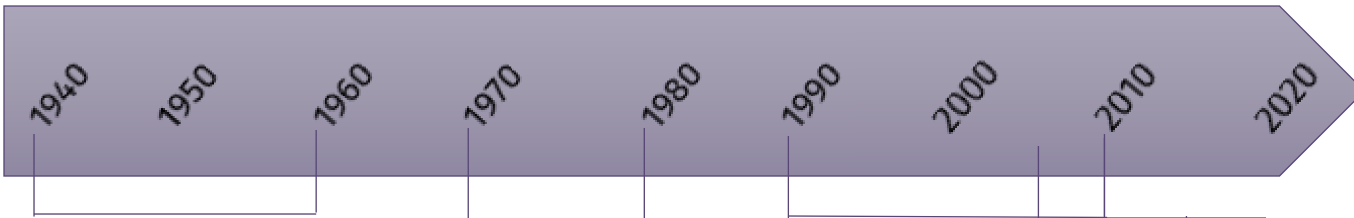
- **Deep learning** is a family of machine learning methods based on artificial neural networks (ANNs) that use multiple layers to **progressively extract higher level features from raw input**

- **ANNs** are computing systems inspired by the **biological neural networks**, based on a collection of units/nodes, the artificial neurons, loosely modelling the neurons.
  a biological brain



- <u>But also</u>: Deep learning methods are representation-learning methods with multiple levels of representation

A little bit of history



Early NN models, not so effective

Little work

The idea of training DNN with backpropagation, but difficulty of training for the limited computing capacity (...and limited availability of data)

Attempts of overcoming training problems

DL explosion! Availability of APIs and pre-trained models

The advent of GPUs prepares the terrain for DNN diffusion

Why now?

Availability of data

Wikipedia
The Free Encyclopedia

Hardware

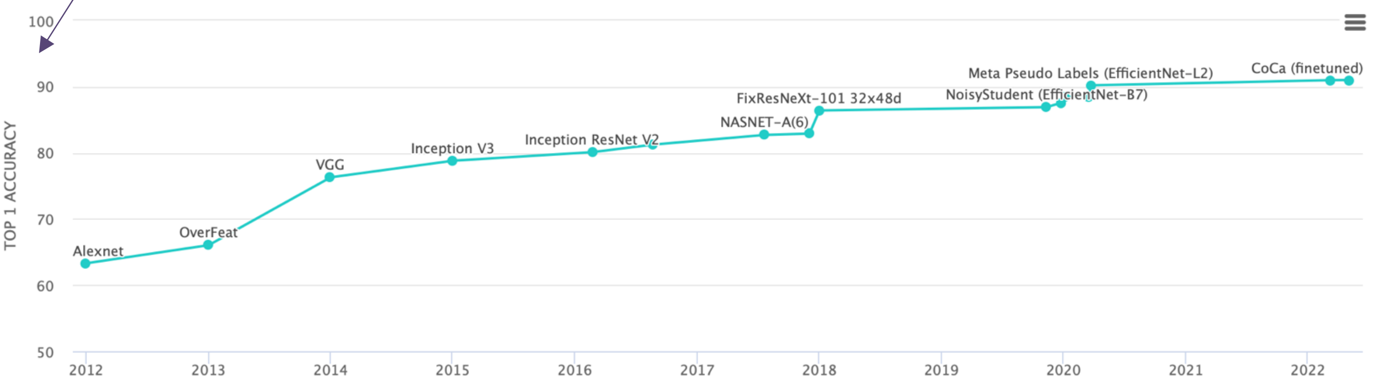
Software

TensorFlow

What now?

Image recognition

Human level



UniGe | MaLGa

[Earliest] Generative models

This person does not exist

[Earliest] Generative models

[Obama fake video]

Large Language Models

ChatGPT

Gemini

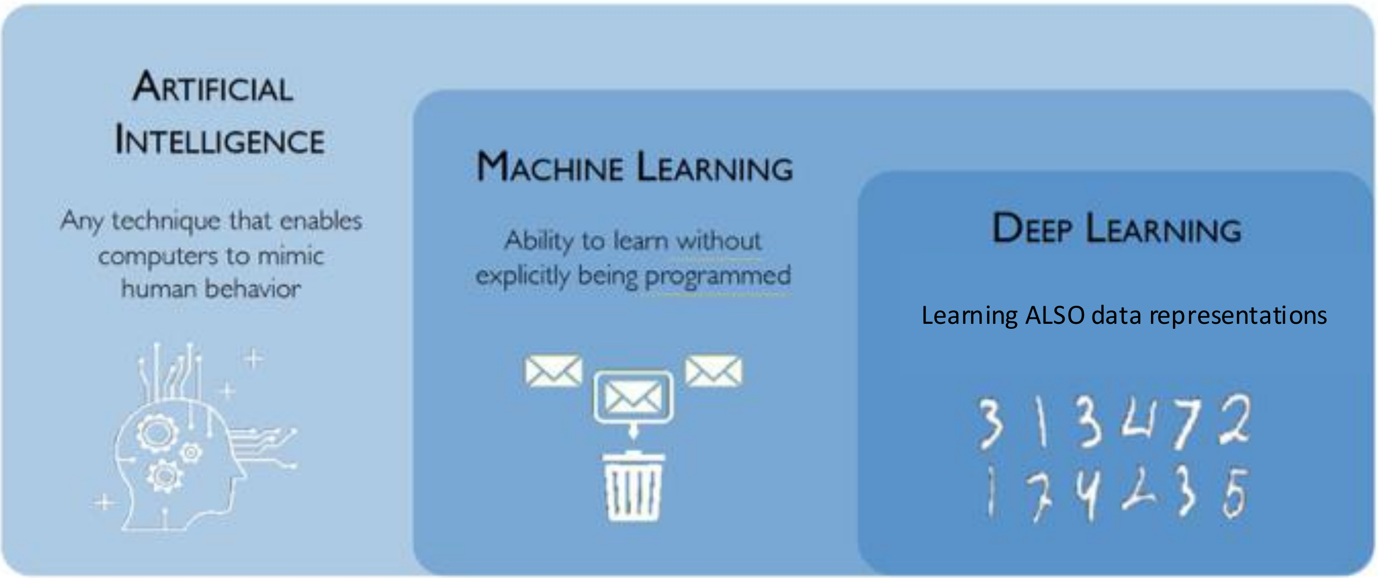
deepseek

Llama 3
From Meta


The street of a medieval fantasy town, at dawn, dark, 4k, highly detailed

Text-to-image generation

Deep Learning



Deep Networks as representation learning

Feature engineering

$$\Phi(\Psi(x))$$

x

Data Representation (e.g. feature extraction)

$\Psi(x)$

Classifier

y

Feature learning

This part learns the representation

This part trains The classifier

x

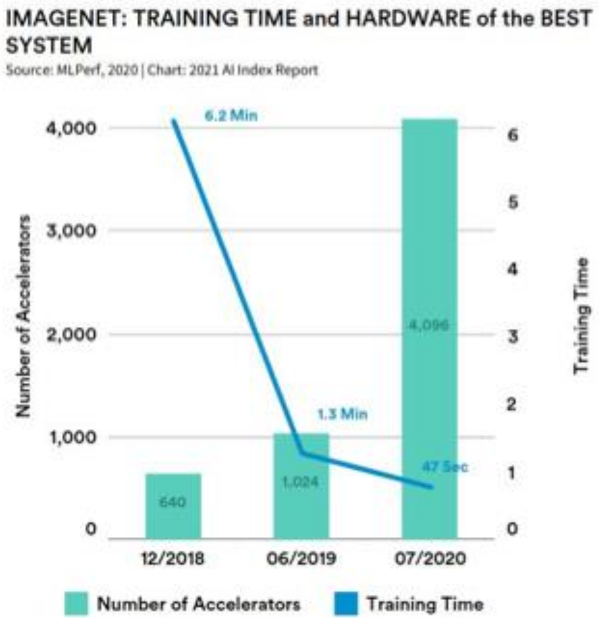
(Deep) NN

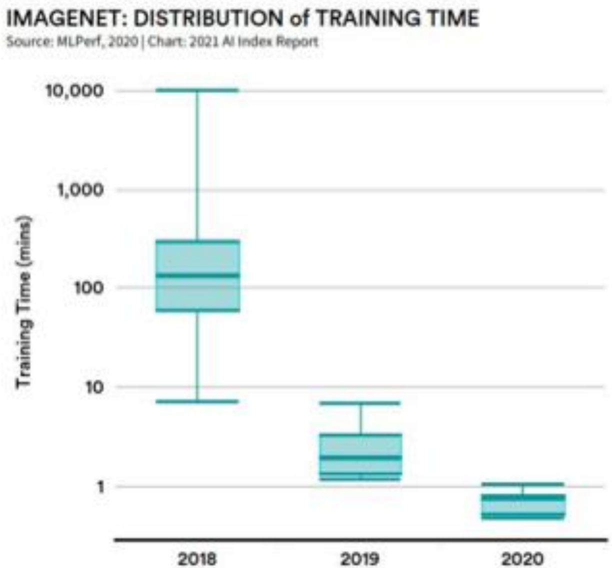
y

$$f(x)$$

Deep learning

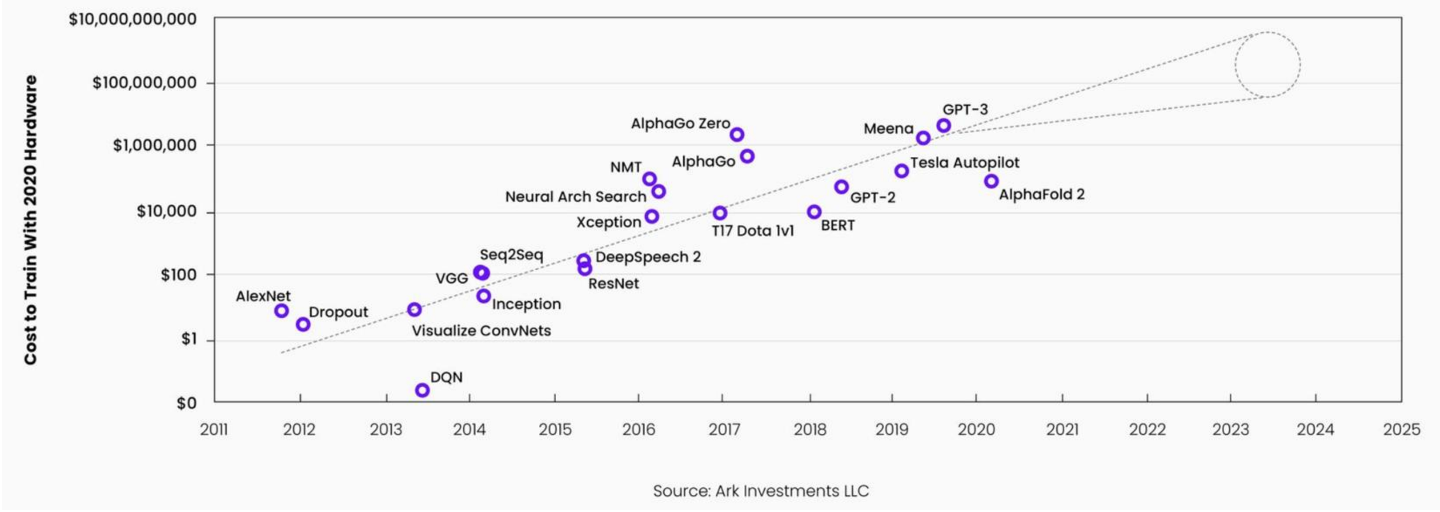
The price we pay





Deep learning

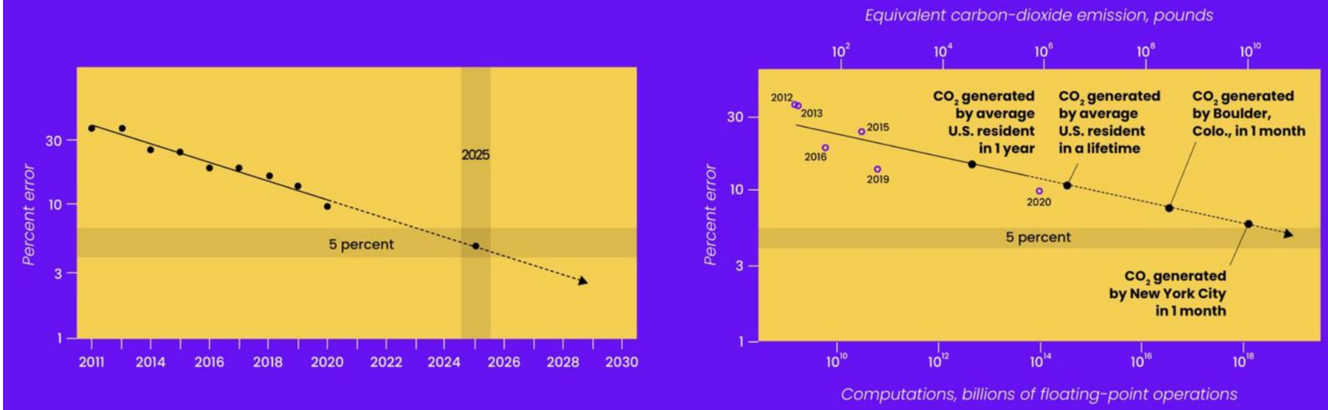
The price we pay



UniGe | MaLGa

Deep learning

The price we pay



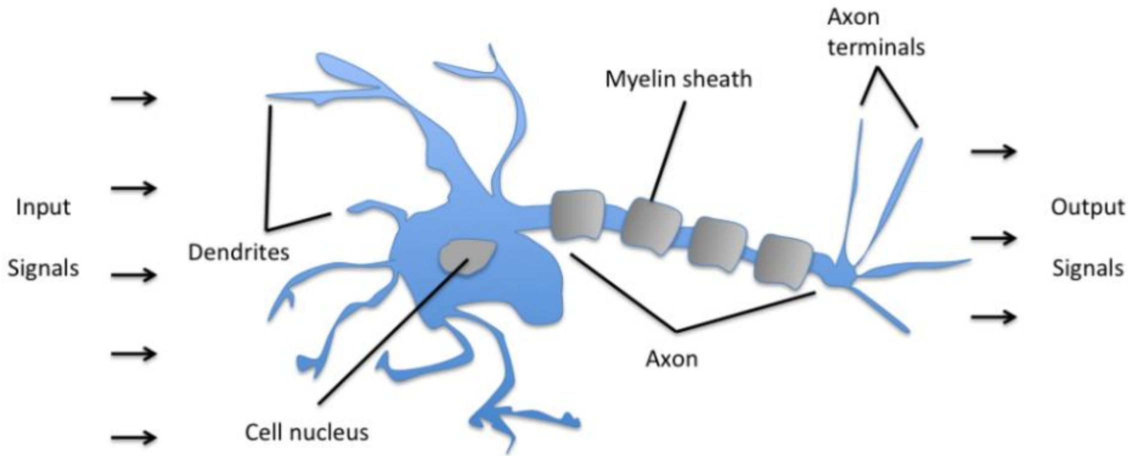
Extrapolating the gains of recent years might suggest that by 2025 the error level in the best deep-learning systems designed for recognizing objects in the ImageNet data set should be reduced to just 5 percent (left). But the computing resources and energy required to train such a future system would be enormous, leading to the emission of as much carbon dioxide as New York City generates in one month (right).

SOURCE: N.C. THOMPSON, K. GREENWALD, K. LEE, G.F. MANSO

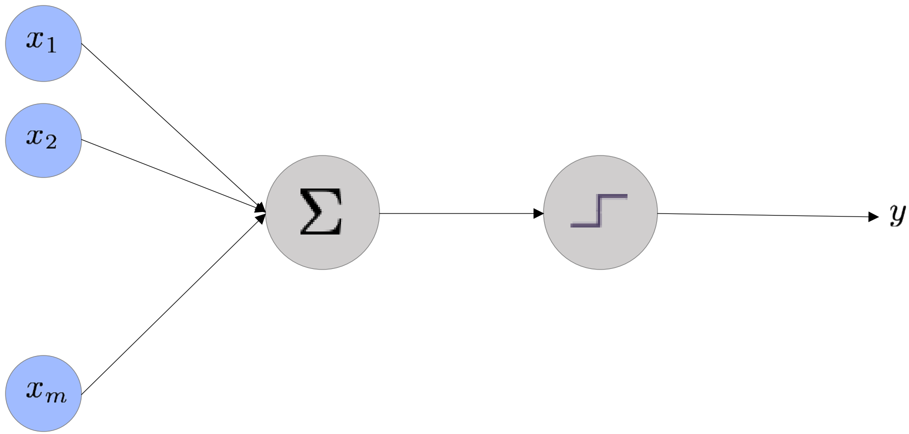
UniGe | MaLGa

Single Layer Perceptron

Biological neuron



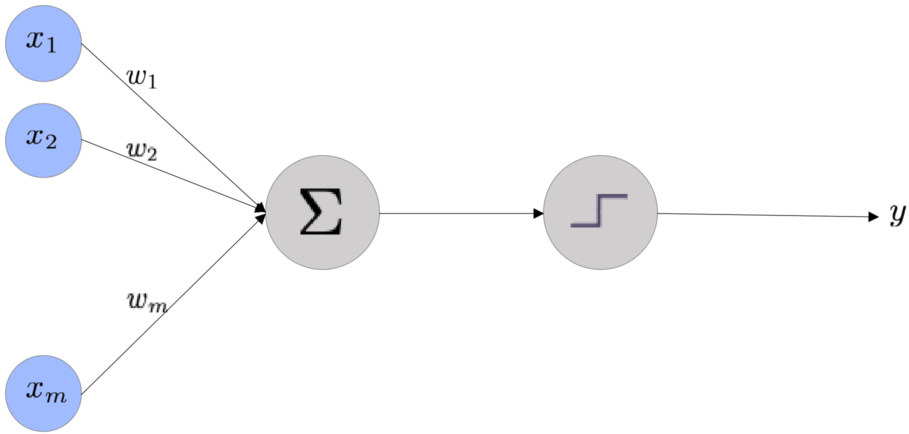
McCulloch & Pitts Neuron Model (1943)



$$f_\theta : \mathbb{R}^m \to \mathbb{R}$$

$$y = f_\theta(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_{i=1}^m x_i > \theta \\ 0 & \text{otherwise} \end{cases}$$

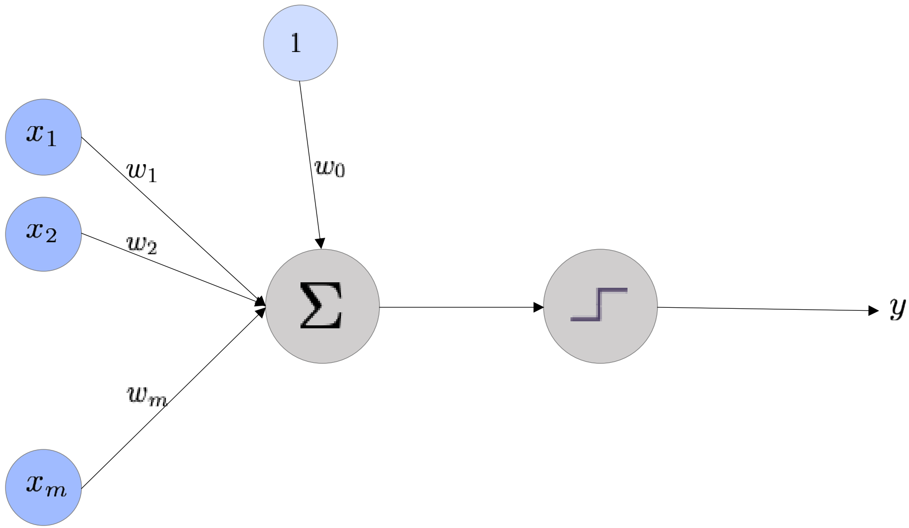
Towards the Single Layer Perceptron



$$f_{\mathbf{w},\theta} : \mathbb{R}^m \to \mathbb{R}$$

$$y = f_{\mathbf{w},\theta}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^\intercal \mathbf{x} > \theta \\ 0 & \text{otherwise} \end{cases}$$

Towards the Single Layer Perceptron



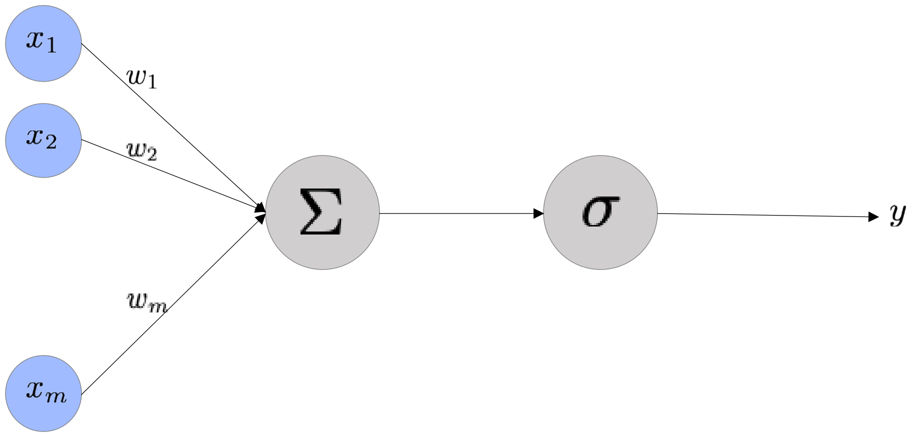
$$f_{\mathbf{w},\theta} : \mathbb{R}^m \rightarrow \mathbb{R}$$

$$y = f_{\mathbf{w},\theta}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^\intercal[1, \mathbf{x}] > \theta \\ 0 & \text{otherwise} \end{cases}$$

From now on we will use as standard the notation $\mathbf{w}^\intercal \mathbf{x}$

It means that we assume the presence of the bias term to be addressed

Towards the Single Layer Perceptron



$$f_{\mathbf{w},\sigma} : \mathbb{R}^m \to \mathbb{R}$$

$$y = f_{\mathbf{w},\sigma}(\mathbf{x}) = \sigma(\mathbf{w}^\intercal \mathbf{x})$$

Non-Linear activation functions

- Nonlinear activation is key to achieving good function approximation
- It takes a single number and maps it to a different numerical value
- Popular functions

Tanh

Sigmoid

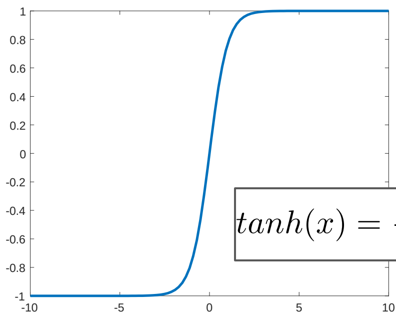
ReLU

Leak(y) ReLU

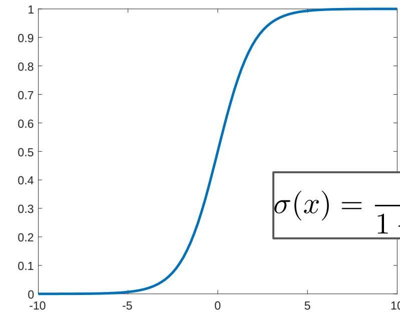
Softmax

To practice with non-linear activation functions: https://playground.tensorflow.org/

Non-linear Activation functions



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



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

Tanh

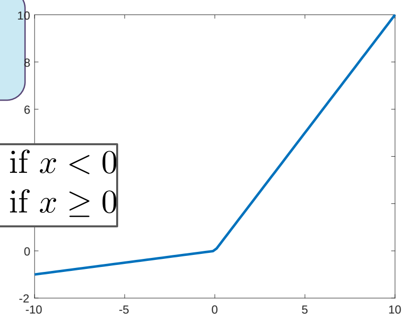
Sigmoid

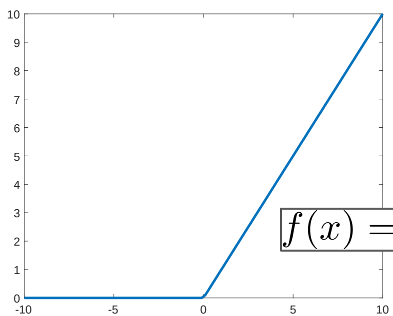
ReLU

Leak(y) ReLU

$$f(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

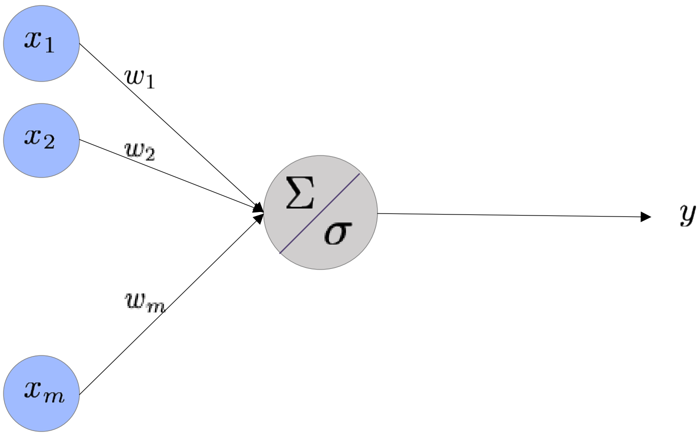
$$\alpha = 0.1$$





$$f(x) = max(0, x)$$

Towards the Single Layer Perceptron



$$f_{\mathbf{w},\sigma} : \mathbb{R}^m \to \mathbb{R}$$

$$y = f_{\mathbf{w},\sigma}(\mathbf{x}) = \sigma(\mathbf{w}^\intercal \mathbf{x})$$

UniGe | MaLGa

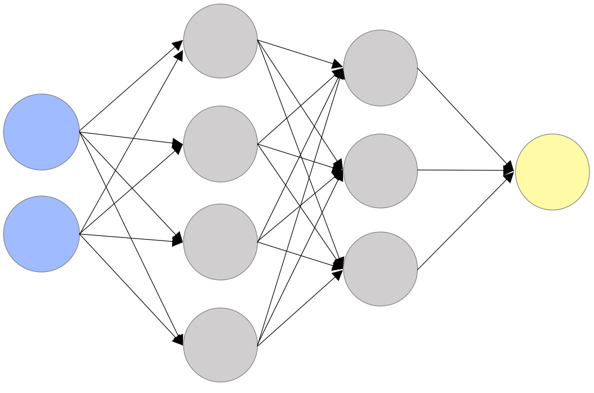
Multi-Layer Perceptron

Neural Networks

Neural Networks are composed of chains of layers that may be of three different types:

- INPUT LAYER
- (MULTIPLE) HIDDEN LAYER(S)
- OUTPUT LAYER

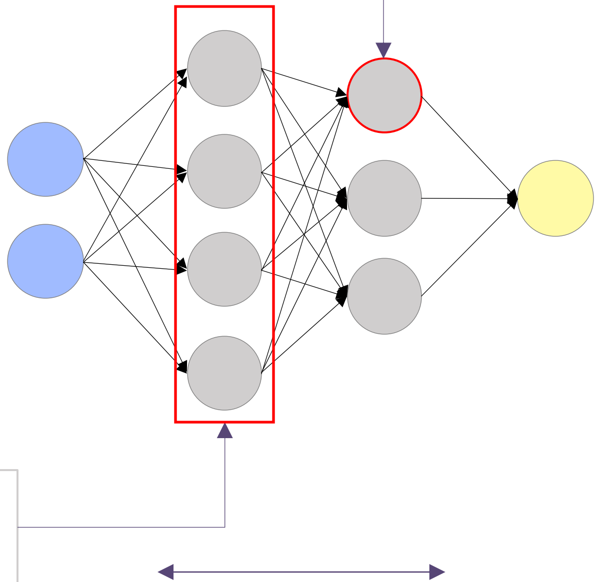
The layers are fully connected



Neural Networks

Each element of the layer plays the role of a neuron

The central layers are hidden. i.e. the intermediate output is not explicitly shown



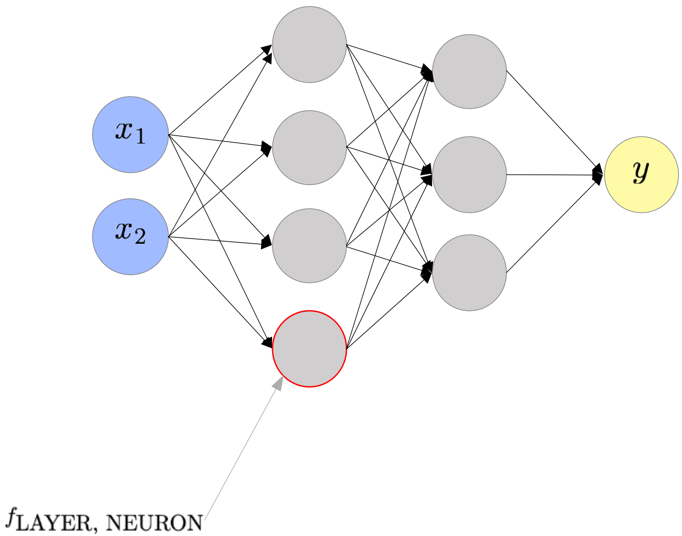
The dimensionality of the hidden layers determine the width

The depth is the number of hidden layers

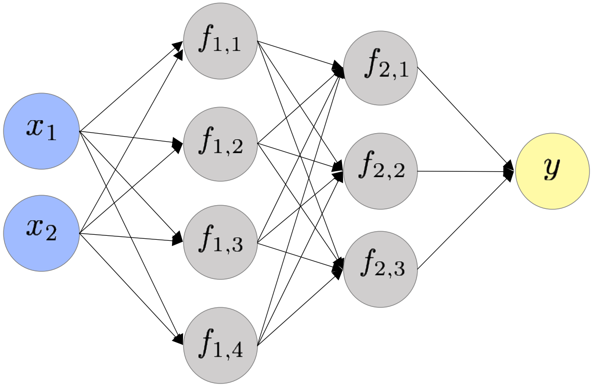
One network, two phases

- Forward propagation: the input flows into the network and produces a cost
- Backward propagation: allows the info to flow back into the net to compute the gradient (during the optimization)

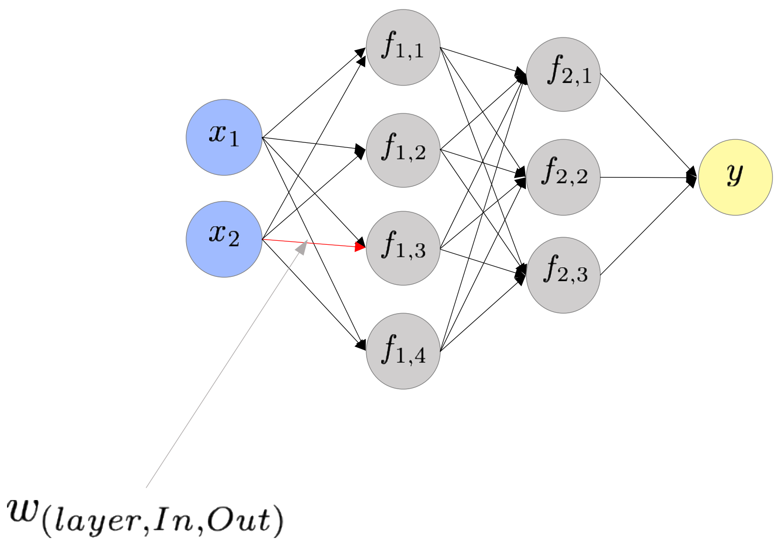
Forward propagation



Forward propagation

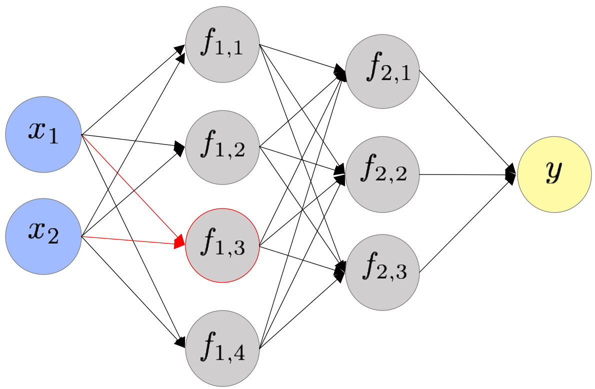


Forward propagation



UniGe | MaLGa

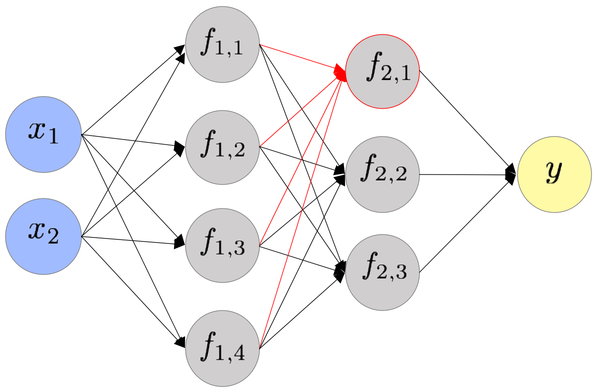
Forward propagation



$$f_{1,3}(\mathbf{x}) = \sigma(\mathbf{w}_{(1,:,3)}^{\intercal}\mathbf{x})$$

$$\mathbf{w}_{(1,:,3)} = [w_{(1,1,3)} w_{(1,2,3)}]$$

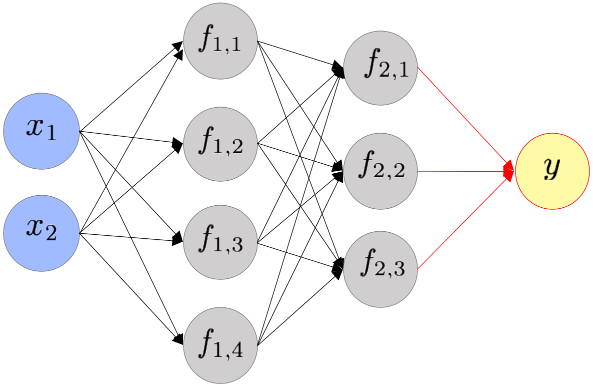
Forward propagation



$$f_{2,1}(\mathbf{f}_{(1,:)}) = \sigma(\mathbf{w}^{\intercal}_{(2,:,1)}\mathbf{f}_{(1,:)})$$

$$\mathbf{f}_{(1,:)} = [f_{1,1} f_{1,2} f_{1,3} f_{1,4}]$$

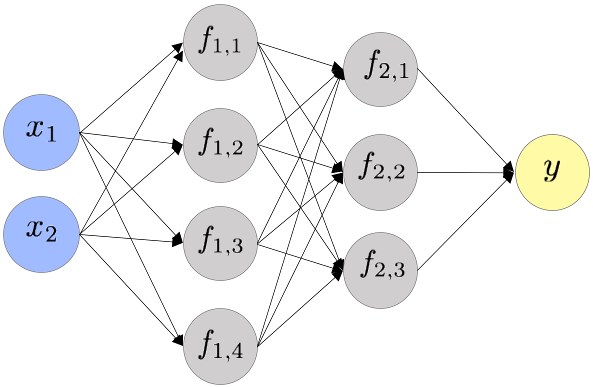
Forward propagation



$$y = \sigma(\mathbf{w}_{(o,:)}^{\intercal}\mathbf{f}_{(2,:)})$$

$$\mathbf{f}_{(2,:)} = [f_{2,1} f_{2,2} f_{2,3}]$$

Forward propagation



... hence, In this example...

$$f_{\mathbf{w}} : \mathbb{R}^2 \to \mathbb{R}$$

$$\mathbf{w} \in \mathbb{R}^{23+8}$$

Bias terms

weights

Training a Neural Network

Training a Neural Network

- Training a (Deep) Neural Network means (as usual) learning the values for the model parameters (weights, bias terms) from the training set

- An essential element of training is (as usual) the loss function, which estimates how much we lose with the prediction in place of the real output y

$$\ell : Y \times Y \to [0, +\infty]$$

where Y is the space of the output of the estimated function

$$f_{\mathbf{w}} : X \to Y$$

Training a Neural Network

- Training a (Deep) Neural Network means (as usual) learning the values for the model parameters (weights, bias terms) from the training set

- An essential element of training is (as usual) the loss function, which estimates how much we lose with the prediction in place of the real output y

$$\ell : Y \times Y \to [0, +\infty]$$

where Y is the space of the output of the estimated function

$$f_{\mathbf{w}} : X \to Y$$

Binary classification

$$f_{\mathbf{w}} : \mathbb{R}^m \to \{-1, 1\}$$

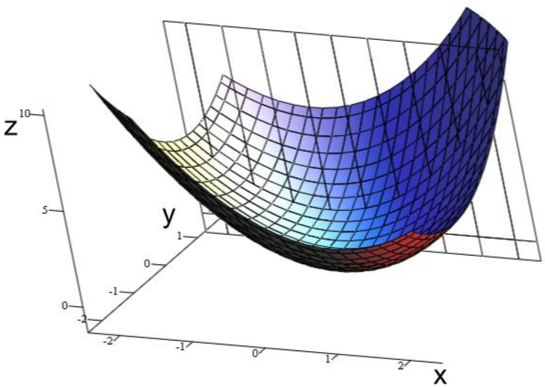
Multi-class classification (with M classes)

$$f_{\mathbf{w}} : \mathbb{R}^m \to \{0, \ldots, M-1\}$$

(Scalar) regression

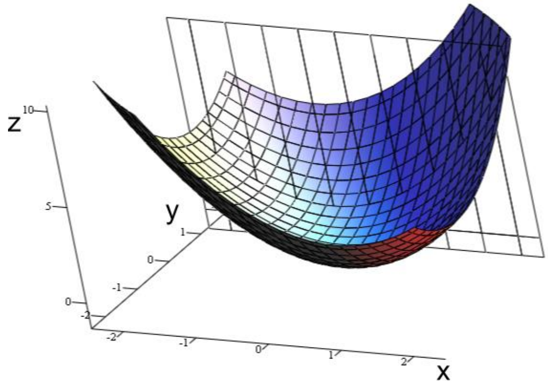
$$f_{\mathbf{w}} : \mathbb{R}^m \to \mathbb{R}$$

Training a deep NN is not...

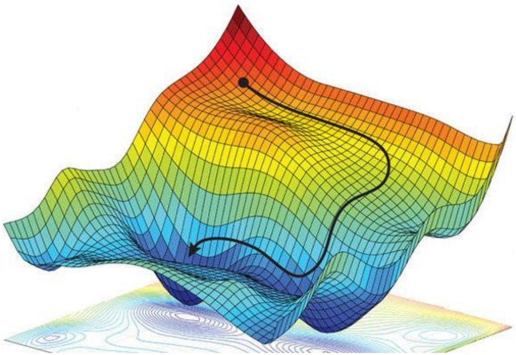


https://towardsdatascience.com/understand-convexity-in-optimization-db87653bf920?gi=b93b0698a062
https://medium.com/swlh/non-convex-optimization-in-deep-learning-26fa30a2b2b3

Training a deep NN is not...



...but more like...



https://towardsdatascience.com/understand-convexity-in-optimization-db87653bf920?gi=b93b0698a062
https://medium.com/swlh/non-convex-optimization-in-deep-learning-26fa30a2b2b3

Training a Neural Network

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^P} \frac{1}{n}\sum_{i=1}^{n} \ell(f_{\mathbf{w}}(\mathbf{x}^i), y^i)$$

Total number of model parameters

$$\mathbf{x}^i \in X$$

$$y^i \in Y$$

Training set $S = \{\mathbf{x}^i, y^i\}_{i=1}^n$

Training a Neural Network

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^P} \frac{1}{n}\sum_{i=1}^{n} \ell(f_{\mathbf{w}}(\mathbf{x}^i), y^i)[+\lambda\Psi(f_{\mathbf{w}})]$$

Total number of model parameters

$$\mathbf{x}^i \in X$$

$$y^i \in Y$$

Regularization term

Training set $S = \{\mathbf{x}^i, y^i\}_{i=1}^{n}$

Training a Neural Network

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^P} \frac{1}{n}\sum_{i=1}^{n} \ell(f_{\mathbf{w}}(\mathbf{x}^i), y^i)[+\lambda\Psi(f_{\mathbf{w}})]$$

Total number of model parameters

$$\mathbf{x}^i \in X$$

Regularization term

$$y^i \in Y$$

Training set $S = \{\mathbf{x}^i, y^i\}_{i=1}^n$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^P} J(S;\mathbf{w})$$

To find the parameters we need an optimization strategy

Training a Neural Network

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^P} J(S;\mathbf{w})$$

- We may resort to iterative approaches → Gradient Descent

$$\mathbf{w}_0 = 0$$

Or with random values

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma\nabla J(S;\mathbf{w}_{t-1})$$

- We need to compute the gradient of a possibly very complex function!

One network, two phases

- Forward propagation: the input flows into the network and produces a cost
- Backward propagation: allows the info to flow back into the net to compute the gradient (during the optimization)

Training a DNN

Back-propagation

- Backpropagation (1960s) aims to minimize the cost function by adjusting the weights and biases of the networks
- The level of adjustment is determined by the gradients of the cost function with respect to those parameters.
- The goal of backpropagation is to compute the partial derivatives of the cost function with respect to any parameter in the network.

Training a DNN

Back-propagation

- A key ingredient of back-propagation is the chain rule of derivation
- Example: is F is a composite function such that

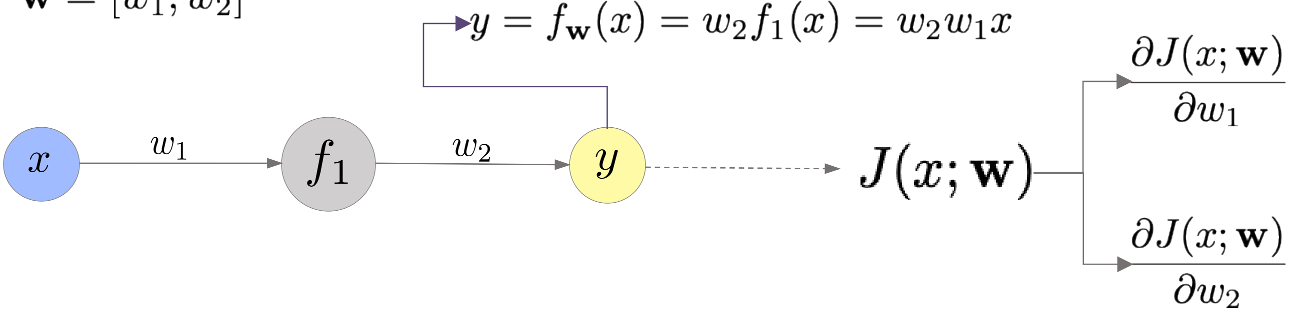
$$F = f \circ g \circ h \circ u \circ v$$

then

$$\frac{\partial F(x)}{\partial x} = \frac{\partial f(g(h(u(v(x)))))}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial h}\frac{\partial h}{\partial u}\frac{\partial u}{\partial v}\frac{\partial v}{\partial x}$$

Backpropagation: example 1

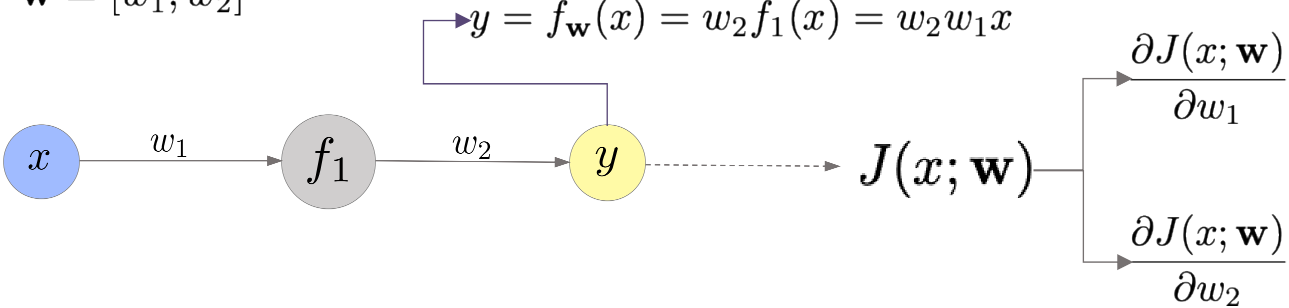
$\mathbf{w} = [w_1, w_2]$



From http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Backpropagation: example 1

$$\mathbf{w} = [w_1, w_2]$$



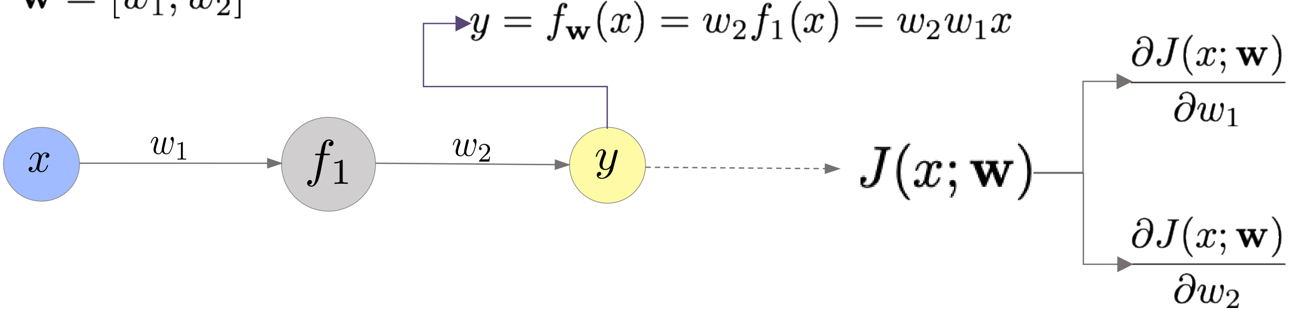
$$\frac{\partial J(x;\mathbf{w})}{\partial w_1}$$

$$\frac{\partial J(x;\mathbf{w})}{\partial w_2}$$

From http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Backpropagation: example 1

$$\mathbf{w} = [w_1, w_2]$$



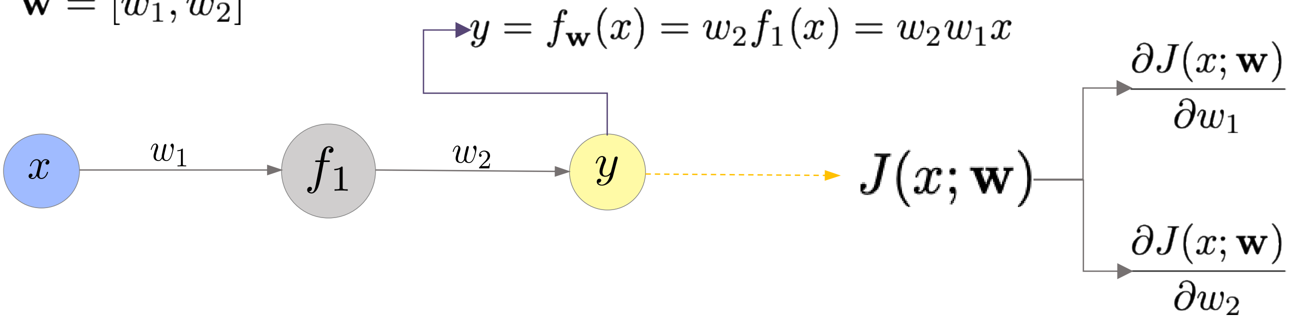
$$\frac{\partial J(x; \mathbf{w})}{\partial w_2} = \frac{\partial J(x; \mathbf{w})}{\partial f_{\mathbf{w}}(x)} \frac{\partial f_{\mathbf{w}}(x)}{\partial w_2}$$

$$\frac{\partial J(x; \mathbf{w})}{\partial w_2}$$

From http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Backpropagation: example 1

$$\mathbf{w} = [w_1, w_2]$$



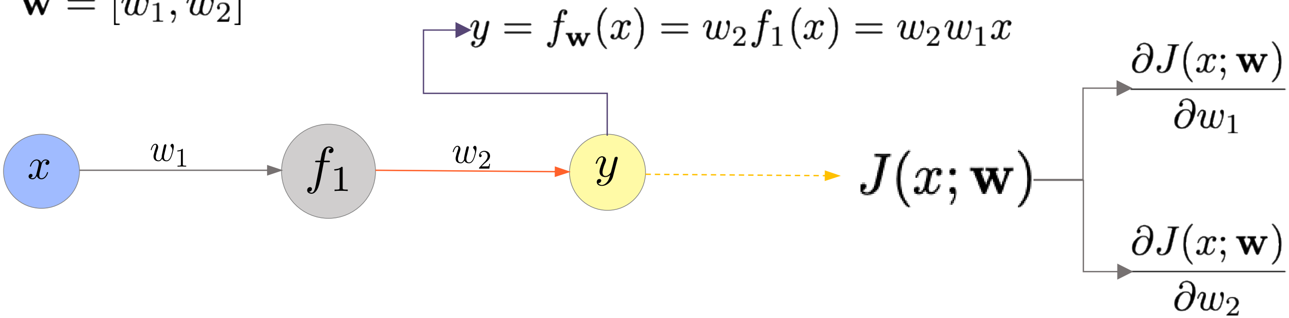
$$\frac{\partial J(x;\mathbf{w})}{\partial w_2} = \boxed{\frac{\partial J(x;\mathbf{w})}{\partial f_{\mathbf{w}}(x)}} \frac{\partial f_{\mathbf{w}}(x)}{\partial w_2}$$

$$\frac{\partial J(x;\mathbf{w})}{\partial w_2}$$

From http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Backpropagation: example 1

$$\mathbf{w} = [w_1, w_2]$$

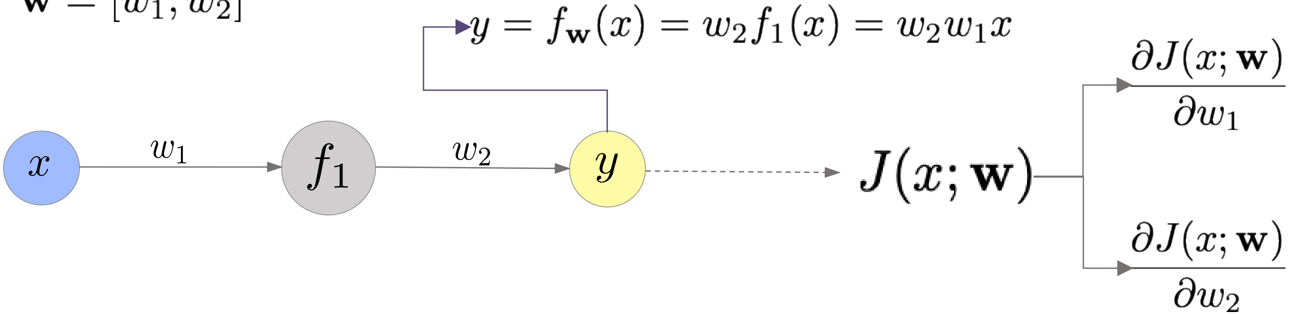


$$\frac{\partial J(x;\mathbf{w})}{\partial w_2} = \boxed{\frac{\partial J(x;\mathbf{w})}{\partial f_{\mathbf{w}}(x)}}\boxed{\frac{\partial f_{\mathbf{w}}(x)}{\partial w_2}}$$

$$\frac{\partial J(x;\mathbf{w})}{\partial w_2}$$

From http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

$$\mathbf{w} = [w_1, w_2]$$



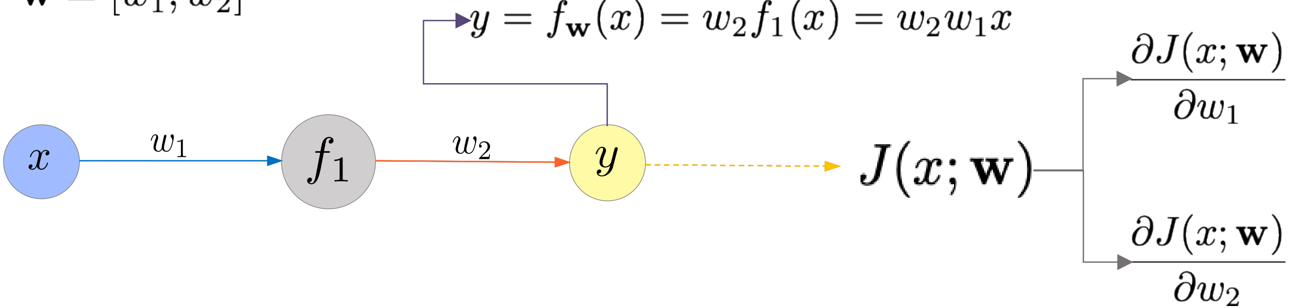
$$\frac{\partial J(x; \mathbf{w})}{\partial w_2} = \frac{\partial J(x; \mathbf{w})}{\partial f_{\mathbf{w}}(x)} \frac{\partial f_{\mathbf{w}}(x)}{\partial w_2}$$

$$\frac{\partial J(x; \mathbf{w})}{\partial w_1} = \frac{\partial J(x; \mathbf{w})}{\partial f_{\mathbf{w}}(x)} \frac{\partial f_{\mathbf{w}}(x)}{\partial f_1(x)} \frac{\partial f_1(x)}{\partial w_1}$$

From http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Backpropagation: example 1

$$\mathbf{w} = [w_1, w_2]$$



$$\frac{\partial J(x;\mathbf{w})}{\partial w_2} = \frac{\partial J(x;\mathbf{w})}{\partial f_{\mathbf{w}}(x)} \frac{\partial f_{\mathbf{w}}(x)}{\partial w_2}$$

$$\frac{\partial J(x;\mathbf{w})}{\partial w_1} = \boxed{\frac{\partial J(x;\mathbf{w})}{\partial f_{\mathbf{w}}(x)}} \boxed{\frac{\partial f_{\mathbf{w}}(x)}{\partial f_1(x)}} \boxed{\frac{\partial f_1(x)}{\partial w_1}}$$

From http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Backpropagation: example 2

- Let's consider the case where we have only one neuron, this time with a non-linearity
- Let's also assume we are using the square loss

$$\mathbf{x}^i \in X$$

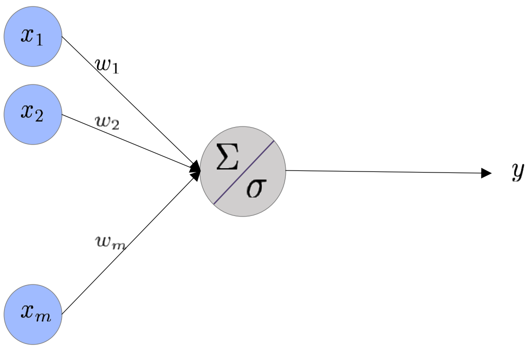
$$y^i \in Y$$

Training set $S = \{\mathbf{x}^i, y^i\}_{i=1}^n$

$$J(S;\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y^i - f_{\mathbf{w}}(\mathbf{x}^i))^2 =$$

$$= \frac{1}{n}\sum_{i=1}^{n}(y^i - f_{\sigma}(\mathbf{w}^\intercal \mathbf{x}^i))^2 =$$

$$= \frac{1}{n}\sum_{i=1}^{n}(y^i - \sigma(\mathbf{w}^\intercal \mathbf{x}^i))^2$$



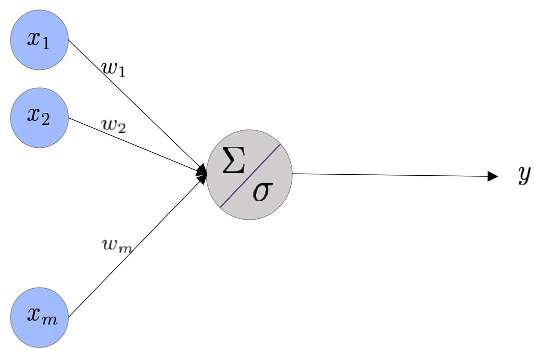
Backpropagation: example 2

- Let's consider the case where we have only one neuron, this time with a non-linearity
- Let's also assume we are using the square loss

$$\mathbf{x}^i \in X$$

$$y^i \in Y$$

Training set $S = \{\mathbf{x}^i, y^i\}_{i=1}^n$



$$J(S;\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y^i - f_{\mathbf{w}}(\mathbf{x}^i))^2 =$$

$$= \boxed{\frac{1}{n}\sum_{i=1}^{n}(y^i - f_{\sigma}(\mathbf{w}^\intercal\mathbf{x}^i))^2} =$$

$$= \frac{1}{n}\sum_{i=1}^{n}(y^i - \sigma(\mathbf{w}^\intercal\mathbf{x}^i))^2$$

Backpropagation: example 2

$$J(S;\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y^i - f_\sigma(\mathbf{w}^\intercal \mathbf{x}^i))^2$$

Backpropagation: example 2

$$J(S;\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y^i - f_\sigma(\mathbf{w}^\intercal \mathbf{x}^i))^2$$

For simplicity, we consider the cost function restricted to sample i

$$J((\mathbf{x}^i, y^i);\mathbf{w}) = (y^i - f_\sigma(\mathbf{w}^\intercal \mathbf{x}^i))^2$$

Backpropagation: example 2

$$J(S;\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y^i - f_\sigma(\mathbf{w}^\intercal \mathbf{x}^i))^2$$

For simplicity, we consider the cost function restricted to sample i

$$\boxed{J((\mathbf{x}^i, y^i);\mathbf{w})} = (y^i - f_\sigma(\mathbf{w}^\intercal \mathbf{x}^i))^2$$

For the sake of compactness, we call it

$$J^i(\mathbf{w})$$

Backpropagation: example 2

$$J(S;\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y^i - f_\sigma(\mathbf{w}^\top \mathbf{x}^i))^2$$

For simplicity, we consider the cost function restricted to sample i

$$\boxed{J((\mathbf{x}^i, y^i);\mathbf{w})} = (y^i - f_\sigma(\mathbf{w}^\top \mathbf{x}^i))^2$$

For the sake of compactness, we call it

$$J^i(\mathbf{w})$$

We apply the chain rule of derivation

$$\frac{\partial J^i(\mathbf{w})}{\partial w_k} = \frac{\partial J^i(\mathbf{w})}{\partial f_\sigma(\mathbf{w}^\top \mathbf{x}^i)} \frac{\partial f_\sigma(\mathbf{w}^\top \mathbf{x}^i)}{\partial \mathbf{w}^\top \mathbf{x}^i} \frac{\partial \mathbf{w}^\top \mathbf{x}^i}{\partial w_k}$$

Backpropagation: example 2

$$\frac{\partial J^i(\mathbf{w})}{\partial w_k} = \frac{\partial J^i(\mathbf{w})}{\partial f_\sigma(\mathbf{w}^\intercal \mathbf{x}^i)} \frac{\partial f_\sigma(\mathbf{w}^\intercal \mathbf{x}^i)}{\partial \mathbf{w}^\intercal \mathbf{x}^i} \frac{\partial \mathbf{w}^\intercal \mathbf{x}^i}{\partial w_k}$$

Backpropagation: example 2

$$\frac{\partial J^i(\mathbf{w})}{\partial w_k} = \frac{\partial J^i(\mathbf{w})}{\partial f_\sigma(\mathbf{w}^\top \mathbf{x}^i)} \frac{\partial f_\sigma(\mathbf{w}^\top \mathbf{x}^i)}{\partial \mathbf{w}^\top \mathbf{x}^i} \boxed{\frac{\partial \mathbf{w}^\top \mathbf{x}^i}{\partial w_k}} \longrightarrow \boxed{x_k}$$

Backpropagation: example 2

$$\frac{\partial J^i(\mathbf{w})}{\partial w_k} = \frac{\partial J^i(\mathbf{w})}{\partial f_\sigma(\mathbf{w}^\top \mathbf{x}^i)} \boxed{\frac{\partial f_\sigma(\mathbf{w}^\top \mathbf{x}^i)}{\partial \mathbf{w}^\top \mathbf{x}^i}} \boxed{\frac{\partial \mathbf{w}^\top \mathbf{x}^i}{\partial w_k}}$$

$$\boxed{\frac{\partial \mathbf{w}^\top \mathbf{x}^i}{\partial w_k}} \rightarrow \boxed{x_k}$$

$$\boxed{\frac{\partial f_\sigma(\mathbf{w}^\top \mathbf{x}^i)}{\partial \mathbf{w}^\top \mathbf{x}^i}} \rightarrow \boxed{f'_\sigma(\mathbf{w}^\top \mathbf{x}^i)}$$

Backpropagation: example 2

$$\frac{\partial J^i(\mathbf{w})}{\partial w_k} = \frac{\partial J^i(\mathbf{w})}{\partial f_\sigma(\mathbf{w}^\intercal \mathbf{x}^i)} \frac{\partial f_\sigma(\mathbf{w}^\intercal \mathbf{x}^i)}{\partial \mathbf{w}^\intercal \mathbf{x}^i} \frac{\partial \mathbf{w}^\intercal \mathbf{x}^i}{\partial w_k}$$

$$x_k^i$$

$$f'_\sigma(\mathbf{w}^\intercal \mathbf{x}^i)$$

$$\frac{\partial J^i(\mathbf{w})}{\partial f_\sigma(\mathbf{w}^T \mathbf{x}^i)} = \frac{\partial (y^i - f_\sigma(\mathbf{w}^T \mathbf{x}^i))^2}{\partial f_\sigma(\mathbf{w}^T \mathbf{x}^i)} = -2(y^i - f_\sigma(\mathbf{w}^T \mathbf{x}^i))$$

Backpropagation: example 2

$$\frac{\partial J^i(\mathbf{w})}{\partial w_k} = \boxed{\frac{\partial J^i(\mathbf{w})}{\partial f_\sigma(\mathbf{w}^\top \mathbf{x}^i)}}\ \boxed{\frac{\partial f_\sigma(\mathbf{w}^\top \mathbf{x}^i)}{\partial \mathbf{w}^\top \mathbf{x}^i}}\ \boxed{\frac{\partial \mathbf{w}^\top \mathbf{x}^i}{\partial w_k}}$$

$$\boxed{x_k^i}$$

$$\boxed{f'_\sigma(\mathbf{w}^\top \mathbf{x}^i)}$$

$$\boxed{\frac{\partial J^i(\mathbf{w})}{\partial f_\sigma(\mathbf{w}^T \mathbf{x}^i)} = \frac{\partial (y^i - f_\sigma(\mathbf{w}^T \mathbf{x}^i))^2}{\partial f_\sigma(\mathbf{w}^T \mathbf{x}^i)} = -2(y^i - f_\sigma(\mathbf{w}^T \mathbf{x}^i))}$$

$$\boxed{\frac{\partial J^i(\mathbf{w})}{\partial w_k} = -2(y^i - f_\sigma(\mathbf{w}^T \mathbf{x}^i)) f'_\sigma(\mathbf{w}^T \mathbf{x}^i) x_k^i}$$

This was for a single unit and a single sample

What to do for using all the samples?

What to do when we have multiple units?

UniGe | MaLGa

Training a [Deep] Neural Network

A parenthesis on Gradient Descent algorithms

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^P} J(S;\mathbf{w})$$

$$\mathbf{w}_0 = 0$$

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma \nabla J(S;\mathbf{w}_{t-1})$$

Training a [Deep] Neural Network

A parenthesis on Gradient Descent algorithms

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^P} J(S;\mathbf{w})$$

$$\mathbf{w}_0 = 0$$

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma \nabla J(S;\mathbf{w}_{t-1})$$

Batch Gradient Descent (GD)

Smoothly converging towards the optimum, but computationally demanding

Training a [Deep] Neural Network

A parenthesis on Gradient Descent algorithms

$$\mathbf{w}^* = \arg \min_{\mathbf{w} \in \mathbb{R}^P} J(S; \mathbf{w})$$

$$\mathbf{w}_0 = 0$$

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma \nabla J(S; \mathbf{w}_{t-1})$$

Batch Gradient Descent (GD)

Smoothly converging towards the optimum, but computationally demanding

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma \nabla J(x^i; \mathbf{w}_{t-1})$$

Stochastic Gradient Descent (SGD)

known to converge well in practice: empirically, it provides a better exploration of the space

May require many iterations

Training a [Deep] Neural Network

A parenthesis on Gradient Descent algorithms

$$\mathbf{w}^* = \arg\min_{\mathbf{w} \in \mathbb{R}^P} J(S; \mathbf{w})$$

$$\mathbf{w}_0 = 0$$

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma \nabla J(S; \mathbf{w}_{t-1})$$

Batch Gradient Descent (DG)

Smoothly converging towards the optimum, but computationally demanding

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma \nabla J(x^i; \mathbf{w}_{t-1})$$

Stochastic Gradient Descent (SGD)

Known to converge well in practice: empirically, it provides a better exploration of the space

May require many iterations

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \gamma \nabla J(x^{[i:i+B]}; \mathbf{w}_{t-1})$$

Mini-batch Gradient Descent

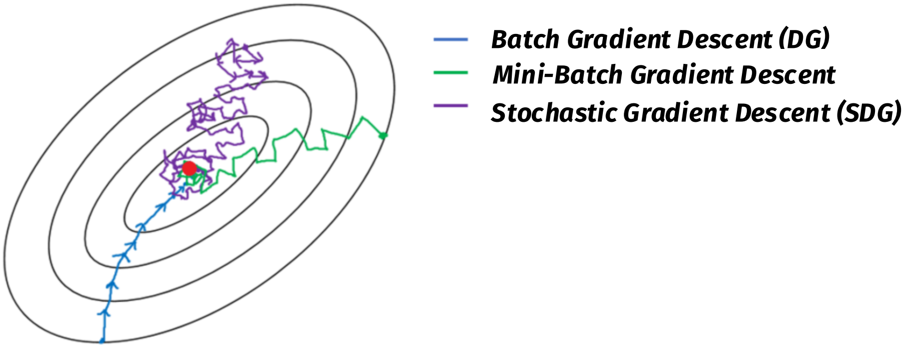
A good compromise

B is the mini-batch size (when B=1 you go back to SGD)

UniGe | MaLGa

Training a [Deep] Neural Network

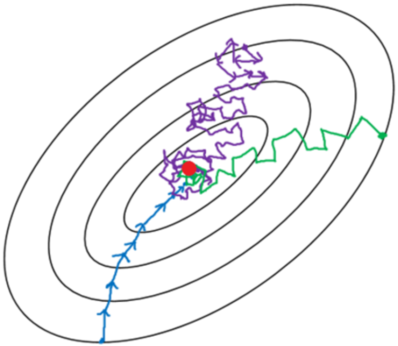
A parenthesis on Gradient Descent algorithms



Picture from https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

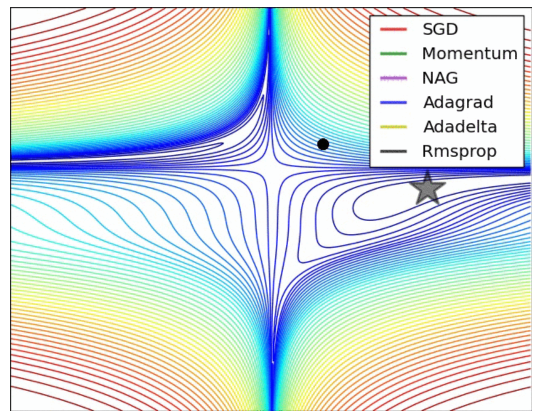
Training a [Deep] Neural Network

A parenthesis on Gradient Descent algorithms



Batch Gradient Descent (DG)
Mini-Batch Gradient Descent
Stochastic Gradient Descent (SDG)

There are also alternative optimization strategies...



Picture from https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

Training a [Deep] Neural Network

A parenthesis on terminology

Given the complexity of deep models, it is necessary to go through the entire training set more than once at training time

One **epoch** is when the dataset is entirely passed forward and backwards through the architecture

The **[mini-]batch** size is the number of training samples in a mini-batch

An **iteration** is the number of batches needed to complete one epoch

→ Number of epochs and batch size are hyper-parameters of the model, usually set a priori

Training a [Deep] Neural Network

Model selection

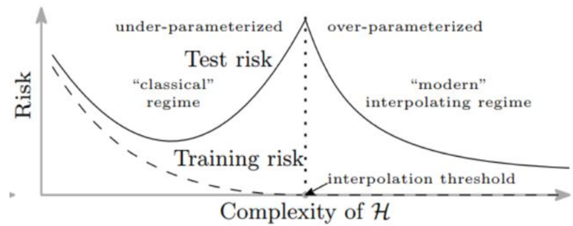
What does it mean to perform model selection with this family of methods?

Training a [Deep] Neural Network

Model selection

What does it mean to perform model selection with this family of methods?

In principle, one could apply the same protocol adopted for more classical methods



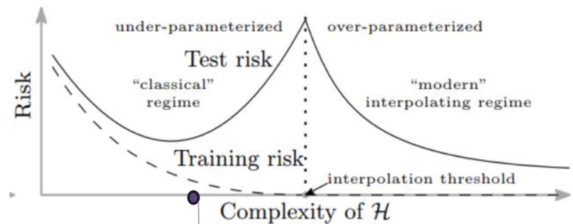
What's the complexity here?

Training a [Deep] Neural Network

Model selection

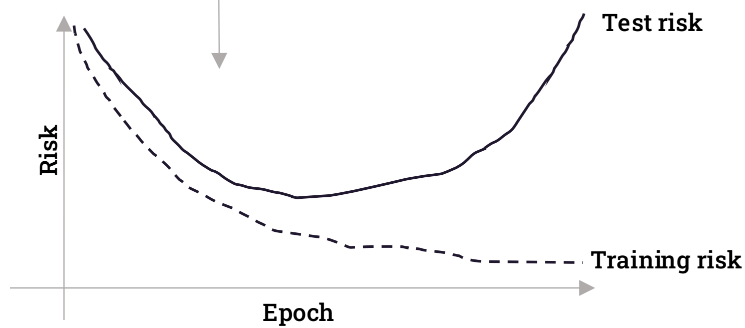
What does it mean to perform model selection with this family of methods?

In principle, one could apply the same protocol adopted for more classical methods



What's the complexity here?

In practice, we fix the model (i.e. a certain complexity) and, often, also the values of many (if not all) hyper-parameters; we reason on the training-validation error vs the number of epochs

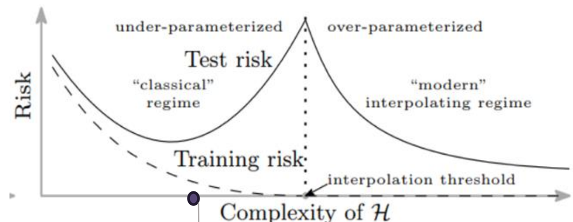


Training a [Deep] Neural Network

Model selection

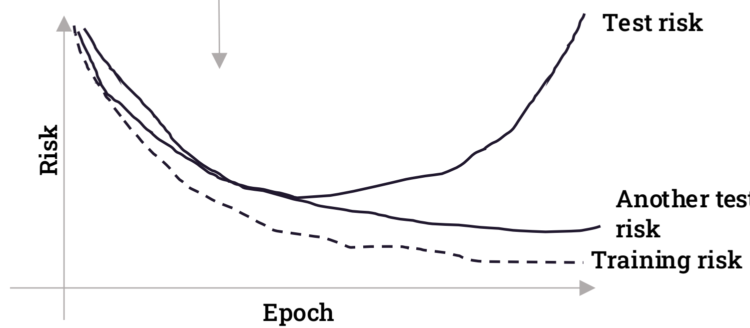
What does it mean to perform model selection with this family of methods?

In principle, one could apply the same protocol adopted for more classical methods



What's the complexity here?

In practice, we fix the model (i.e. a certain complexity) and, often, also the values of many (if not all) hyper-parameters; we reason on the training-validation error vs the number of epochs



Training a [Deep] Neural Network

How to prevent overfitting

- Classical approaches can still be adopted:
    - Adding regularization terms
    - Using Early Stopping criterion

- One alternative (popular) option: Dropout

Training a [Deep] Neural Network

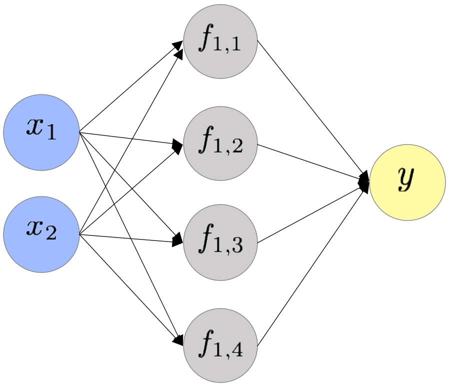
Dropout

- Overfitted DNN models tend to suffer from a problem of co-adaptation: models weights are adjusted co-linearly to learn the model training data too well… so the model doesn't generalize
- With limited training data weights likely become adapted to them, and the model doesn't generalize
- Bagging and Dropout are ways to break this co-adaptation

Training a [Deep] Neural Network

Bagging: intuition

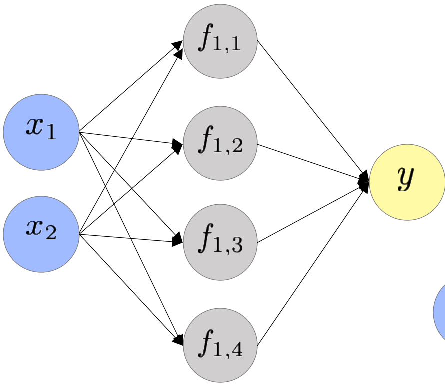
P = 0.5

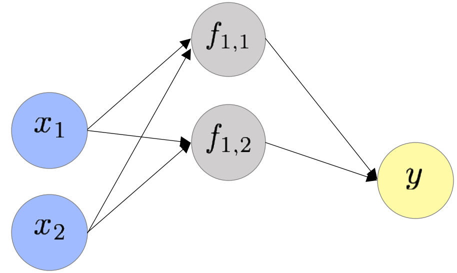


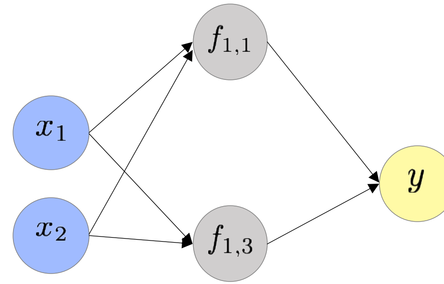
Training a [Deep] Neural Network

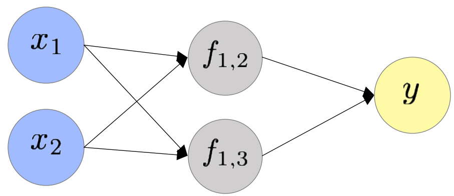
Bagging: intuition

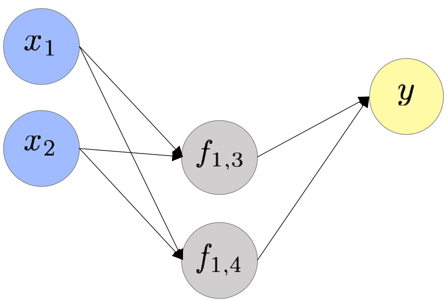
P = 0.5

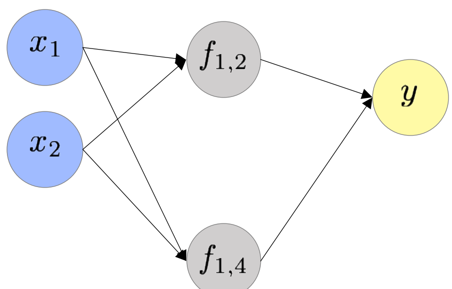


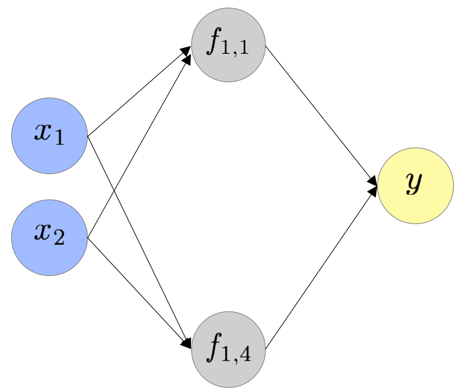












Training a [Deep] Neural Network

Bagging and Dropout

With Bagging

- Each sub-network is trained and evaluated on each test sample
- The final prediction is given by the votes of all models

Bagging may be computationally very expensive

Dropout is a way to approximate the same behaviour

→ At each step of the optimisation, some fraction of weights are dropped out of each layer (=set to 0)

More details about deep networks training

Loss functions

- For regression problems you may use the square loss

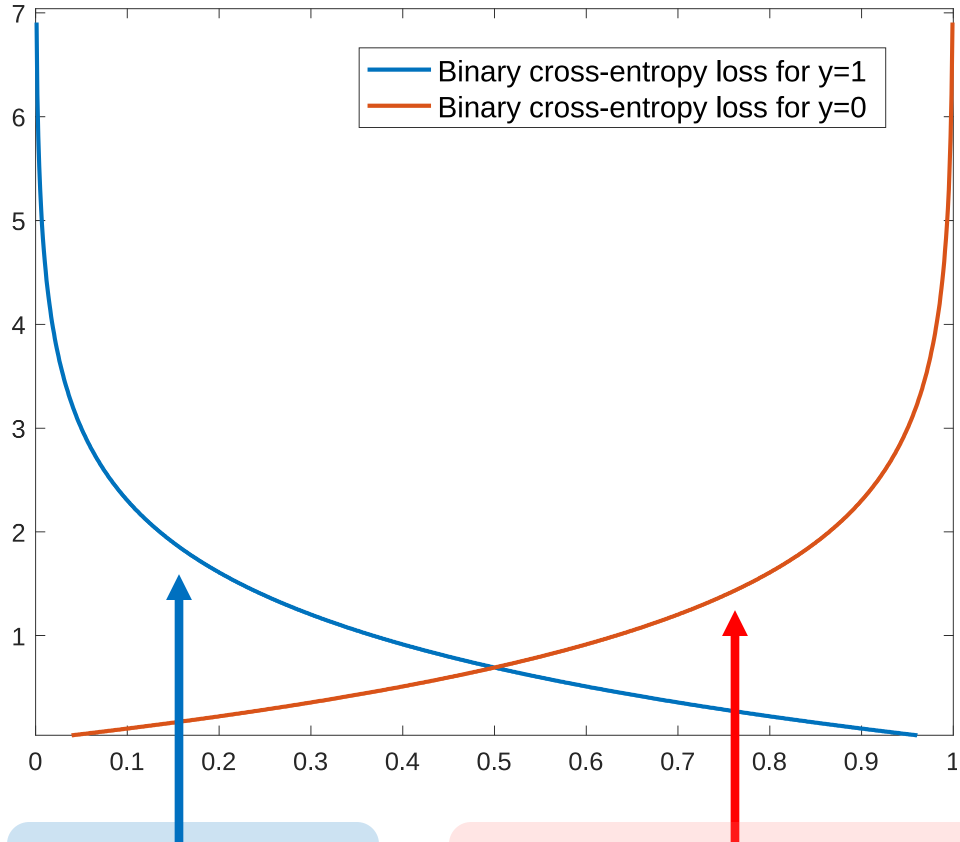
$$\ell(y_i, \hat{f}(x_i)) = (y_i - \hat{f}(x_i))^2$$

- For classifiers?
  - Binary classification: cross-entropy log

$$\ell(y_i, \hat{f}(x_i)) = -y_i \cdot log(\hat{f}(x_i)) - (1 - y_i) \cdot log(1 - \hat{f}(x_i))$$

	$\hat{f}(x_i) = 0$	$\hat{f}(x_i) = 1$
$y_i = 0$	0	inf
$y_i = 1$	inf	0

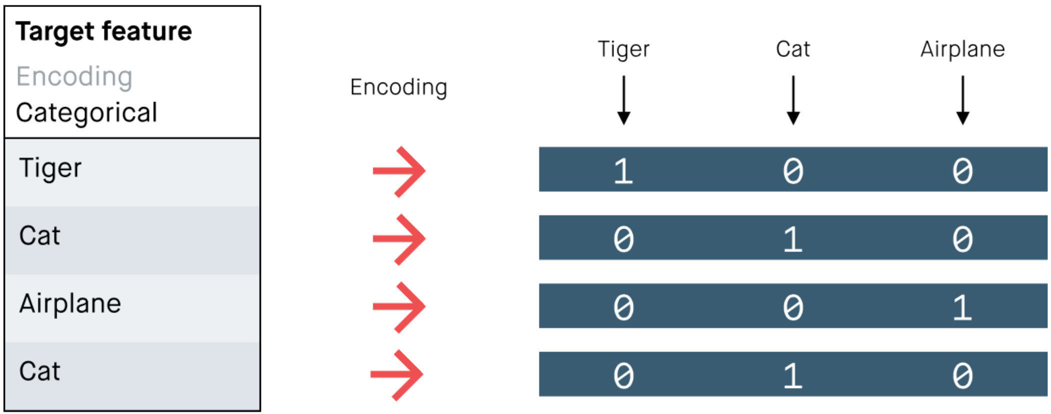
Loss functions



$$\ell(y_i, \hat{f}(x_i)) = y_i \cdot log(\hat{f}(x_i)) + (1 - y_i) \cdot log(1 - \hat{f}(x_i))$$

Loss functions

Multi-class classification: categorical cross-entropy loss



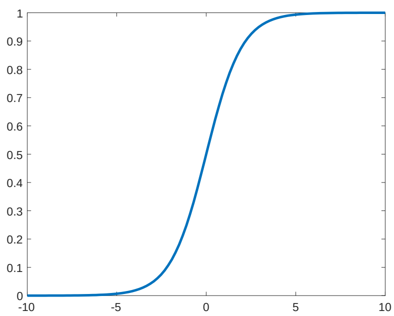
$$\ell(y_i, \hat{f}(x_i)) = -\frac{1}{M}\sum_{k=1}^{M}(y_i^k \cdot log(\hat{f}(x_i)^k) + (1 - y_i^k) \cdot log(1 - \hat{f}(x_i)^k))$$

https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy

Activation funcions for output units

For binary classification

A good choice is the sigmoid output → it «enforces» the binary values



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

For multi-class classification

It is covenient to use the Softmax operator, that allows us to «represent» the probability distribution over the M different classes

$$\mathbf{z} = \mathbf{W}^T\mathbf{h} + \mathbf{b}$$

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{M} e^{z_k}}$$

Some terminology

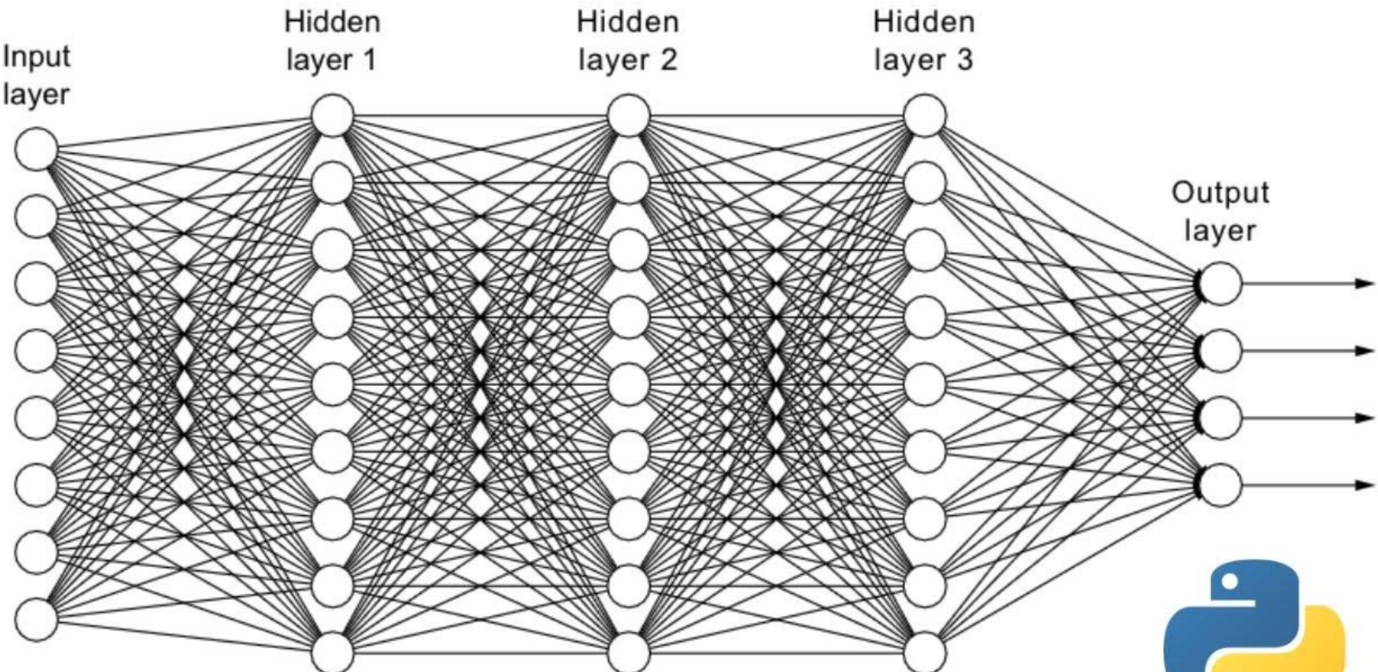
- One **epoch** is when the entire dataset is passed forward and backward through the neural network only once (multiple times are usually needed)
- The **batch** size is the number of training examples in a mini-batch
- An **iteration** is the number of batches needed to complete one epoch
- Ex. For a dataset of 10000 sample with mini-batch size 1000, 10 iterations will complete 1 epoch

Some considerations

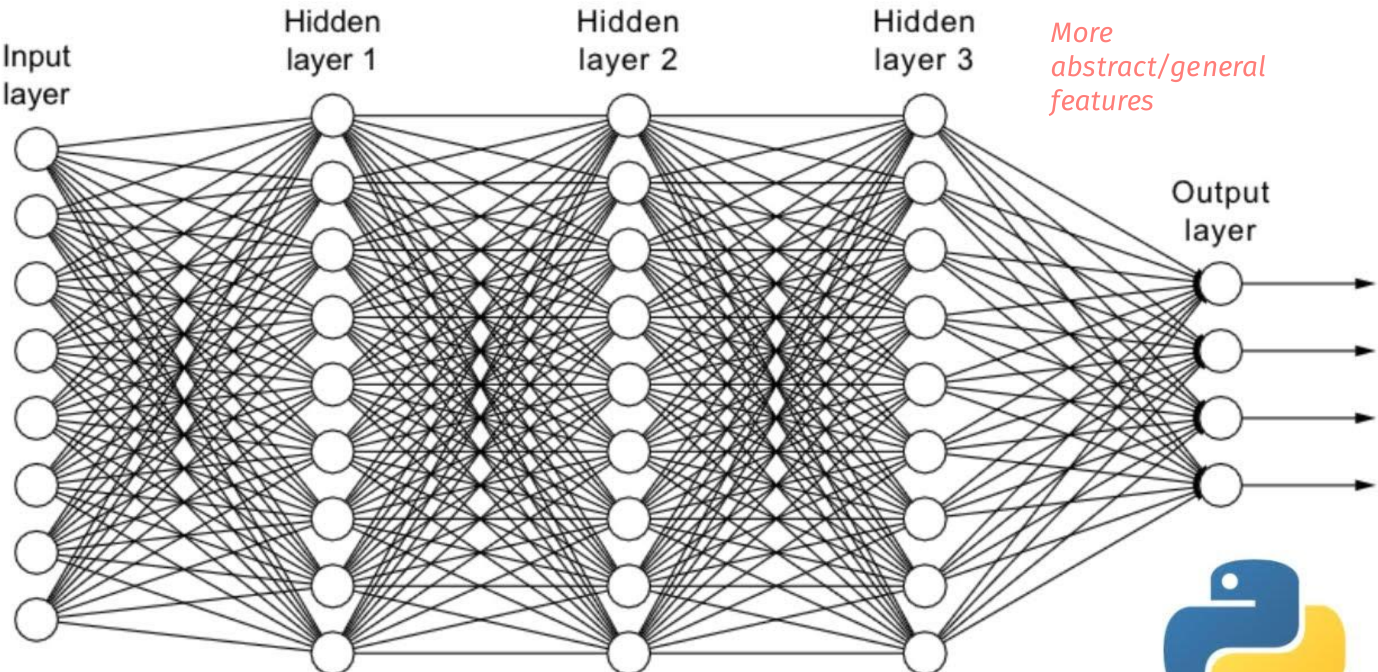
Success with a price

- Lots of parameters → lots of data → strong demand in terms of computation
- In many applications it’s not easy to have data, in general it’s not easy to annotate data
- Difficult to interpret the models
- Hard to encode prior knowledge
- The need of rethinking generalization?

How to interpret a deep architecture?



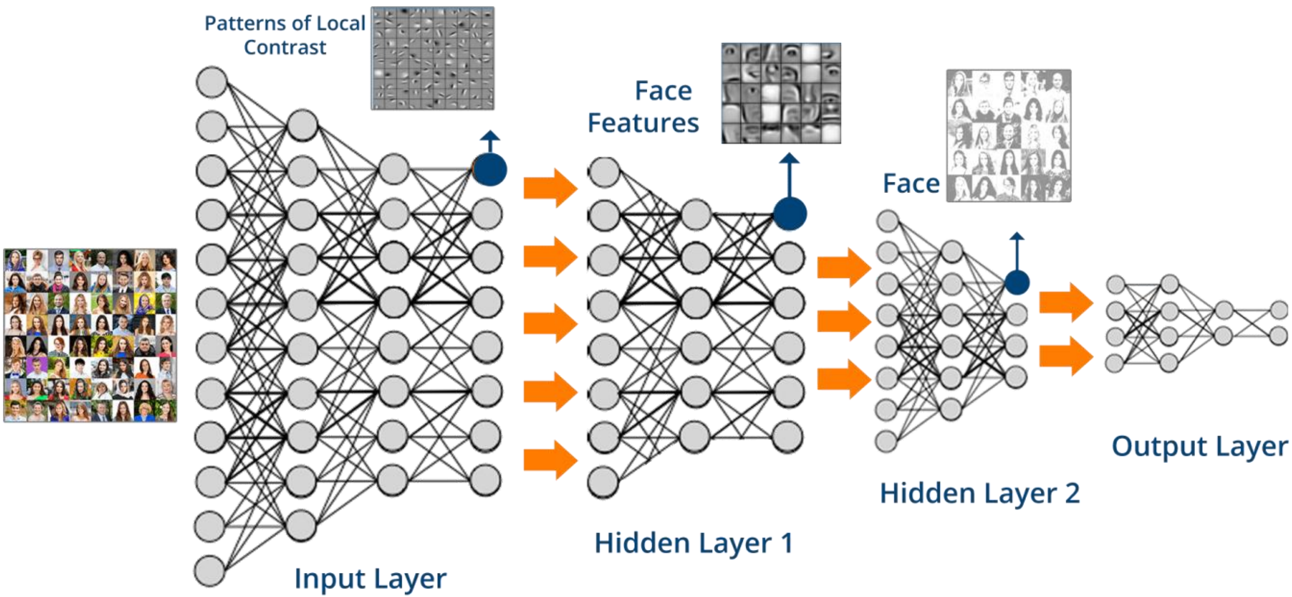
How to interpret a deep architecture?



More abstract/general features

Low-level features

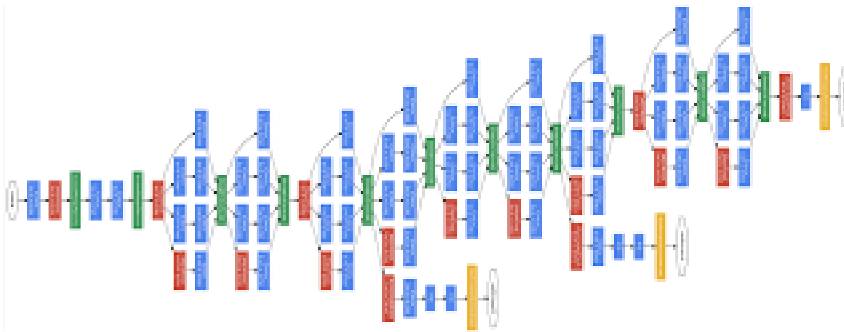
How to interpret a deep architecture?

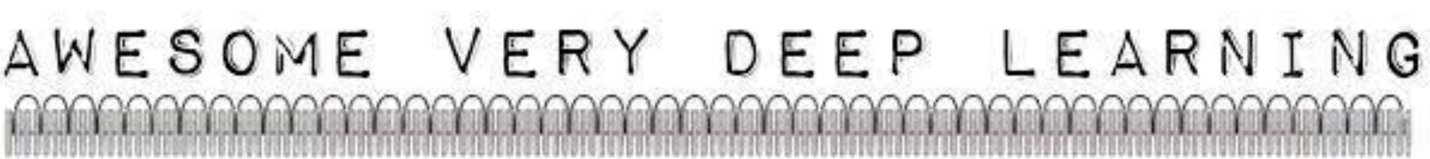


How to interpret a deep architecture?

But how to understand how a NN works? It’s an open question...

And things may become even more complicated...





UniGe

MaLGa