Lectures on Machine Learning

Lecture 2: from parameter learning to non-linear models

Stefano Carrazza

TAE2018, 2-15 September 2018

European Organization for Nuclear Research (CERN)

Acknowledgement: This project has received funding from HICCUP ERC Consolidator grant (614577) and by the European Unions Horizon 2020 research and innovation programme under grant agreement no. 740006.

Outline

Lecture 1 (yesterday)

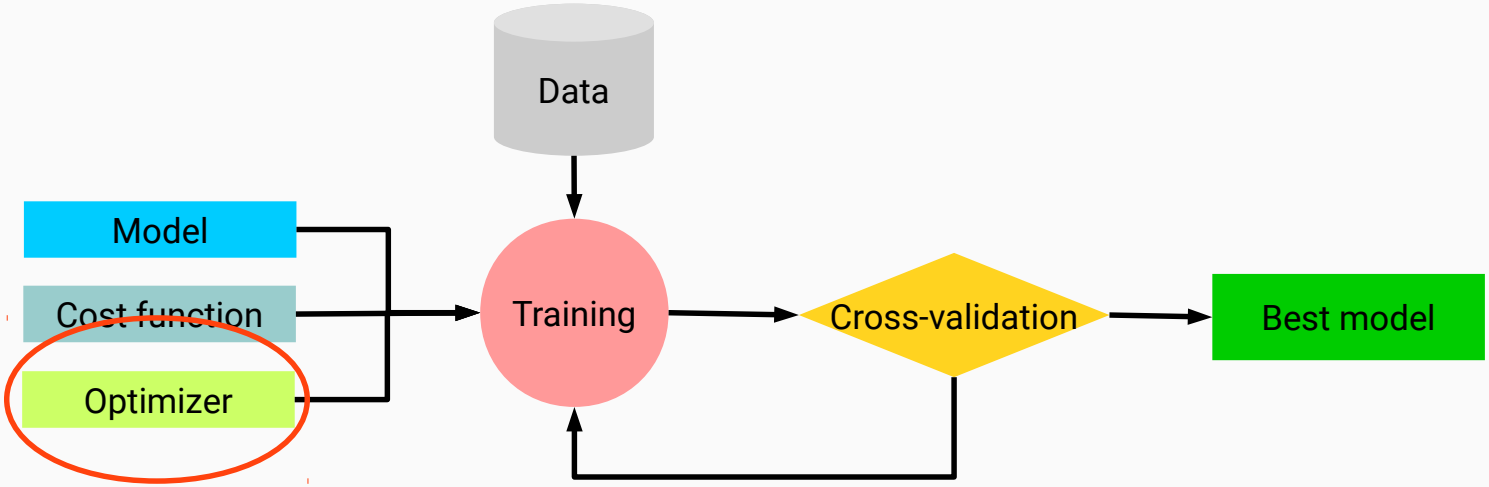
- Artificial intelligence
- Machine learning
- Model representation
- Metrics

Lecture 2 (today)

- Parameter learning
- Non-linear models
- Beyond neural networks
- Clustering

Parameter learning

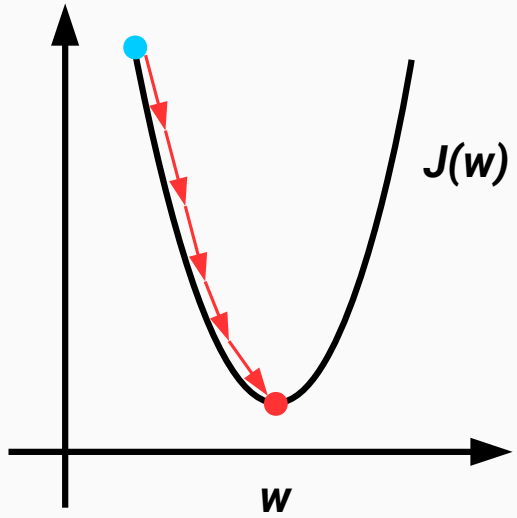
Parameter learning



Cost function minimization

Optimization algorithms **minimize an objective function**, $J(\boldsymbol{w})$, that depends on the model internal learnable parameters $\boldsymbol{w}$.

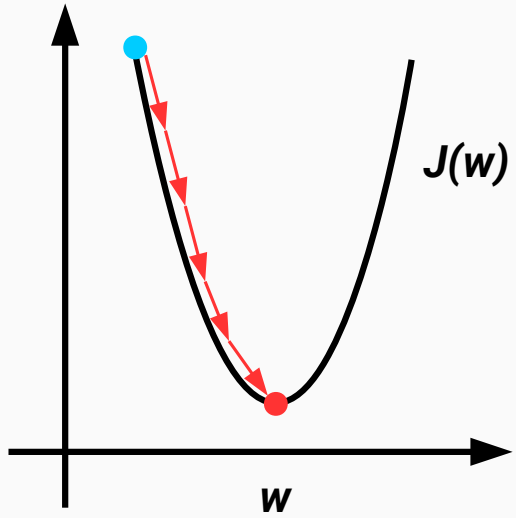
$$\arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$$



Cost function minimization

Optimization algorithms **minimize an objective function**, $J(\boldsymbol{w})$, that depends on the model internal learnable parameters $\boldsymbol{w}$.

$$\arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$$



The most popular techniques are:

- normal equations (least squares)
- derivative based optimization
- evolutionary algorithms

The choice of a technique depends on the model and problem employed.

Normal equations

The normal equation is a method to solve for $\boldsymbol{w}$ analytically.

- it is employed in linear and non-linear least squares optimization.
- it is fast for small models with few features, otherwise it can be computationally intensive and slow.

Normal equations example

Example: multivariate linear regression.

Suppose we have m training examples and n features in a matrix X, size (m, n), and the observed values $\boldsymbol{y}$ we have to solve the system:

$$X\boldsymbol{w} = \boldsymbol{y}$$

How to solve it when the system if overdetermined?

Normal equations example

Example: multivariate linear regression.

Suppose we have m training examples and n features in a matrix X, size (m, n), and the observed values $\boldsymbol{y}$ we have to solve the system:

$$X\boldsymbol{w} = \boldsymbol{y}$$

How to solve it when the system if overdetermined?

1. Define the cost function for the problem:

$$J(\boldsymbol{w}) = ||\boldsymbol{y} - X\boldsymbol{w}||^2$$

Normal equations example

Example: multivariate linear regression.

Suppose we have m training examples and n features in a matrix X, size (m, n), and the observed values $\boldsymbol{y}$ we have to solve the system:

$$X\boldsymbol{w} = \boldsymbol{y}$$

How to solve it when the system if overdetermined?

1. Define the cost function for the problem:

$$J(\boldsymbol{w}) = ||\boldsymbol{y} - X\boldsymbol{w}||^2$$

2. Compute and impose:

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = \mathbf{0}$$

Normal equations example

Example: multivariate linear regression.

Suppose we have m training examples and n features in a matrix X, size (m, n), and the observed values $\boldsymbol{y}$ we have to solve the system:

$$X\boldsymbol{w} = \boldsymbol{y}$$

How to solve it when the system if overdetermined?

1. Define the cost function for the problem:

$$J(\boldsymbol{w}) = ||\boldsymbol{y} - X\boldsymbol{w}||^2$$

2. Compute and impose:

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = \mathbf{0}$$

3. We obtain the solution:

$$\boldsymbol{w} = (X^T X)^{-1} X^T \boldsymbol{y} = X^+ \boldsymbol{y}$$

with X^+ is the pseudoinverse of X.

Derivative based optimization

More general optimization algorithms based on derivatives:

- **First order optimization algorithms:** uses the gradient of the cost function with respect to the parameters in a iteractive procedure.
  $\rightarrow$ gradient descent algorithms.

Derivative based optimization

More general optimization algorithms based on derivatives:

- **First order optimization algorithms:** uses the gradient of the cost function with respect to the parameters in a iteractive procedure.
  $\rightarrow$ gradient descent algorithms.
- **Second order optimization algorithms:** uses the Hessian of the cost function and takes care of the curvature of surface.
  $\rightarrow$ if the Hessian is known it may be faster than gradient descent,
  $\rightarrow$ otherwise slow due to the Hessian evaluation.

Derivative based optimization

More general optimization algorithms based on derivatives:

- **First order optimization algorithms:** uses the gradient of the cost function with respect to the parameters in a iteractive procedure.
  $\rightarrow$ gradient descent algorithms.
- **Second order optimization algorithms:** uses the Hessian of the cost function and takes care of the curvature of surface.
  $\rightarrow$ if the Hessian is known it may be faster than gradient descent,
  $\rightarrow$ otherwise slow due to the Hessian evaluation.

In practice Gradient Descent is the most popular technique in ML.

Which method?

Q: normal equation or derivative based?

Suppose we have $\boldsymbol{m}$ training examples and $\boldsymbol{n}$ features.

Normal equation

- ✓ no parameters to tune
- ✓ no iterations
- ✗ slow if n is large
- ✗ requires $(X^T X)^{-1}, \mathcal{O}(n^3)$

Which method?

Q: normal equation or derivative based?

Suppose we have $\boldsymbol{m}$ training examples and $\boldsymbol{n}$ features.

Normal equation	Gradient descent
✓ no parameters to tune	✓ efficient when n is large
✓ no iterations	✓ easy to implement/use
✗ slow if n is large	✗ requires iterations
✗ requires $(X^T X)^{-1}, \mathcal{O}(n^3)$	✗ requires parameter tune

Gradient descent idea

Basic idea:

Assuming we want to minimize $J(\boldsymbol{w})$ where $\boldsymbol{w}$ is a vector of parameters:

- select a initial solution vector $\boldsymbol{w}$,
- change the $\boldsymbol{w}$ to reduce $J(\boldsymbol{w})$

Repeat until a minimum of $J(\boldsymbol{w})$ is reached.

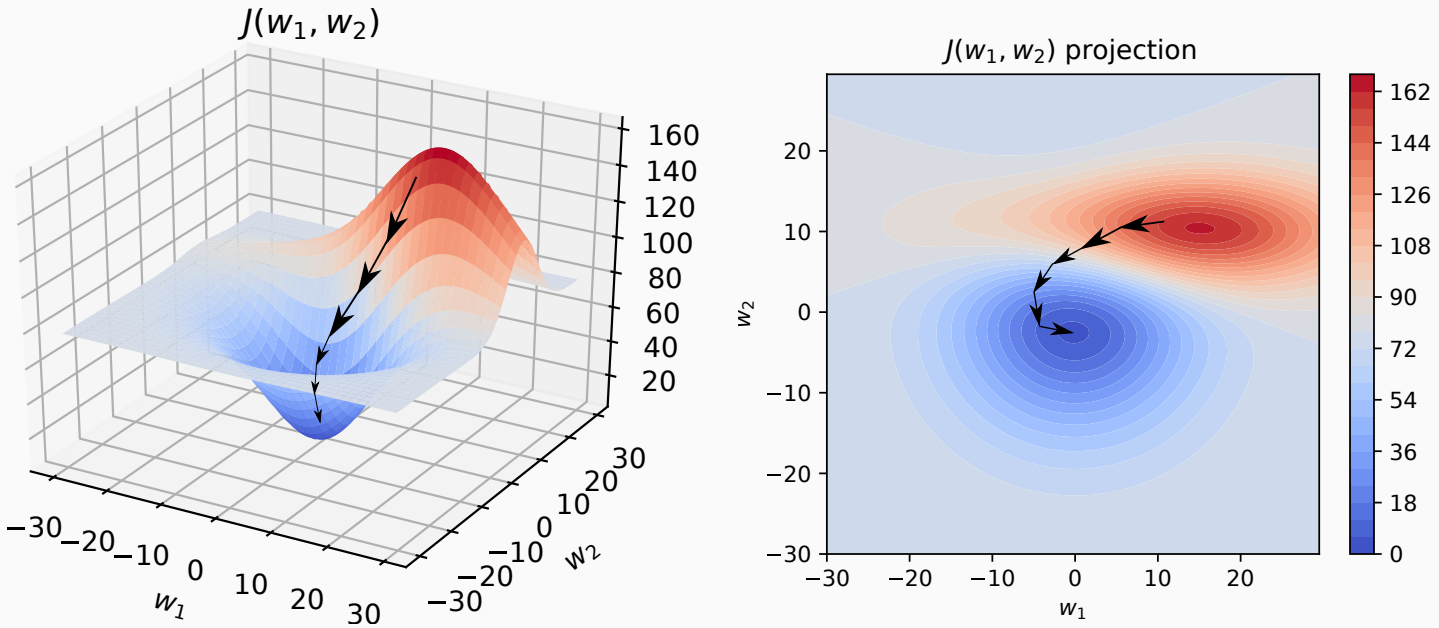
Gradient descent idea

Basic idea:

Assuming we want to minimize $J(\boldsymbol{w})$ where $\boldsymbol{w}$ is a vector of parameters:

- select a initial solution vector $\boldsymbol{w}$,
- change the $\boldsymbol{w}$ to reduce $J(\boldsymbol{w})$

Repeat until a minimum of $J(\boldsymbol{w})$ is reached.



Gradient descent algorithm

Simultaneously for each parameter in $\boldsymbol{w}$ repeat until convergence:

$$w_i := w_i - \eta \frac{\partial}{\partial w_i} J(\boldsymbol{w})$$

where

- η is the learning rate.

Gradient descent algorithm

Simultaneously for each parameter in $\boldsymbol{w}$ repeat until convergence:

$$w_i := w_i - \eta \frac{\partial}{\partial w_i} J(\boldsymbol{w})$$

where

- η is the learning rate.
- $\eta \geq 0$, it can be a fixed number, because the gradient term will automatically compensate with smaller steps: $\nabla_w J|_{w \to w_{\text{best}}} \to 0$.

Gradient descent algorithm

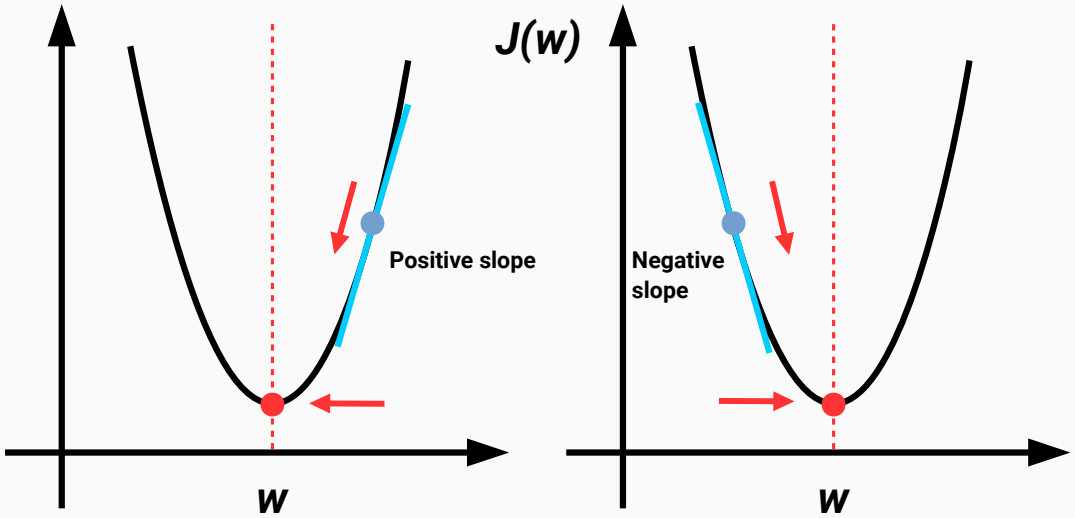
Simultaneously for each parameter in $\boldsymbol{w}$ repeat until convergence:

$$w_i := w_i - \eta \frac{\partial}{\partial w_i} J(\boldsymbol{w})$$

where

- η is the learning rate.
- $\eta \geq 0$, it can be a fixed number, because the gradient term will automatically compensate with smaller steps: $\nabla_w J|_{w \to w_{\text{best}}} \to 0$.

Why the negative sign in term $-\eta$? (example in 1D)

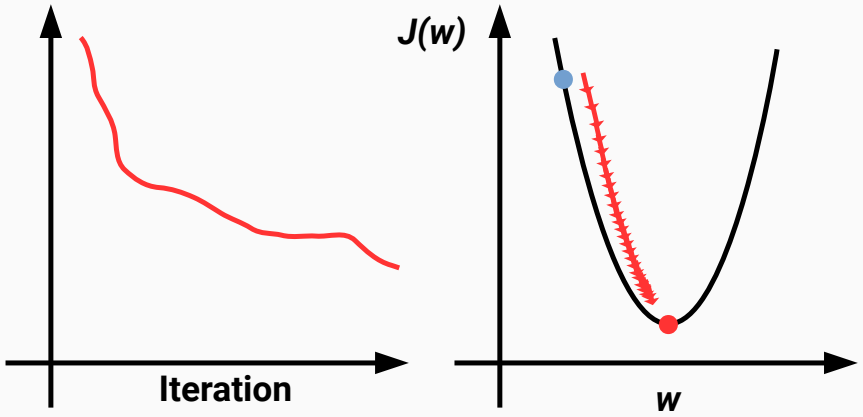


- if $\nabla_w J(w) > 0$ then w decreases
- if $\nabla_w J(w) < 0$ then w increases

Gradient descent and learning rate

The η is another example of hyperparameter which requires tune.

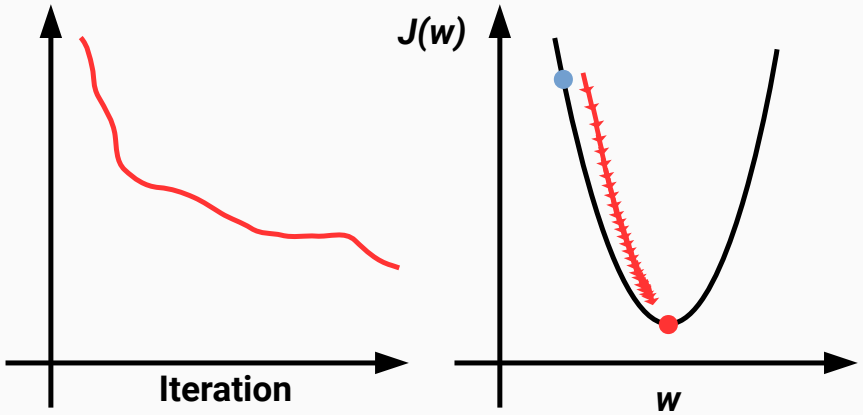
- if η is **too small**, gradient descent can be slow.



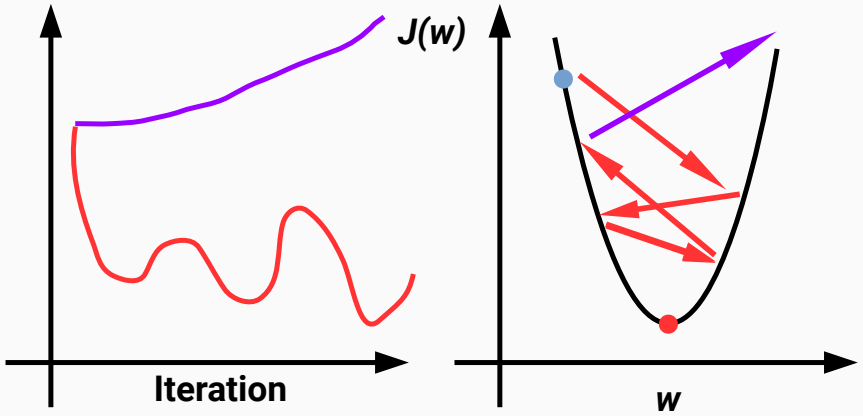
Gradient descent and learning rate

The η is another example of hyperparameter which requires tune.

- if η is **too small**, gradient descent can be slow.



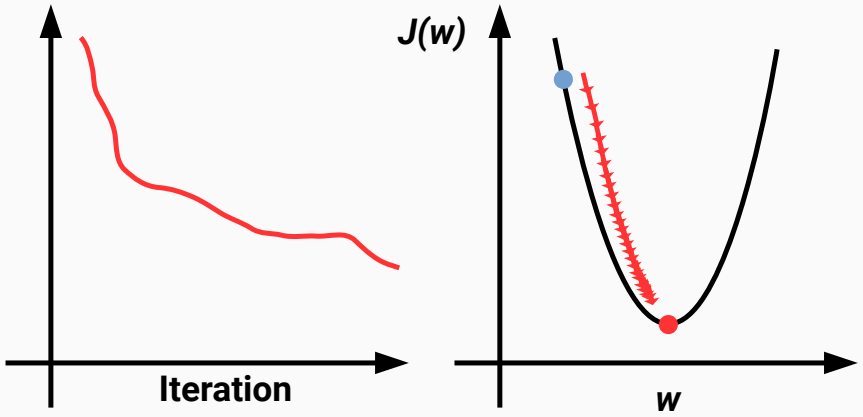
- if η is **too large**, gradient descent may fail to converge or diverge.



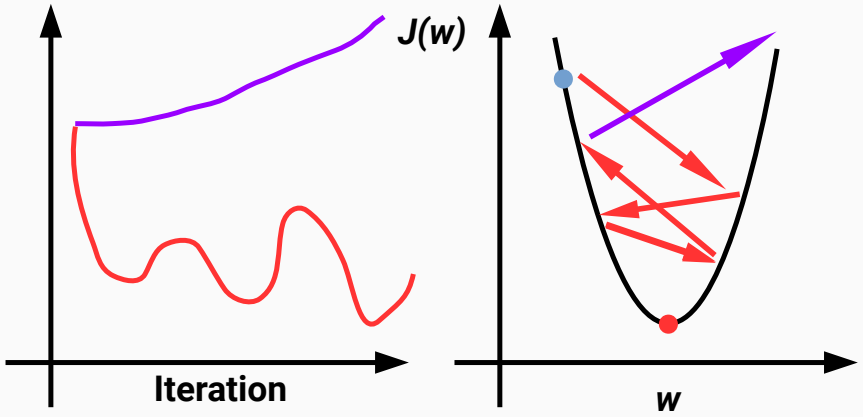
Gradient descent and learning rate

The η is another example of hyperparameter which requires tune.

- if η is **too small**, gradient descent can be slow.



- if η is **too large**, gradient descent may fail to converge or diverge.



- Practical hint, start with small η values and then increase slowly.

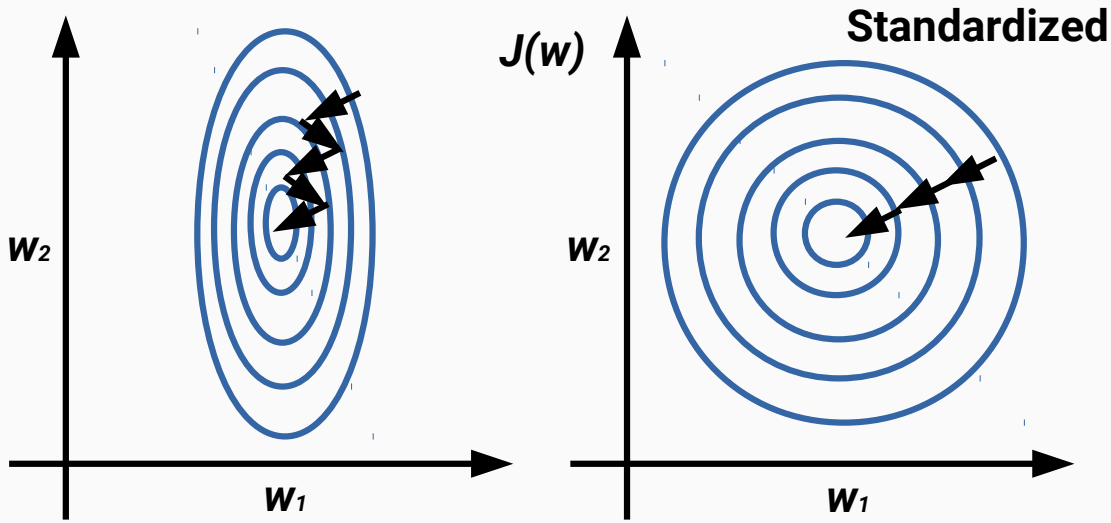
Gradient descent and feature scaling

Another practical hint: **feature scaling**.

Make sure the input features x_i are in a similar scale, *e.g.* standardization:

$$x_i := \frac{x_i - \mu_{x_i}}{\sigma_{x_i}}$$

where μ_{x_i} and σ_{x_i} are the mean and standard deviation respectively.



Gradient descent variants

When performing gradient descent the cost function $J(\boldsymbol{w})$ is evaluated over the training data, *e.g.* for the MSE cost function:

$$\frac{\partial}{\partial \boldsymbol{w}} J(\boldsymbol{w}) = \frac{\partial}{\partial \boldsymbol{w}} \left(\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_{\boldsymbol{w}}(\boldsymbol{x}_i))^2 \right).$$

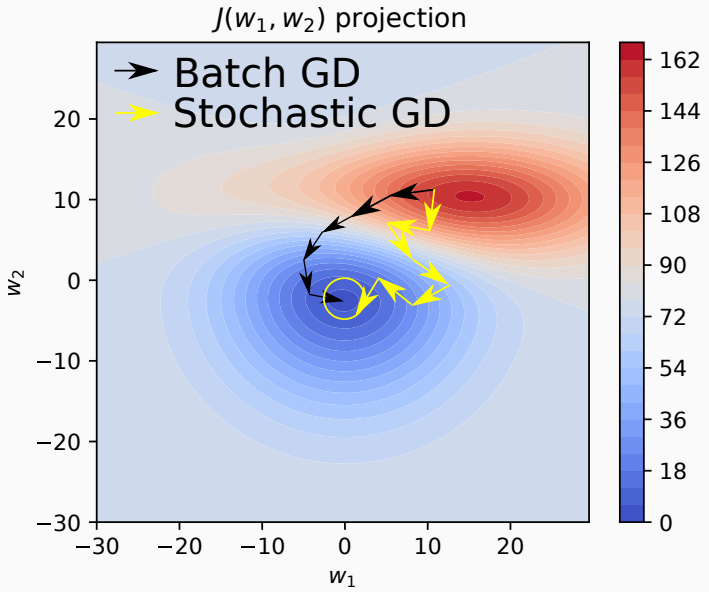
If the training data set is too large, there are gradient descent variations that may improve convergence in terms of speed and quality:

- Batch Gradient Descent: all training data points are evaluated in the cost function gradient at each iteration.

Gradient descent variants

- Stochastic Gradient Descent (SGD):
  1. randomly shuffle training examples,
  2. use 1 example at each iteration.

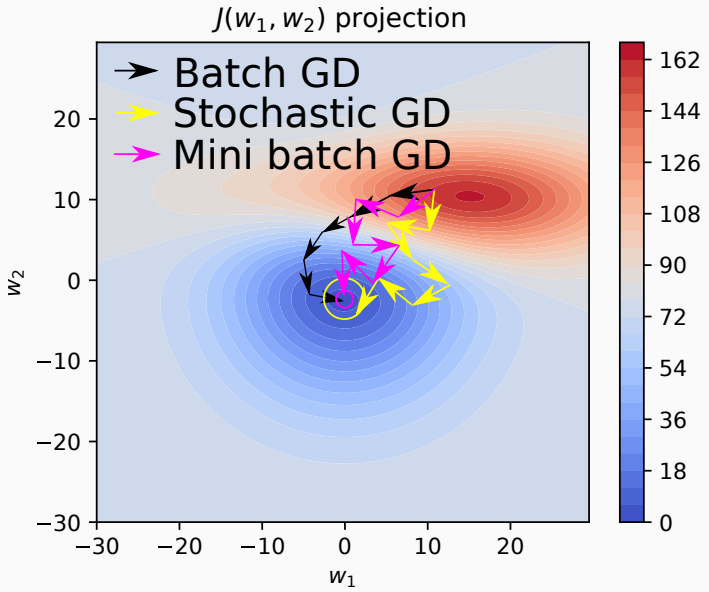
  Features:
  - parameters updates have high variance and cost function fluctuates.
  - helps to discover new and possibly better minima.
  - requires to slowly decrease the learning rate η to reduce fluctuations.



Gradient descent variants

- Mini Batch Gradient Descent:
  1. use a subset of size b (batch size) of examples in each iteration,
  2. use the batch set example at each iteration.

  Features:
  - takes the best from both previous methods,
  - reduces the variance in the parameter updates (stable convergence),
  - good for data parallelism, efficient for matrix operations



Gradient descent schemes

SGD has many improvements and extensions, for example:

- Momentum: it stores the update $\Delta \boldsymbol{w}$ at each iteration and update parameters following:

$$\boldsymbol{w} := \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) + \alpha \Delta \boldsymbol{w}$$

Gradient descent schemes

SGD has many improvements and extensions, for example:

- Momentum: it stores the update $\Delta \boldsymbol{w}$ at each iteration and update parameters following:

$$\boldsymbol{w} := \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) + \alpha \Delta \boldsymbol{w}$$

- Root Mean Square Propagation (RMSProp): it introduces an adaptive learning rate for each parameter:

$$\boldsymbol{w} := \boldsymbol{w} - \frac{\eta}{\sqrt{v(\boldsymbol{w}, ite)}} \nabla_{\boldsymbol{w}} J(\boldsymbol{w})$$

Gradient descent schemes

SGD has many improvements and extensions, for example:

- Momentum: it stores the update $\Delta \boldsymbol{w}$ at each iteration and update parameters following:

$$\boldsymbol{w} := \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) + \alpha \Delta \boldsymbol{w}$$

- Root Mean Square Propagation (RMSProp): it introduces an adaptive learning rate for each parameter:

$$\boldsymbol{w} := \boldsymbol{w} - \frac{\eta}{\sqrt{v(\boldsymbol{w}, ite)}} \nabla_{\boldsymbol{w}} J(\boldsymbol{w})$$

- Others: Averaging, AdaGrad, Adam, etc...

Gradient descent schemes

SGD has many improvements and extensions, for example:

- Momentum: it stores the update $\Delta \boldsymbol{w}$ at each iteration and update parameters following:

$$\boldsymbol{w} := \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) + \alpha \Delta \boldsymbol{w}$$

- Root Mean Square Propagation (RMSProp): it introduces an adaptive learning rate for each parameter:

$$\boldsymbol{w} := \boldsymbol{w} - \frac{\eta}{\sqrt{v(\boldsymbol{w}, ite)}} \nabla_{\boldsymbol{w}} J(\boldsymbol{w})$$

- Others: Averaging, AdaGrad, Adam, etc...

All these schemes and respective parameters can be considered as extra hyperparameters to tune.

Examples of second order optimization

Popular examples of second order optimization algorithms:

- Newton's method: an iterative method based on Taylor expansion.
  Example in 1D: consider the Taylor expansion

$$J_T(w) = J_T(w_n + \Delta w) \approx J(w_n) + J'(w_n)\Delta w + \frac{1}{2}J''(w_n)\Delta w^2$$

  We aim to find Δw which satisfies:

$$\nabla_{\Delta w} J_T(w_n + \Delta w) = J'(w_n) + J''(w_n)\Delta w = 0$$

$$w_{n+1} = w_n + \Delta w = w_n - \frac{J'(w_n)}{J''(w_n)}$$

  $w_1, w_2, \ldots$ will converge to a stationary point w^* where $J'(w^*) = 0$.

Examples of second order optimization

Popular examples of second order optimization algorithms:

- Newton's method: an iterative method based on Taylor expansion.
  Example in 1D: consider the Taylor expansion

$$J_T(w) = J_T(w_n + \Delta w) \approx J(w_n) + J'(w_n)\Delta w + \frac{1}{2}J''(w_n)\Delta w^2$$

  We aim to find Δw which satisfies:

$$\nabla_{\Delta w} J_T(w_n + \Delta w) = J'(w_n) + J''(w_n)\Delta w = 0$$

$$w_{n+1} = w_n + \Delta w = w_n - \frac{J'(w_n)}{J''(w_n)}$$

  $w_1, w_2, \ldots$ will converge to a stationary point w^* where $J'(w^*) = 0$.
  Generalization in N dimensions:

$$(\boldsymbol{H}J(\boldsymbol{w_n}))\boldsymbol{\Delta w} = -\nabla J(\boldsymbol{w}_n)$$

  where $\boldsymbol{H}$ is the Hessian matrix.

Examples of second order optimization

Popular examples of second order optimization algorithms:

- Quasi-newton methods: *i.e.* methods which optimizes even if the Hessian matrix is expensive or not available. The Taylor's series is:

$$J(\boldsymbol{w}_n + \Delta\boldsymbol{w}) \approx J(\boldsymbol{w}_n) + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_n)^T \Delta\boldsymbol{w} + \frac{1}{2}\Delta\boldsymbol{w}^T B \Delta\boldsymbol{w},$$

  where B is an approximation to the Hessian matrix and

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_n + \Delta\boldsymbol{w}) \approx \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_n) + B\Delta\boldsymbol{w},$$

  which produces the Newton step:

$$\Delta\boldsymbol{w} = -B^{-1}\nabla_{\boldsymbol{w}} J(\boldsymbol{w}_n).$$

  Some methods: BFGS, L-BFGS, DFP, Broyden.
  - differ by the choice of the solution to update B.

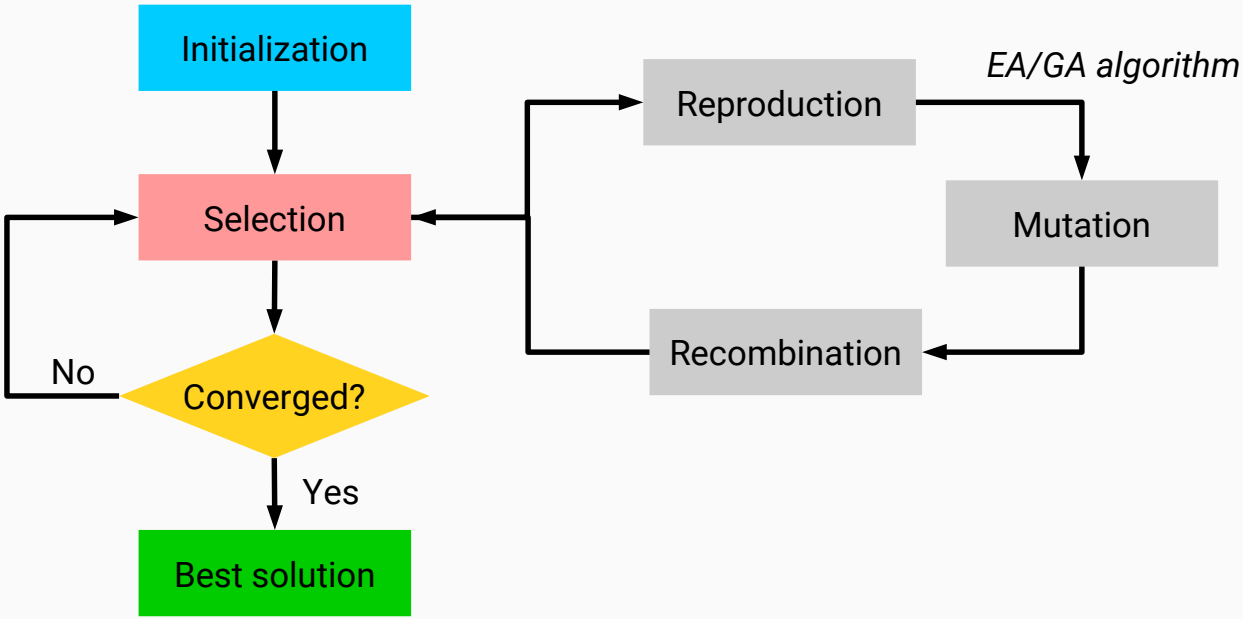
Popular in ML since the beginning of the deep learning era.

Evolutionary algorithms

Evolutionary algorithms (EA), inspired by biological evolution, is a generic population-based metaheuristic optimization algorithm.

Techniques in EA use mechanisms such as: reproduction, mutation, recombination, and selection.

Genetic algorithm is the most popular technique of EA.

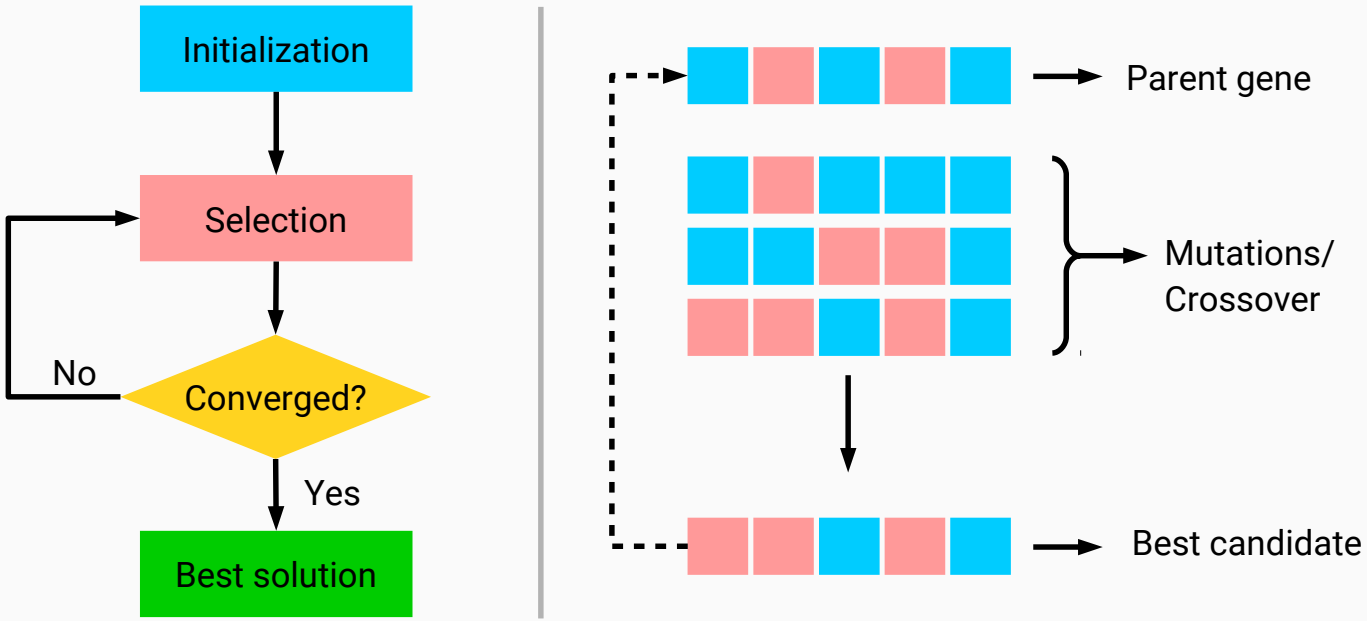


Genetic algorithm

Genetic algorithm is well suited when:

- gradients are not available,
- non parametric function,
- non homogeneous cost function along all training set points.

Example:



Artificial neural networks

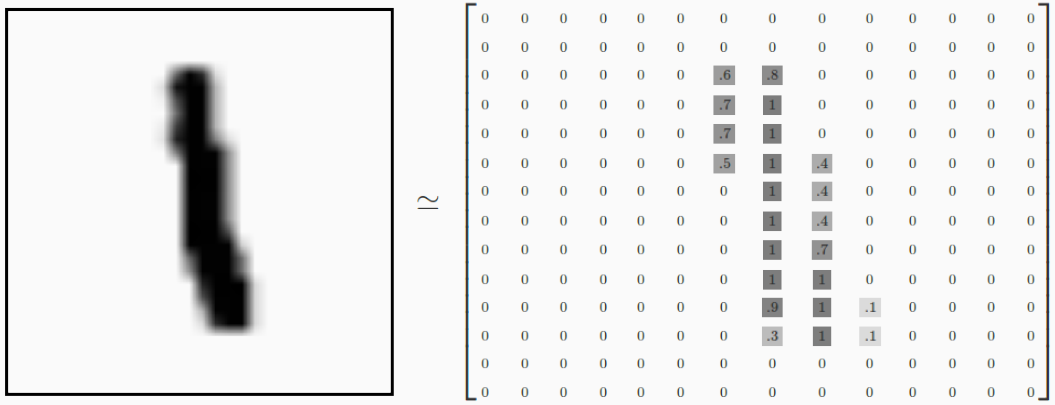
Limitations of linear models

Why not linear models everywhere?

Limitations of linear models

Why not linear models everywhere?

Example: consider 1 image from the MNIST database:



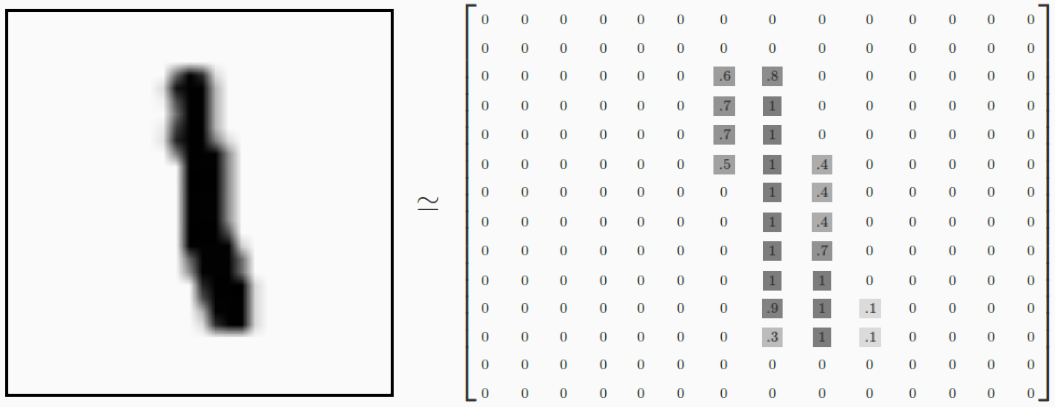
Each image has 28x28 pixels $= 785$ features (x3 if including RGB colors).

If consider quadratic function $\mathcal{O}(n^2)$ so linear models are impractical.

Limitations of linear models

Why not linear models everywhere?

Example: consider 1 image from the MNIST database:

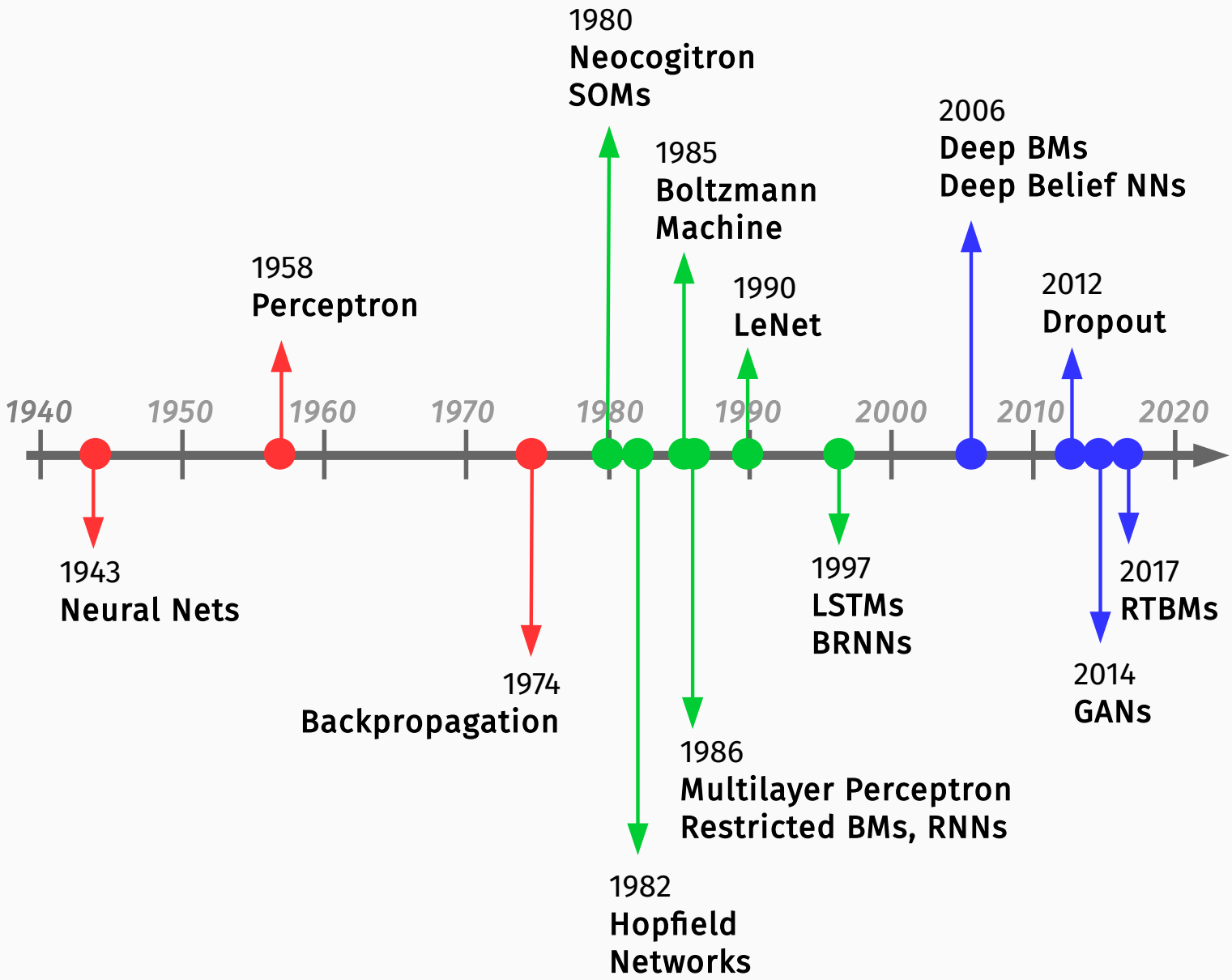


Each image has 28x28 pixels $= 785$ features (x3 if including RGB colors).

If consider quadratic function $\mathcal{O}(n^2)$ so linear models are impractical.

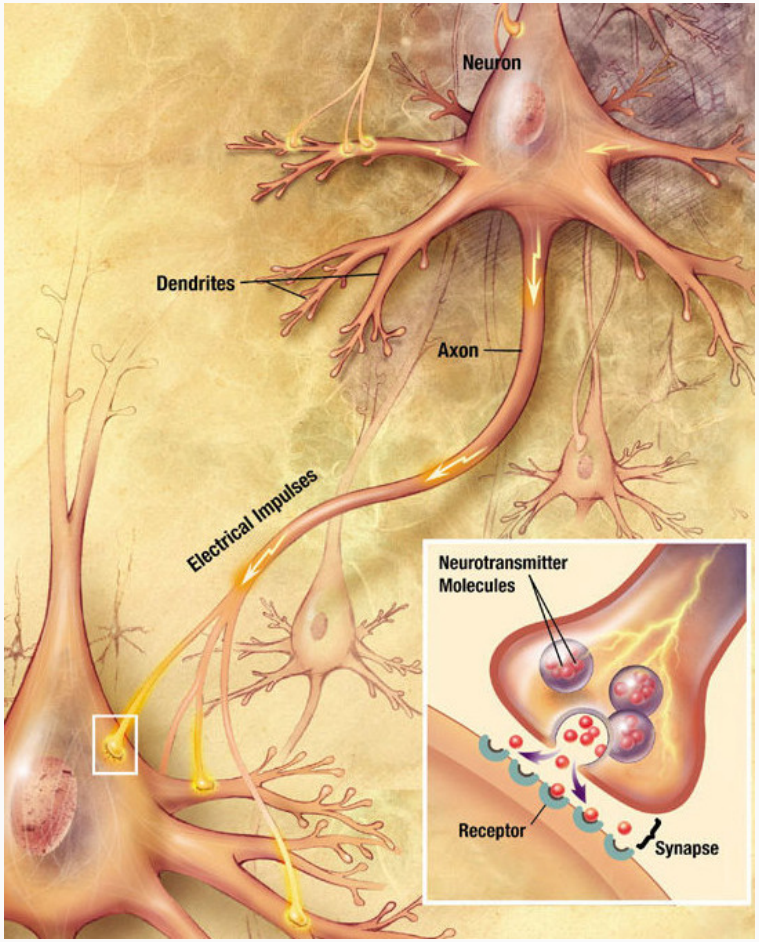
Solution: use non-linear models.

Non-linear models timeline



Neural networks

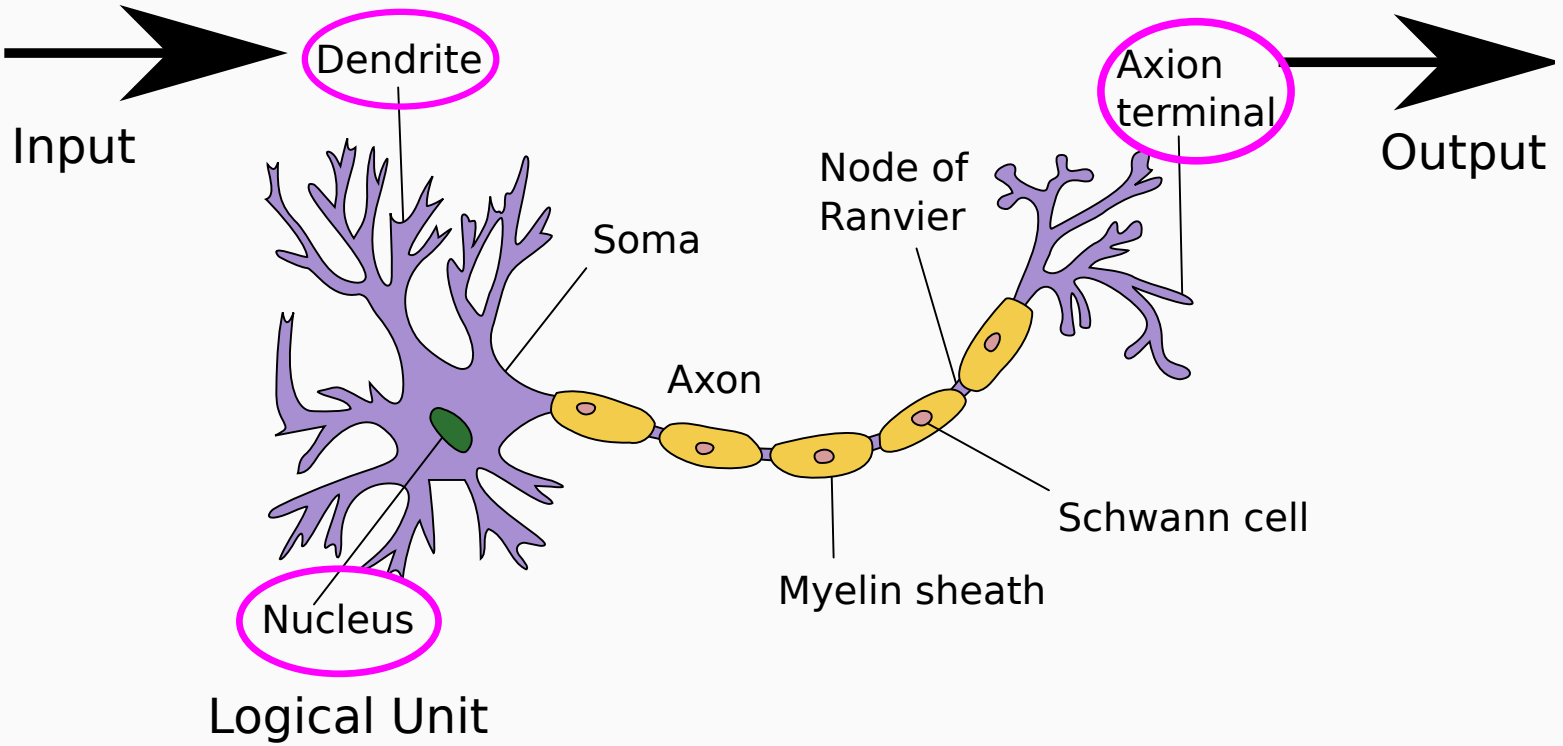
Artificial neural networks are computer systems inspired by the biological neural networks in the brain.



Currently the state-of-the-art technique for several ML applications.

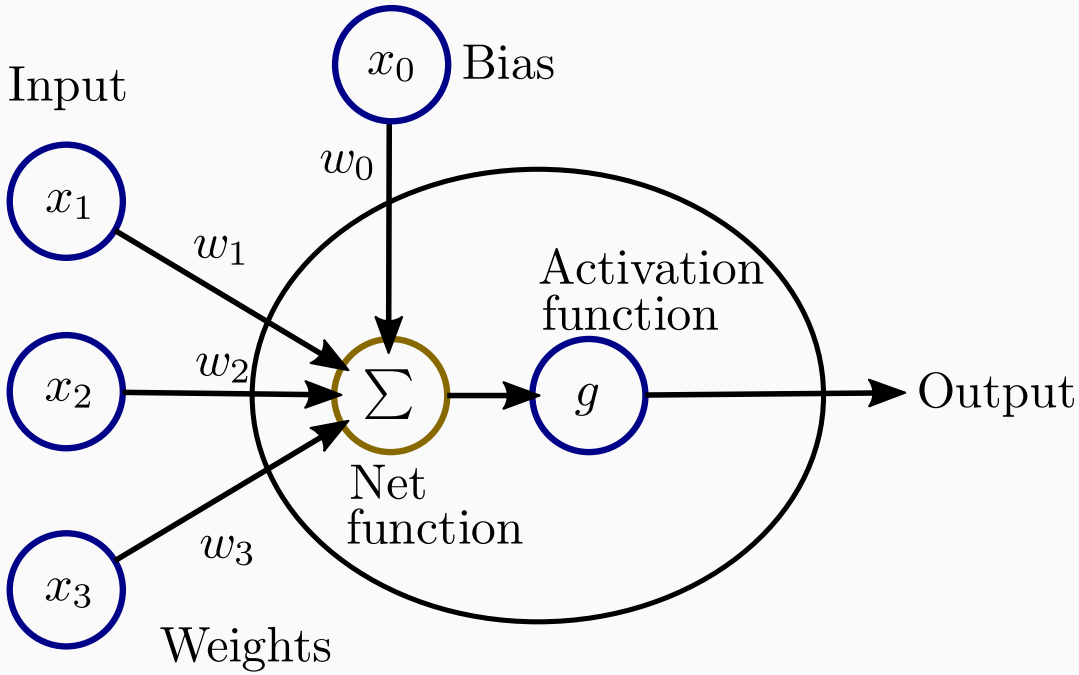
Neuron model

We can imagine the following data communication pattern:



Neuron model

Schematically:



where

- each **node** has an associate weights and bias w and inputs x,
- the output is modulated by an **activation function**, g.

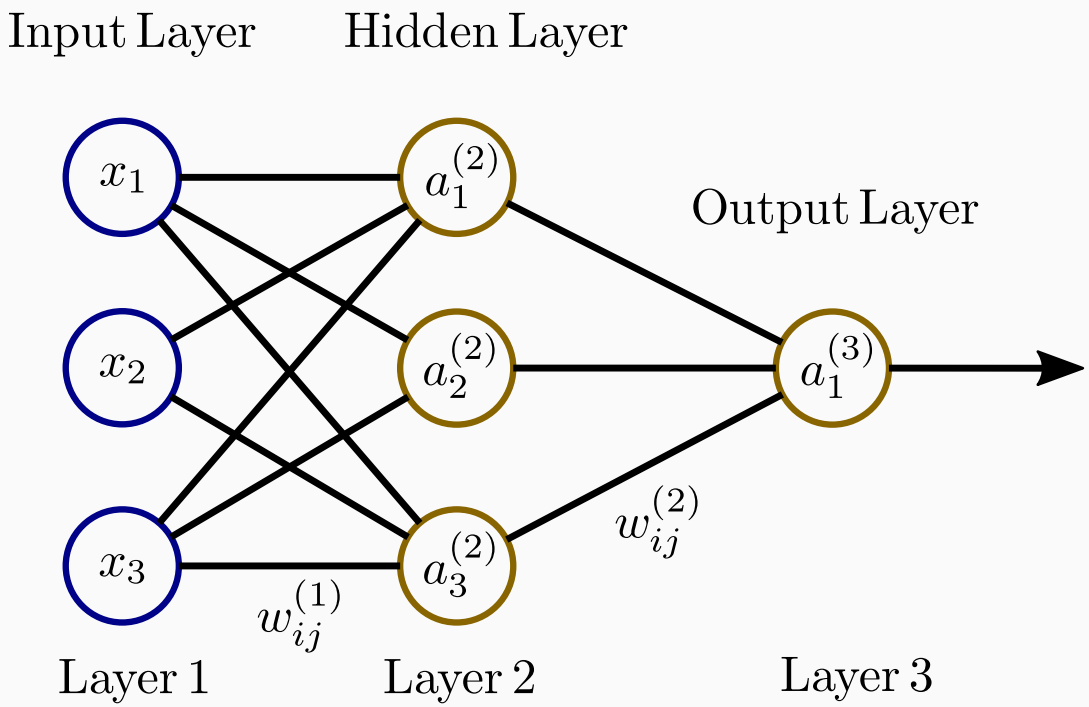
Some examples of activation functions: sigmoid, tanh, linear, ...

$$g_w(x) = \frac{1}{1+e^{-w^T x}}, \quad \tanh(w^T x), \quad x.$$

Neural networks

In practice, we simplify the bias term with $x_0 = 1$.

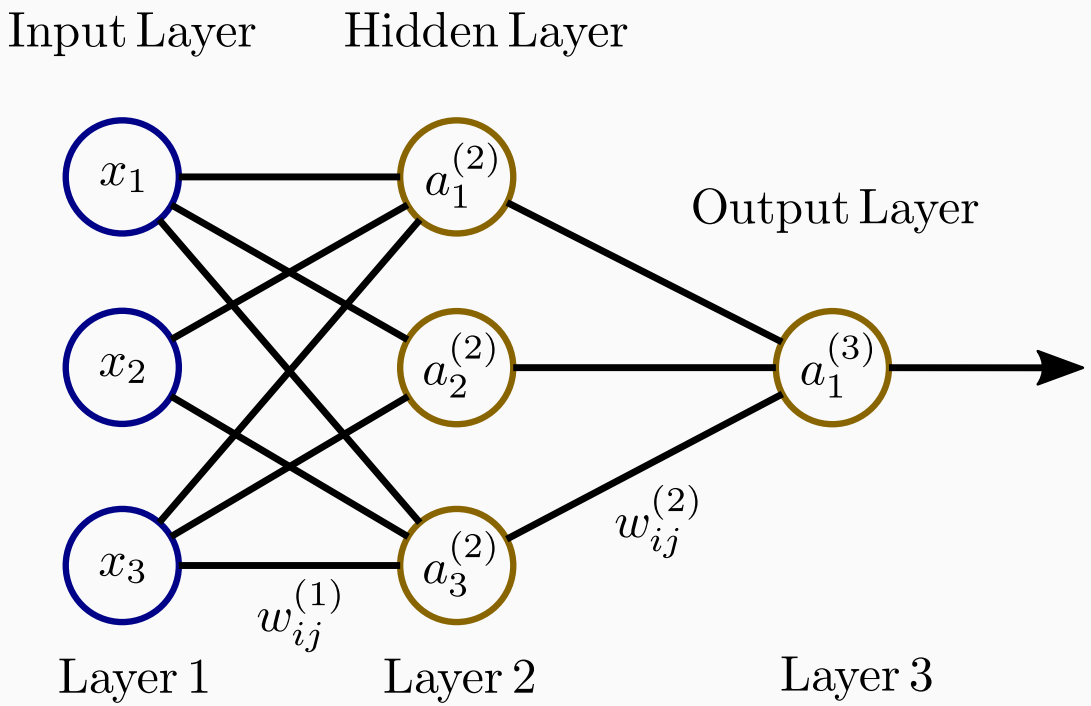
Neural network $\rightarrow$ connecting multiple units together.



where

- $a_i^{(l)}$ is the activation of unit i in layer l,
- $w_{ij}^{(l)}$ is the weight between nodes i, j from layers $l, l+1$ respectively.

Neural networks

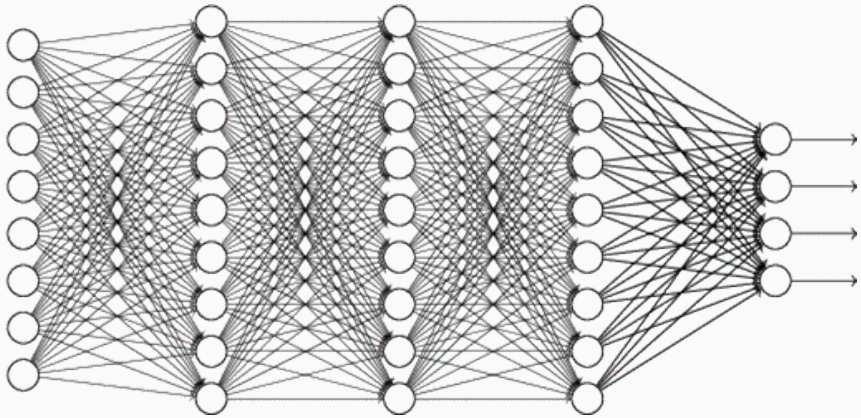


- $a_1^{(2)} = g(w_{10}^{(1)} + w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3)$
- $a_2^{(2)} = g(w_{20}^{(1)} + w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3)$
- $a_3^{(2)} = g(w_{30}^{(1)} + w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3)$
- **Output** $\rightarrow a_1^{(3)} = g(w_{10}^{(2)} + w_{11}^{(2)}a_1^{(2)} + w_{12}^{(2)}a_2^{(2)} + w_{13}^{(2)}a_3^{(2)})$

Neural networks

Some useful names:

- Feedforward neural network: no cyclic connections between nodes from the same layer (previous example).
- Multilayer perceptron (MLP): is a feedforward neural network with at least 3 layers.
- Deep neural networks: term referring to neural networks with more than one hidden layer.



Training neural networks

The training NNs is usually performed with gradient descent methods.

Following the previous section, we have to compute the cost function gradient with respect to parameters $w_{ij}^{(l)}$:

$$w_{ij}^{(l)} := w_{ij}^{(l)} - \eta \nabla_{ij}^{(l)} J \quad \rightarrow \quad \nabla_{ij}^{(l)} J = \frac{\partial}{\partial w_{ij}^{(l)}} J(\boldsymbol{w})$$

Training neural networks

The training NNs is usually performed with gradient descent methods.

Following the previous section, we have to compute the cost function gradient with respect to parameters $w_{ij}^{(l)}$:

$$w_{ij}^{(l)} := w_{ij}^{(l)} - \eta \nabla_{ij}^{(l)} J \quad \rightarrow \quad \nabla_{ij}^{(l)} J = \frac{\partial}{\partial w_{ij}^{(l)}} J(\boldsymbol{w})$$

Use the backpropagation algorithm to compute the gradient of a NN.

- can be used with any gradient-based optimizer, including quasi-Newton methods.
- reduces the large amount of computations thanks to chain rule
- requires the derivative of the cost function with respect to the output layer $w_{ij}^{(l)}$ with $l =$ output.

Backpropagation algorithm

The backpropagation steps:

1: perform a forward propagation (calculate $a_i^{(l)}$)

Backpropagation algorithm

The backpropagation steps:

1: perform a forward propagation (calculate $a_i^{(l)}$)

2: perform a backward propagation: evaluate for each node a "prediction error":

$$\delta_j^{(l)} = \text{“error” of node } j \text{ in layer } l.$$

Backpropagation algorithm

The backpropagation steps:

1: perform a forward propagation (calculate $a_i^{(l)}$)

2: perform a backward propagation: evaluate for each node a "prediction error":

$$\delta_j^{(l)} = \text{“error” of node } j \text{ in layer } l.$$

3: calculate $\nabla_{ij}^{(l)} J$ using erros $\delta_i^{(l)}$ and $a_i^{(l)}$.

Backpropagation algorithm

The backpropagation steps:

1: perform a forward propagation (calculate $a_i^{(l)}$)

2: perform a backward propagation: evaluate for each node a "prediction error":

$$\delta_j^{(l)} = \text{“error” of node } j \text{ in layer } l.$$

3: calculate $\nabla_{ij}^{(l)} J$ using erros $\delta_i^{(l)}$ and $a_i^{(l)}$.

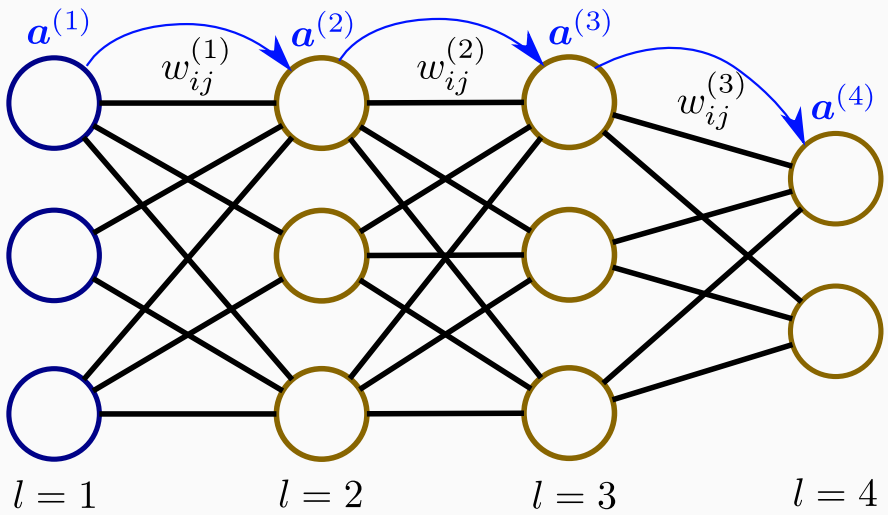
4: perform weight updates, $\Delta w_{ij}^{(l)}$, via gradient descent using $\nabla_{ij}^{(l)} J$.

Backpropagation algorithm

Suppose we have a MLP and one training example $(\boldsymbol{x}, \boldsymbol{y})$.

Step 1: We first perform a forward propagation pass:

- $\boldsymbol{a}^{(1)} = \boldsymbol{x}$

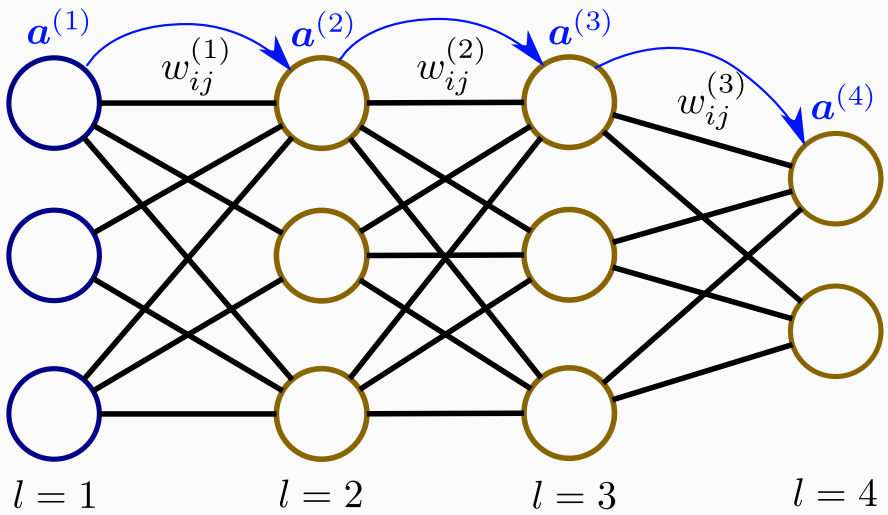


Backpropagation algorithm

Suppose we have a MLP and one training example $(\boldsymbol{x}, \boldsymbol{y})$.

Step 1: We first perform a forward propagation pass:

- $\boldsymbol{a}^{(1)} = \boldsymbol{x}$
- $\boldsymbol{z}^{(2)} = \boldsymbol{w}^{(1)}\boldsymbol{a}^{(1)}$

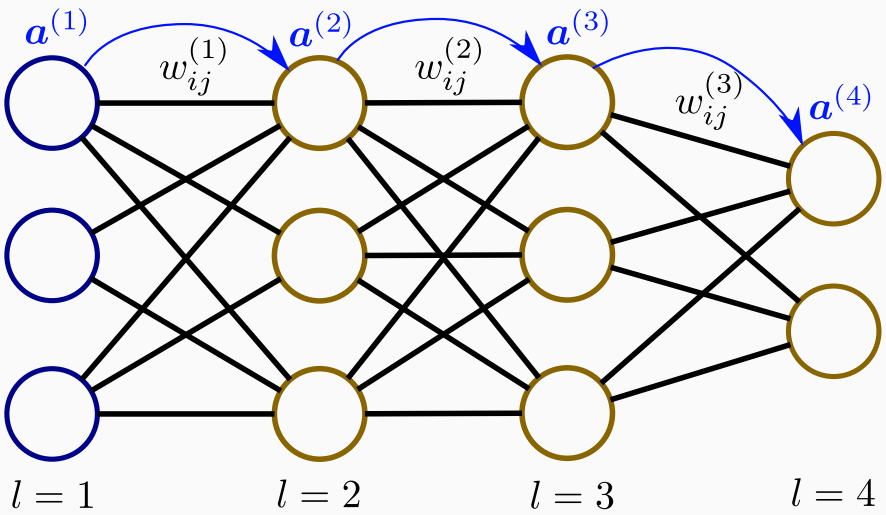


Backpropagation algorithm

Suppose we have a MLP and one training example $(\boldsymbol{x}, \boldsymbol{y})$.

Step 1: We first perform a forward propagation pass:

- $\boldsymbol{a}^{(1)} = \boldsymbol{x}$
- $\boldsymbol{z}^{(2)} = \boldsymbol{w}^{(1)}\boldsymbol{a}^{(1)}$
- $\boldsymbol{a}^{(2)} = g(\boldsymbol{z}^{(2)})$

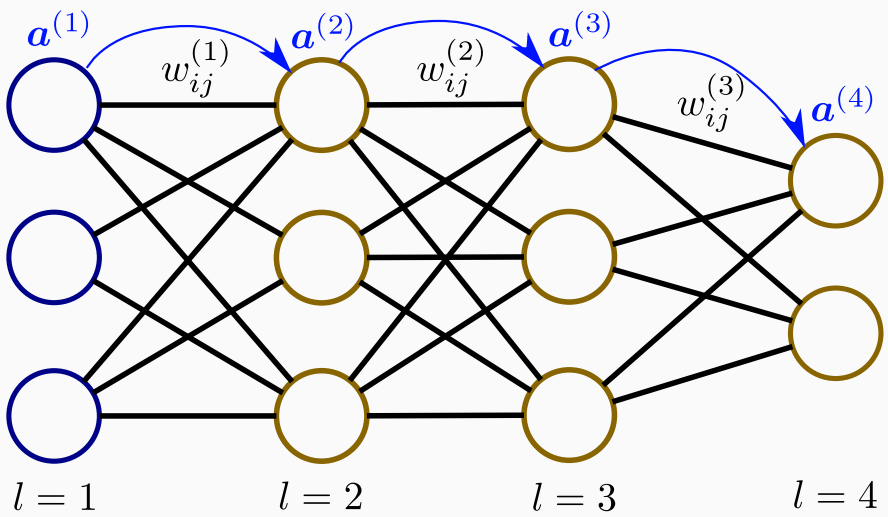


Backpropagation algorithm

Suppose we have a MLP and one training example $(\boldsymbol{x}, \boldsymbol{y})$.

Step 1: We first perform a forward propagation pass:

- $\boldsymbol{a}^{(1)} = \boldsymbol{x}$
- $\boldsymbol{z}^{(2)} = \boldsymbol{w}^{(1)}\boldsymbol{a}^{(1)}$
- $\boldsymbol{a}^{(2)} = g(\boldsymbol{z}^{(2)})$
- $\boldsymbol{z}^{(3)} = \boldsymbol{w}^{(2)}\boldsymbol{a}^{(2)}$

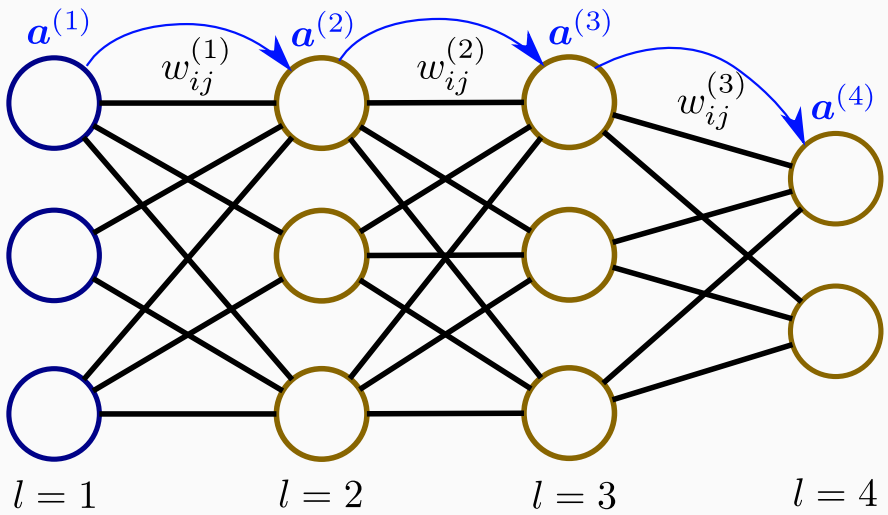


Backpropagation algorithm

Suppose we have a MLP and one training example $(\boldsymbol{x}, \boldsymbol{y})$.

Step 1: We first perform a forward propagation pass:

- $\boldsymbol{a}^{(1)} = \boldsymbol{x}$
- $\boldsymbol{z}^{(2)} = \boldsymbol{w}^{(1)}\boldsymbol{a}^{(1)}$
- $\boldsymbol{a}^{(2)} = g(\boldsymbol{z}^{(2)})$
- $\boldsymbol{z}^{(3)} = \boldsymbol{w}^{(2)}\boldsymbol{a}^{(2)}$
- $\boldsymbol{a}^{(3)} = g(\boldsymbol{z}^{(3)})$

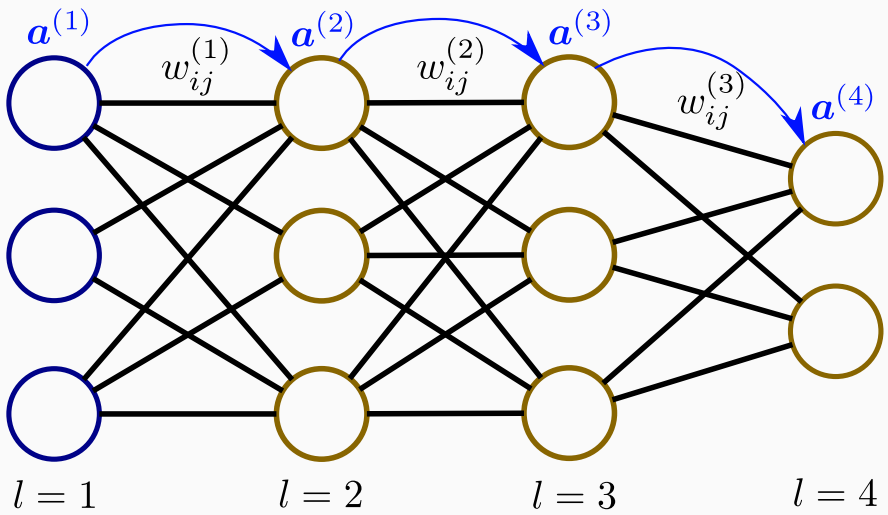


Backpropagation algorithm

Suppose we have a MLP and one training example $(\boldsymbol{x}, \boldsymbol{y})$.

Step 1: We first perform a forward propagation pass:

- $\boldsymbol{a}^{(1)} = \boldsymbol{x}$
- $\boldsymbol{z}^{(2)} = \boldsymbol{w}^{(1)}\boldsymbol{a}^{(1)}$
- $\boldsymbol{a}^{(2)} = g(\boldsymbol{z}^{(2)})$
- $\boldsymbol{z}^{(3)} = \boldsymbol{w}^{(2)}\boldsymbol{a}^{(2)}$
- $\boldsymbol{a}^{(3)} = g(\boldsymbol{z}^{(3)})$
- $\boldsymbol{z}^{(4)} = \boldsymbol{w}^{(3)}\boldsymbol{a}^{(3)}$

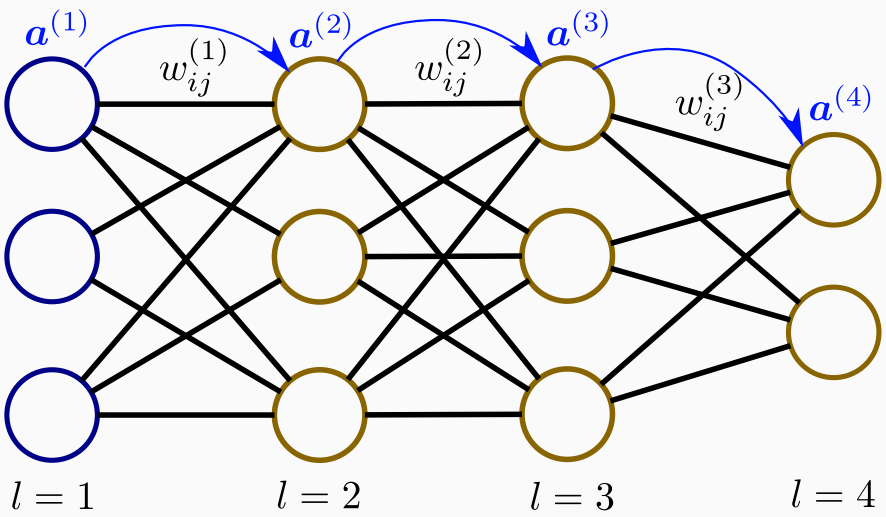


Backpropagation algorithm

Suppose we have a MLP and one training example $(\boldsymbol{x}, \boldsymbol{y})$.

Step 1: We first perform a forward propagation pass:

- $\boldsymbol{a}^{(1)} = \boldsymbol{x}$
- $\boldsymbol{z}^{(2)} = \boldsymbol{w}^{(1)}\boldsymbol{a}^{(1)}$
- $\boldsymbol{a}^{(2)} = g(\boldsymbol{z}^{(2)})$
- $\boldsymbol{z}^{(3)} = \boldsymbol{w}^{(2)}\boldsymbol{a}^{(2)}$
- $\boldsymbol{a}^{(3)} = g(\boldsymbol{z}^{(3)})$
- $\boldsymbol{z}^{(4)} = \boldsymbol{w}^{(3)}\boldsymbol{a}^{(3)}$
- Output $\boldsymbol{a}^{(4)} = g(\boldsymbol{z}^{(4)})$



At this step we know the output of the current MLP setup.

Backpropagation algorithm

2. evaluate for each node the error $\delta_j^{(k)}$ for $k = 2, 3, \ldots, L$.

Some remarks:

It is possible to proof using derivative chain rules that:

$$\nabla_{ij}^{(l)} J = \frac{\partial J}{\partial z_i^{(l+1)}} a_j^{(l)} \equiv \delta_i^{(l+1)} a_j^{(l)},$$

for $l = 1, \ldots, L-1$.

Backpropagation algorithm

2. evaluate for each node the error $\delta_j^{(k)}$ for $k = 2, 3, \ldots, L$.

Some remarks:

It is possible to proof using derivative chain rules that:

$$\nabla_{ij}^{(l)} J = \frac{\partial J}{\partial z_i^{(l+1)}} a_j^{(l)} \equiv \delta_i^{(l+1)} a_j^{(l)},$$

for $l = 1, \ldots, L-1$.

The recursive relation for the error is:

$$\delta_i^{(l)} = \sum_k w_{ki}^{(l)} \delta_k^{(l+1)} \cdot g'(z_i^{(l)})$$

and at $l = L$, *i.e.* the highest l index:

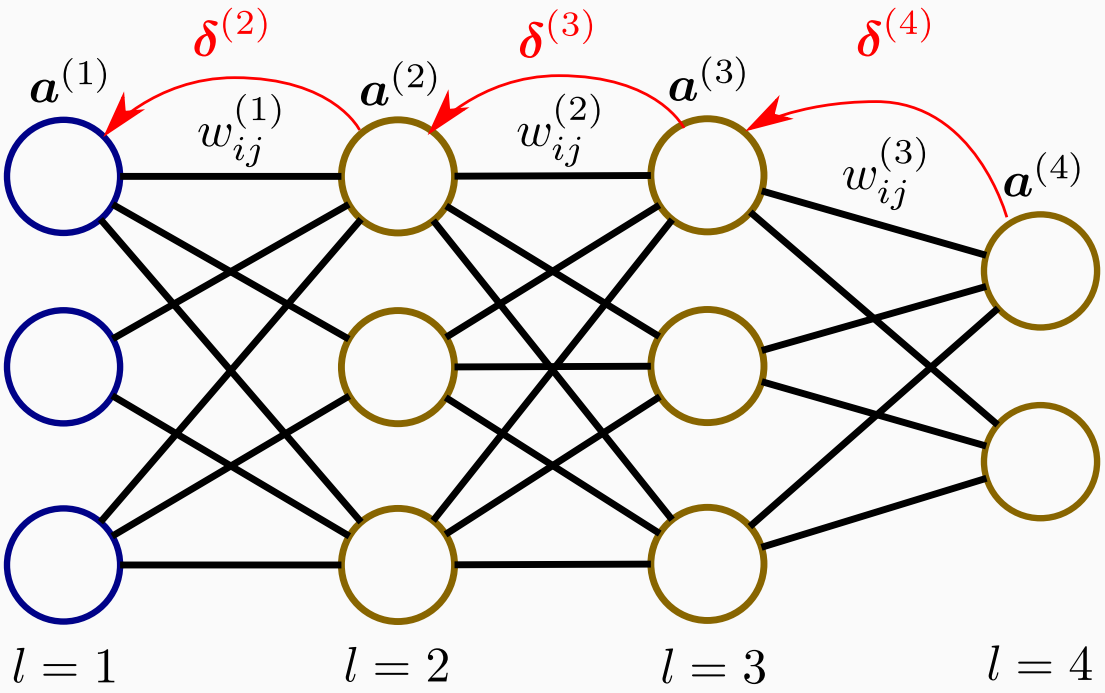
$$\delta_i^{(L)} = \frac{\partial J}{\partial a_i^{(L)}} \cdot g'(z_i^{(L)})$$

where $g'(z_i^{(l)}) = a_i^{(l)}(1 - a_i^{(l)})$ if g is the sigmoid function.

Backpropagation algorithm

Example: evaluating error $\delta_j^{(l)}$ for a MLP with sigmoids in the hidden layers and linear activation function in the output layer:

- $\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \boldsymbol{y}$
- $\boldsymbol{\delta}^{(3)} = (\boldsymbol{w}^{(3)})^T \boldsymbol{\delta}^{(4)} \cdot (\boldsymbol{a}^{(3)}(1 - \boldsymbol{a}^{(3)}))$
- $\boldsymbol{\delta}^{(2)} = (\boldsymbol{w}^{(2)})^T \boldsymbol{\delta}^{(3)} \cdot (\boldsymbol{a}^{(2)}(1 - \boldsymbol{a}^{(2)}))$



Backpropagation algorithm summary

Data: training set $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ with $i = 1, \ldots, m$ examples.
Result: the trained neural network
Initialize network weights;
while *stopping criterion is not satisfied* **do**
    Set all $\Delta w_{ij}^{(l)} = 0$.
    for $k = 1$ **to** m **do**
        Perform forward pass and compute $\boldsymbol{a}^{(l)}$ for $l = 1, 2, 3, \ldots, L$;
        Perform backward pass and compute $\boldsymbol{\delta}^{(l)}$ for $l = 2, \ldots, L$;
        $\Delta w_{ij}^{(l)} := \Delta w_{ij}^{(l)} + a_j^l \delta_i^{(l+1)}$
    end
    Update network weights using gradient descent;
end

Training neural networks

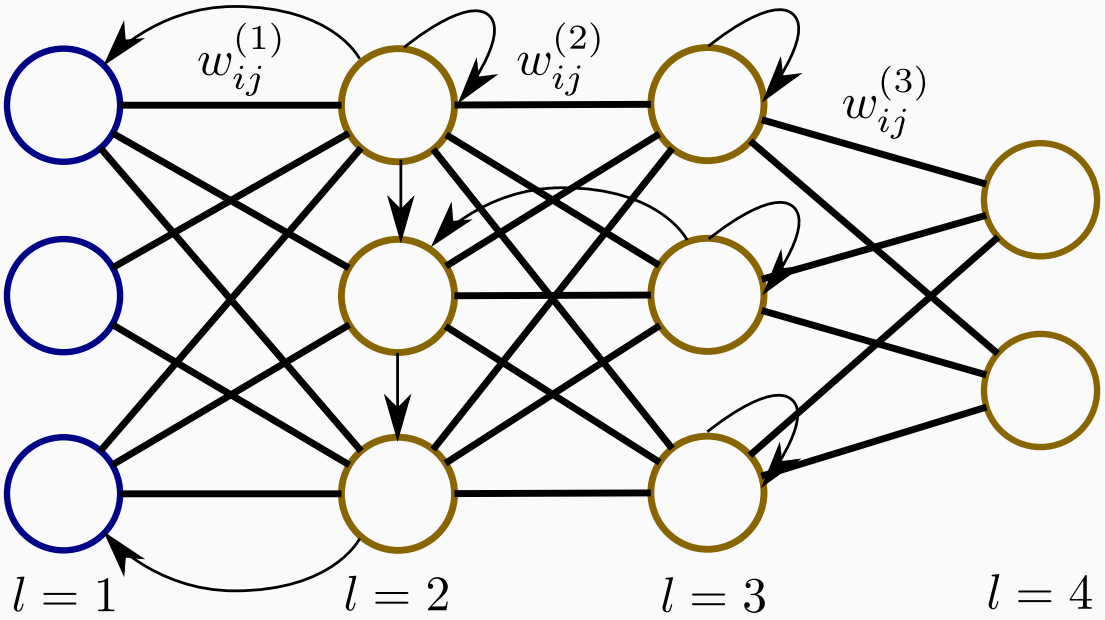
Some remarks and example of neural network initialization:

- zero: all weights are set to zero so all neurons perform the same calculation. The complexity of the neural network is equivalent to a single neuron.
- random: breaks parameter symmetry.
- glorot/xavier: initialize each weight with a small Gaussian value with mean zero and variance based on the in/out size of the weight.
- he: avoid activation function saturation. Weights are random initialized considering the size of the previous layer.

Artificial neural networks architectures

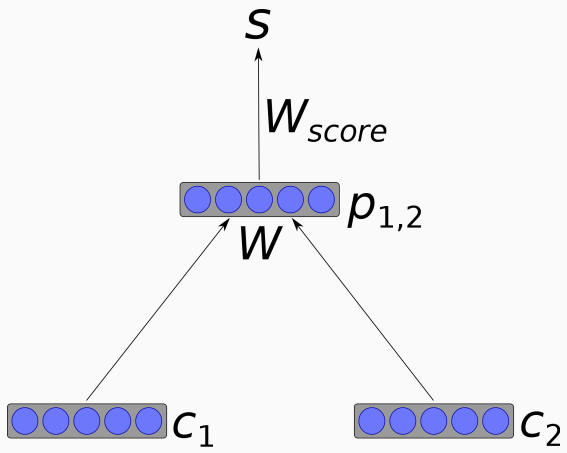
Some examples of neural network popular architectures:

- Recurrent neural networks: neural networks where connections between nodes form a directed cycle.
  - built-in internal state memory
  - built-in notion of time ordering for a time sequence



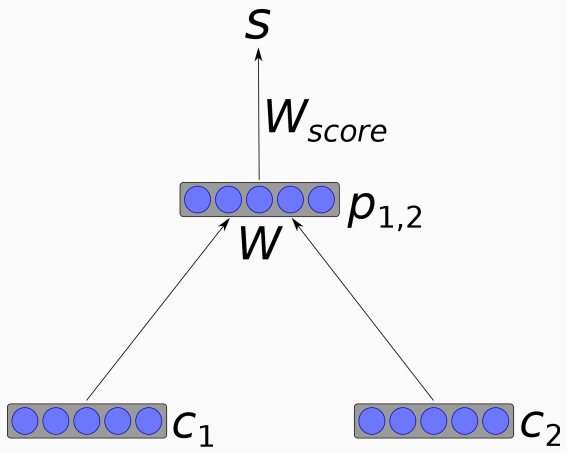
Artificial neural networks architectures

- Recursive neural networks: a variation of recurrent neural network where pairs of layers or nodes are merged recursively.
  - successful applications on natural language processing.
  - some recent applications for model inference.



Artificial neural networks architectures

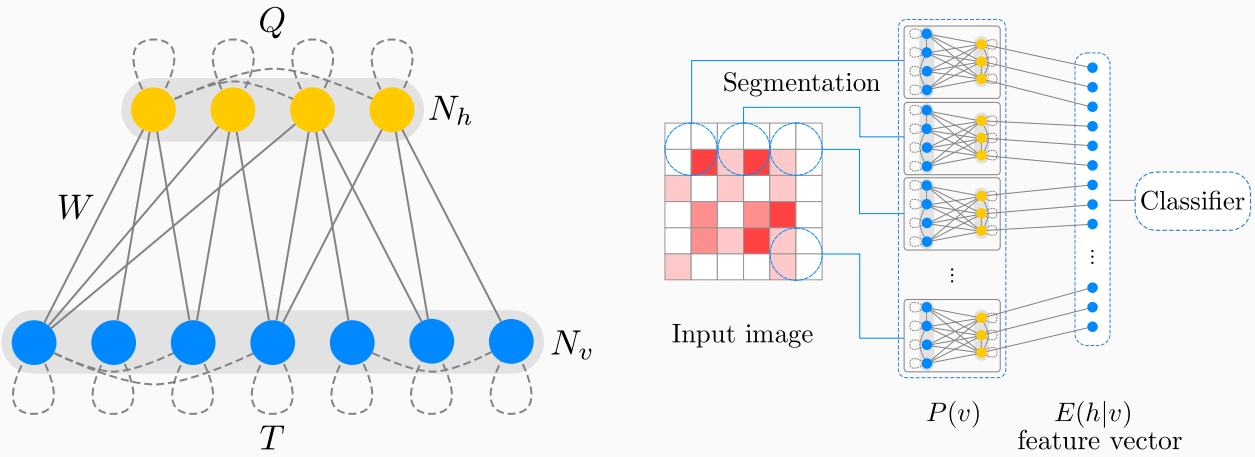
- Recursive neural networks: a variation of recurrent neural network where pairs of layers or nodes are merged recursively.
  - successful applications on natural language processing.
  - some recent applications for model inference.



- Long short-term memory: another variation of recurrent neural networks composed by custom units cells:
  - LSTM cells have an input gate, an output gate and a forget gate.
  - powerful when making predictions based on time series data.

Artificial neural networks architectures

- Boltzmann Machines: is a generative stochastic recursive artificial neural network.
  - comes with energy-based model features and advantages.
  - generalizations like RTBMs can be used for pdf estimate, filtering, regression, classification and sampling.

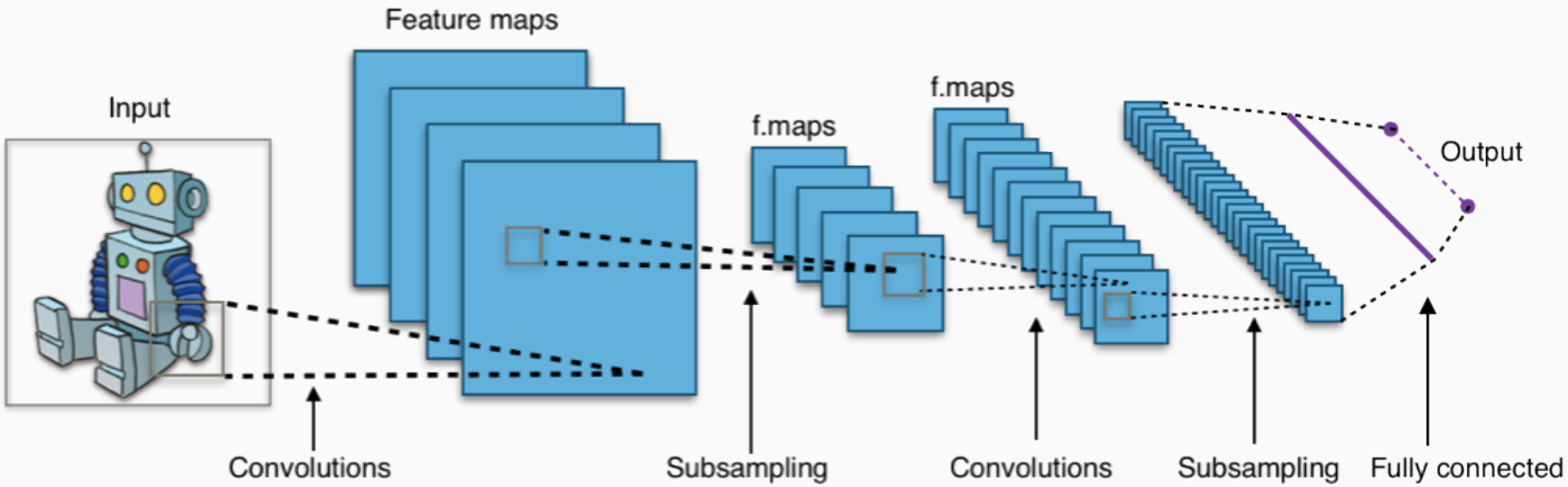


The system energy for given state vectors (v, h):

$$E(v,h) = \frac{1}{2}v^t T v + \frac{1}{2}h^t Q h + v^t W h + B_h h + B v v$$

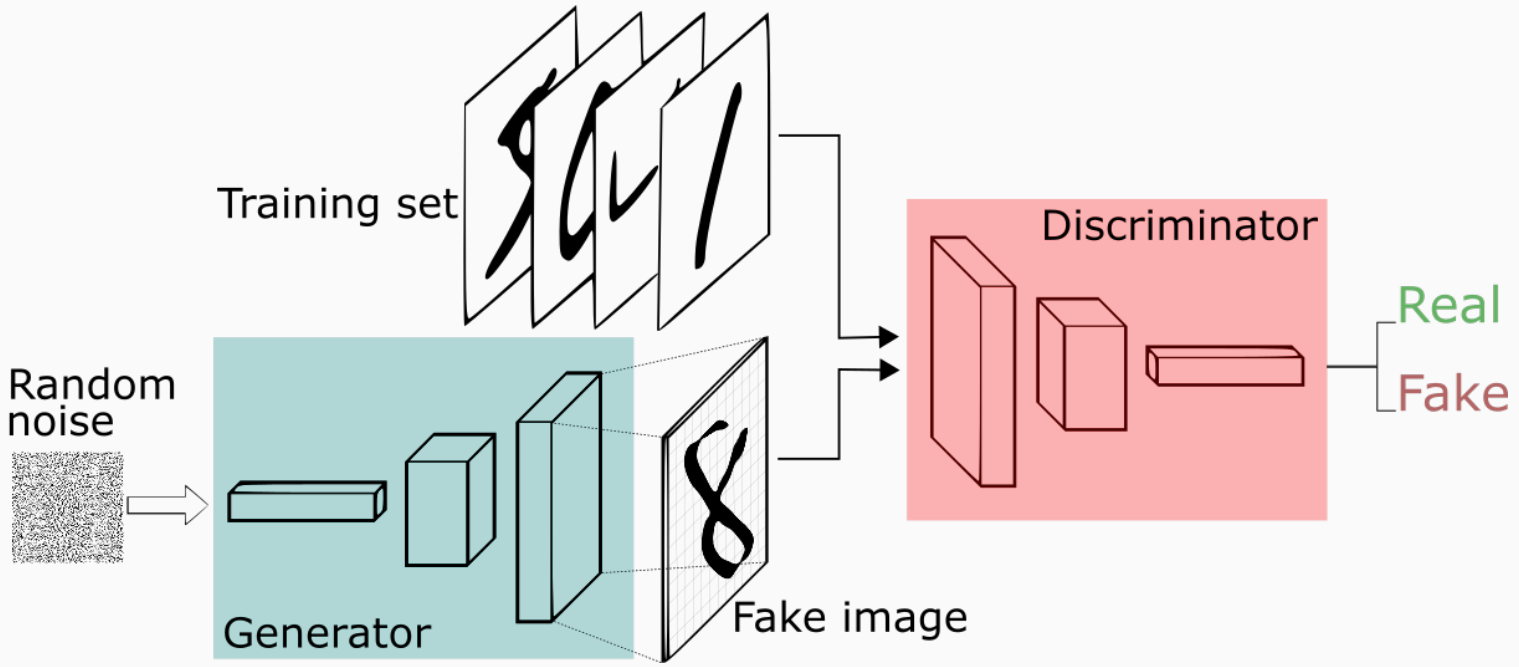
Artificial neural networks architectures

- Convolutional neural networks: multilayer perceptron designed to require minimal preprocessing, *i.e.* space invariant architecture.
  - the hidden layers consist of convolutional layers, pooling layer, fully connected layers and normalization layers
  - great successful applications in image and video recognition.



Artificial neural networks architectures

- Generative adversarial network: unsupervised machine learning system of two neural networks contesting with each other.
  - one network generate candidates while the other discriminates.



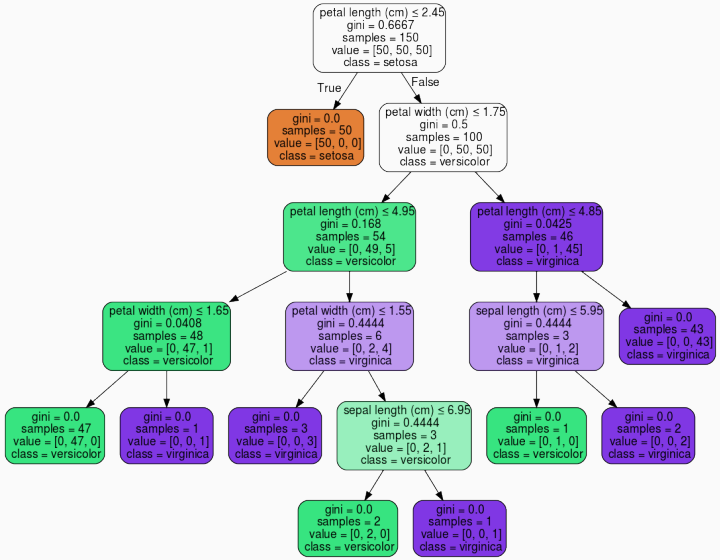
Beyond neural networks

Beyond neural networks

Even if neural networks are the most popular architecture nowadays employed in ML and Deep Learning, there are other models and techniques that are used frequently with great success in HEP-EXP

Supervised learning examples:

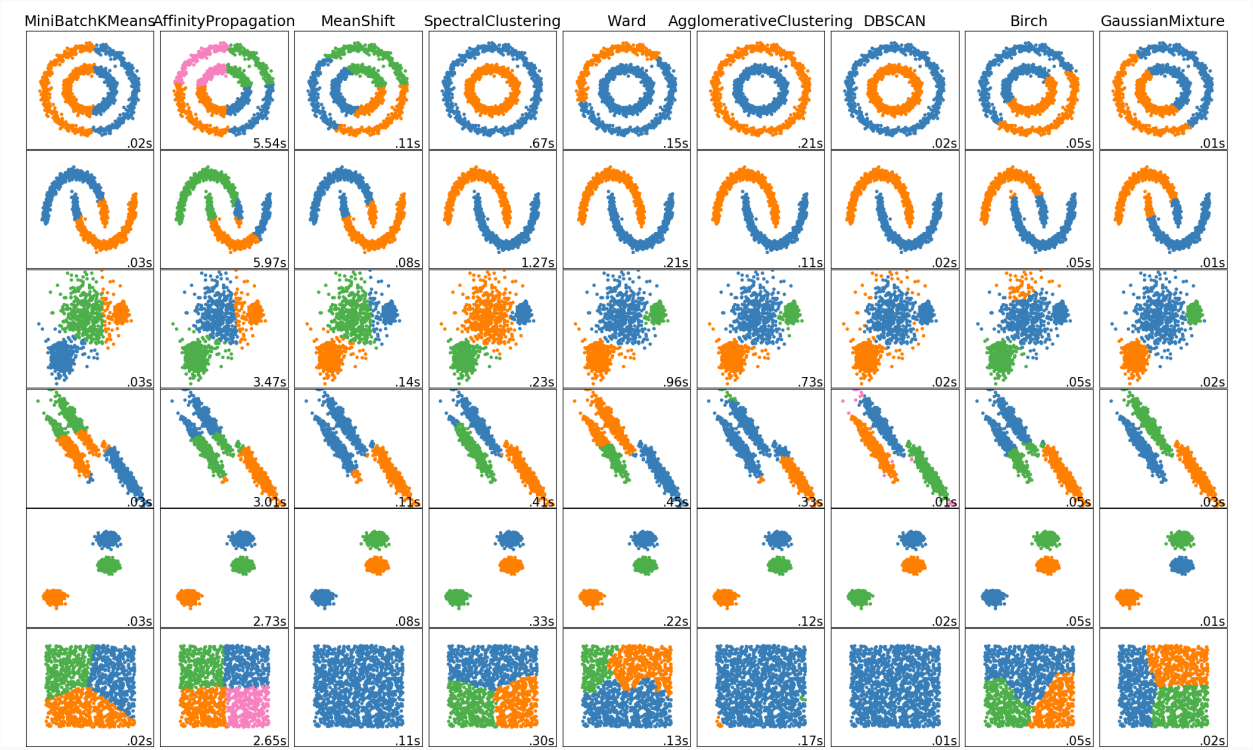
- Decision tree
- Ensemble models (random forest, bagging, boosting)
- Support Vector Machines (SVM)
- k-nearest neighbors algorithm (k-NN)



Clustering

Unsupervised learning examples:

- k-means
- Mean-shift
- Hierarchical
- Gaussian mixture models
- Density-based spatial
- Affinity propagation



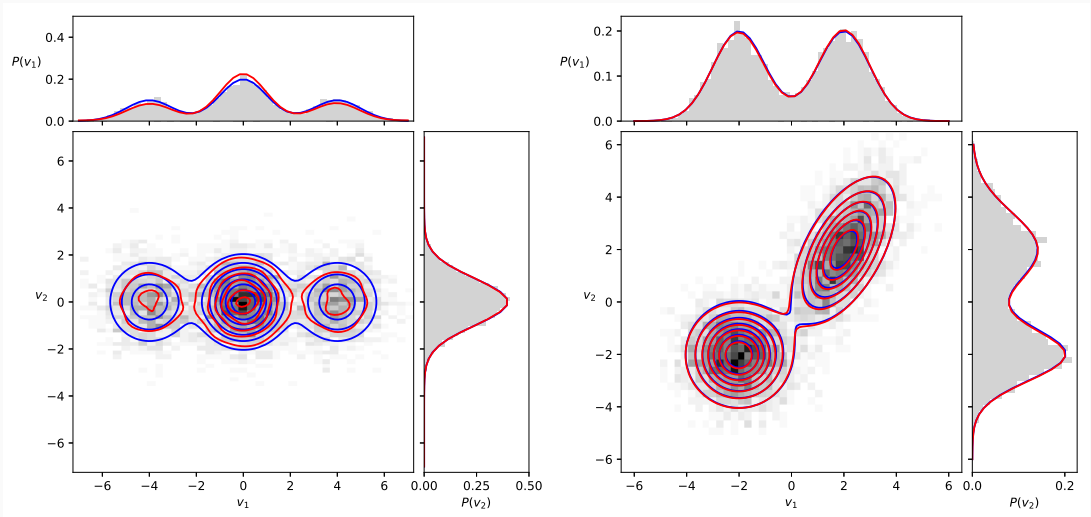
Others

Dimensionality reduction:

- Principal component analysis (PCA)
- Linear discriminant analysis (LDA)

Anomaly detection:

- GMM density estimate
- Kernel density estimate
- Restricted boltzmann machines
- k-NN



Summary

Summary

We have covered the following topics:

- Parameter learning: normal equations, first and second order optimization, genetic algorithm
- Neural network definition and most popular architectures
- Other models used in ML.