

Multilayer perceptron

Victor Kitov

v.v.kitov@yandex.ru

Table of Contents

- 1 Architecture
- 2 Sufficient number of layers
- 3 Activation functions
- 4 Optimization
- 5 Comments

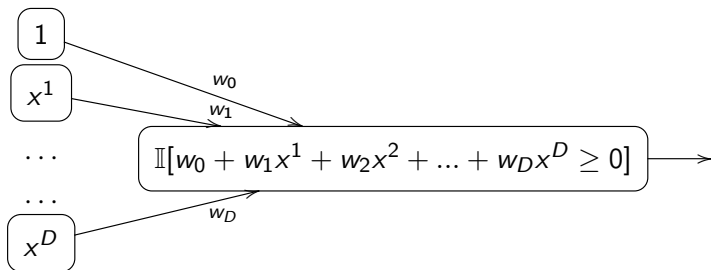
History

- Neural networks originally appeared as an attempt to model human brain



- Human brain consists of multiple interconnected neuron cells
 - cerebral cortex (the largest part) is estimated to contain 15–33 billion neurons
 - communication is performed by sending electrical and electro-chemical signals
 - signals are transmitted through axons - long thin parts of neurons.

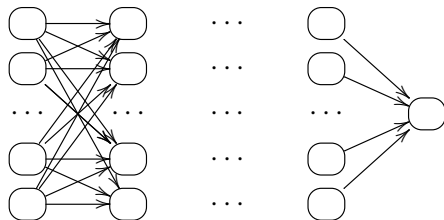
Simple model of a neuron



- Neuron get's activated in the half-space, defined by $w_0 + w_1x^1 + w_2x^2 + \dots + w_Dx^D \geq 0$.
- Each node is called a neuron
- Each edge is associated a weight
- w_0 stands for bias

Multilayer perceptron architecture¹

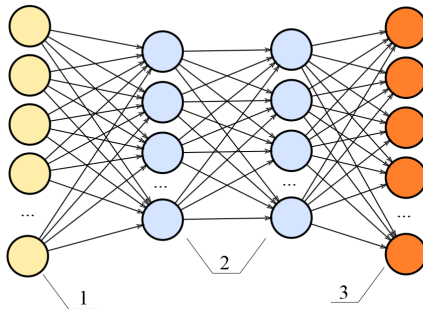
- Hierarchically nested set of neurons.
- Each node has its own weights.



This is structure of **multilayer perceptron** - acyclic directed graph.

¹Propose neural networks estimating OR,AND,XOR functions on boolean inputs.

Layers



- Structure of neural network:
 - 1-input layer
 - 2-hidden layers
 - 3-output layer

Definition details

- Label each neuron with integer j .
- Denote: I_j - input to neuron j , O_j - output of neuron j
- Output of neuron j : $O_j = \varphi(I_j)$.
- Input to neuron j : $I_j = \sum_{k \in inc(j)} w_{kj} O_k + w_{0j}$,
 - w_{0j} is the bias term
 - $inc(j)$ is a set of neurons with outgoing edges incoming to neuron j .
 - further we will assume that at each layer there is a vertex with constant output $O_{const} \equiv 1$, so we can simplify notation

$$I_j = \sum_{k \in inc(j)} w_{kj} O_k$$

Output generation

- **Forward propagation** is a process of successive calculations of neuron outputs for given features.

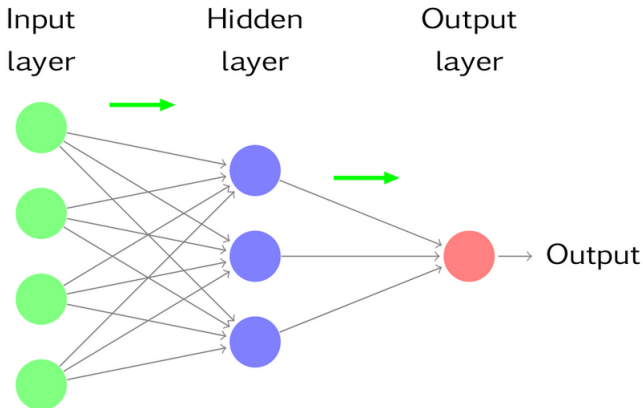


Table of Contents

- 1 Architecture
- 2 Sufficient number of layers
- 3 Activation functions
- 4 Optimization
- 5 Comments

Number of layers selection

- Consider indicator activations.
- **# layers selection for classification:**
 - single layer network selects arbitrary half-spaces
 - 2-layer network selects arbitrary convex polyhedron (by intersection of 1-layer outputs)
 - therefore it can approximate arbitrary convex sets
 - 3-layer network selects (by union of 2-layer outputs) arbitrary finite sets of polyhedra
- So 3 layered NN can approximate all regular² sets!

²With well-defined volume - Borel measurable.

Number of layers selection

- Consider indicator & identity (linear) activations.
- **# layers selection for regression:**
 - single layer can approximate arbitrary linear function
 - 2-layer network can model indicator function of arbitrary convex polyhedron
 - 3-layer network can uniformly approximate arbitrary continuous function (as sum weighted sum of indicators convex polyhedra)
- So 3 layered NN can approximate all regular³ dependencies!

³Function should be Borel measurable.

Number of layers selection

Sufficient amount of layers

Any continuous function on a compact space can be uniformly approximated by 2-layer neural network with linear output and any (without polynomial) activation functions.

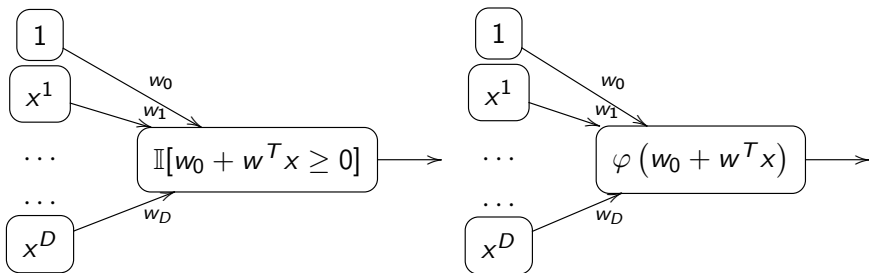
- In practice $\# \text{layers} = 2$ is enough, but may require too many neurons.
- So often it is more convenient to use more layers with less total amount of neurons
 - less parameters \Rightarrow less overfitting.

Table of Contents

- 1 Architecture
- 2 Sufficient number of layers
- 3 Activation functions**
- 4 Optimization
- 5 Comments

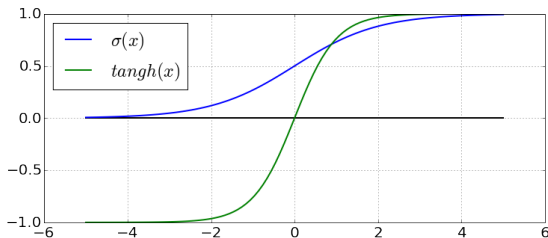
Continuous activations

- Pitfall of $\mathbb{I}[\cdot]$: it causes step-wise constant outputs, weight optimization methods become inapplicable.
- We can replace $\mathbb{I}[w^T x + w_0 \geq 0]$ with smooth activation $\varphi(w^T x + w_0)$



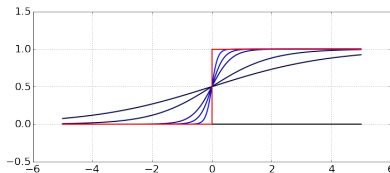
Typical activation functions

- sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
 - 1-layer neural network with sigmoidal activation is equivalent to logistic regression
- hyperbolic tangent: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
 - more computationally efficient: $\text{SoftSign}(x) = \frac{x}{1+|x|}$

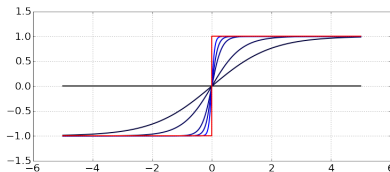


Activation functions

Activation functions are smooth approximations of step functions:



$\sigma(ax)$ limits to 0/1-step function as $a \rightarrow \infty$

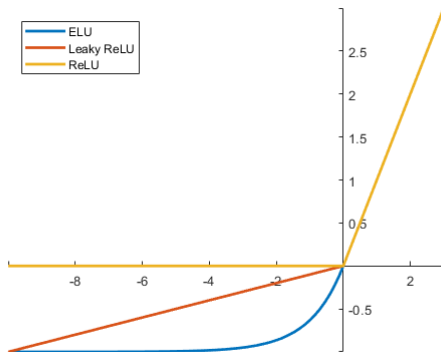


$\tanh(ax)$ limits to -1/1-step function as $a \rightarrow \infty$

Activation functions

- Rectified linear unit (ReLU): $\varphi(x) = x\mathbb{I}[x \geq 0]$
- SoftPlus: $\varphi(x) = \ln(1 + e^x)$
- Leaky ReLU: $\varphi(x) = \begin{cases} x, & x \geq 0 \\ 0.01x, & x < 0 \end{cases}$
- Parametric ReLU (PReLU): $\varphi(x|\alpha) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$
- Exponential LU (ELU): $\varphi(x) = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$

ReLU visualization



Activation functions

MaxOut: $\varphi(x) = \max\{w_1^T x + b_1, w_2^T x + b_2\}$

Gaussian: $\varphi(x|\mu, \sigma^2) = \exp\left(-\frac{\|x-\mu\|^2}{2\sigma^2}\right)$

Recommendations:

- do not use sigmoid (saturates, non-centered output)
- start from ReLU
- if many “dead” neurons - use leaky ReLU, PReLU, ELU, etc.

Activations at output layer

- Regression: $\varphi(I) = I$
- Classification:
 - binary: $y \in \{+1, -1\}$

$$p(y = +1|x) = \frac{1}{1 + e^{-I}}$$

- multiclass: $y \in 1, 2, \dots, C$

$$\varphi(O_1, \dots, O_C) = p(y = j|x) = \frac{e^{O_j}}{\sum_{k=1}^C e^{O_k}}, j = 1, 2, \dots, C$$

where O_1, \dots, O_C are outputs of output layer.

Table of Contents

- 1 Architecture
- 2 Sufficient number of layers
- 3 Activation functions
- 4 Optimization
 - Loss functions
 - Optimization
 - Backpropagation algorithm
- 5 Comments

4 Optimization

- Loss functions
- Optimization
- Backpropagation algorithm

Regression

- Scalar regression $y \in \mathbb{R}$:

$$MSE(x, y) = (\hat{y}(x) - y)^2$$

- Vector regression $\mathbf{y} \in \mathbb{R}^K$:

$$MSE(x, y) = \|\hat{\mathbf{y}}(x) - \mathbf{y}\|_2^2$$

- Total loss: summed loss over objects of the training set.

Classification (class probabilities output)

- Two classes $y \in \{0, 1\}$:

$$NLL(x, y) = -\ln p(y = 1|x)^y [1 - p(y = 1|x)]^{1-y}$$

- C classes $y \in \{1, 2, \dots, C\}$:

$$NLL(x, y) = -\ln \prod_{c=1}^C p(y = c|x)^{y_c}, \quad y_c = \mathbb{I}\{y = c\}$$

- Total loss: summed loss over objects of the training set.

Classification (class scores output)

- Two classes ($y \in \{-1, 1\}$):

$$\text{hinge}(x, y) = [O_{-y}(x) + 1 - O_y(x)]_+$$

- C classes ($y \in \{1, 2, \dots, C\}$):

$$\text{hinge}_1(x, y) = \left[\max_{c \neq y} O_c + 1 - O_y \right]_+$$

$$\text{hinge}_2(x, y) = \sum_{c \neq y} [O_c + 1 - O_y]_+$$

- Total loss: summed loss over objects of the training set.

4 Optimization

- Loss functions
- Optimization
- Backpropagation algorithm

Neural network optimization

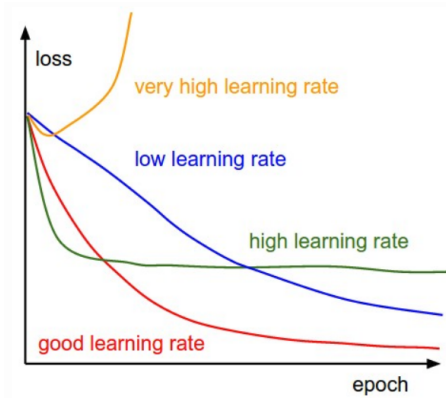
- Denote $\mathcal{L}(\hat{y}, y)$ - the loss function, $W = \#[\text{weights in the network}]$, η - step size.
- We may optimize neural network using SGD:

```
initialize randomly w # small values for sigmoid and tanh

while not (stop condition):
    sample random object  $(x_i, y_i)$ 
     $w := w - \eta \nabla_w \mathcal{L}(w, x_i, y_i)$ 
```

- Standardization of features makes GD & SGD converge faster
- Other optimization methods are more efficient (SGD+momentum, Adam)

Learning rate selection is important



Important to decrease learning rate on plateaus!

SGD with momentum

```
initialize randomly  $w$  # small values for sigmoid and tanh  
  
while not (stop condition):  
    sample random object  $(x_i, y_i)$   
     $\Delta w := \alpha \Delta w + (1 - \alpha) \nabla_w \mathcal{L}(w, x_i, y_i)$  # alpha is typically 0.9.  
     $w := w - \eta \Delta w$ 
```

- Converges faster, because
 - gradient is based on several observations instead of one
 - gradient directions aggregation averages uninformative trembling:



Gradient calculation

- Direct $\nabla \mathcal{L}(w)$ calculation, using

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\mathcal{L}(w + \varepsilon_i) - \mathcal{L}(w)}{\varepsilon} + O(\varepsilon) \quad (1)$$

or better

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\mathcal{L}(w + \varepsilon_i) - \mathcal{L}(w - \varepsilon_i)}{2\varepsilon} + O(\varepsilon^2) \quad (2)$$

has complexity $O(W^2)$

- need to calculate W derivatives
- complexity for each derivative: $2W$

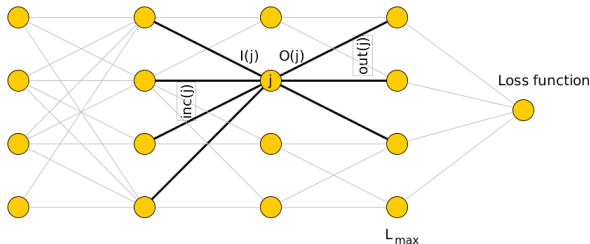
Backpropagation algorithm needs only $O(W)$ to evaluate all derivatives.

4 Optimization

- Loss functions
- Optimization
- Backpropagation algorithm

Definitions

- j - neuron number, having non-linearity $\varphi_j(\cdot)$
- I_j - input to neuron j , $O_j = \varphi_j(I_j)$ - output of neuron j
- $I_j = \sum_{k \in \text{inc}(j)} w_{kj} O_k$
- $\text{out}(j)$ - set of neurons having incoming connection from neuron j .
- bias omitted for simplicity



Definitions

- Denote w_{ij} be the weight of edge, connecting i -th and j -th neuron.
- Define $\delta_j = \frac{\partial \mathcal{L}}{\partial I_j} = \frac{\partial \mathcal{L}}{\partial O_j} \frac{\partial O_j}{\partial I_j}$
- Since \mathcal{L} depends on w_{ij} through the following functional relationship $\mathcal{L}(w_{ij}) \equiv \mathcal{L}(O_j(I_j(w_{ij})))$, using the chain rule we obtain:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial I_j} \frac{\partial I_j}{\partial w_{ij}} = \delta_j O_i$$

because $\frac{\partial I_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{k \in \text{inc}(j)} w_{kj} O_k \right) = O_i$, where $\text{inc}(j)$ is a set of all neurons with outgoing edges to neuron j .

- $\frac{\partial \mathcal{L}}{\partial I_j} = \frac{\partial \mathcal{L}}{\partial O_j} \frac{\partial O_j}{\partial I_j} = \frac{\partial \mathcal{L}}{\partial O_j} \varphi'(I_j)$, where φ is the activation function.

Backpropagation steps 1,2

- 1) If $layer(j) = L_{max}$ (lies in the output layer) $\frac{\partial \mathcal{L}}{\partial O_j}$ is calculated directly.

- e.g. for $\mathcal{L} = \frac{1}{2} \sum_{i \in OL} (O_i - y_i)^2$: $\frac{\partial \mathcal{L}}{\partial O_j} = O_j - y_j$

- 2) $\frac{\partial \mathcal{L}}{\partial I_j}$ is calculated from $\frac{\partial \mathcal{L}}{\partial O_j}$:

$$\frac{\partial \mathcal{L}}{\partial I_j} = \frac{\partial \mathcal{L}(O_j)}{\partial I_j} = \frac{\partial \mathcal{L}}{\partial O_j} \frac{\partial O_j}{\partial I_j} = \frac{\partial \mathcal{L}}{\partial O_j} \varphi'_j(I_j)$$

Backpropagation steps 3,4

3) $\frac{\partial \mathcal{L}}{\partial O_j}$ is calculated from $\frac{\partial \mathcal{L}}{\partial I_j}$ of next layer:

$$\frac{\partial \mathcal{L}}{\partial O_j} = \frac{\partial \mathcal{L}(\{I_s\}_{s \in \text{out}(j)})}{\partial O_j} = \sum_{s \in \text{out}(j)} \frac{\partial \mathcal{L}}{\partial I_s} \frac{\partial I_s}{\partial O_j} = \sum_{s \in \text{out}(j)} \frac{\partial \mathcal{L}}{\partial I_s} w_{js}$$

4) Using $\frac{\partial \mathcal{L}}{\partial I_j}$ we can calculate $\frac{\partial \mathcal{L}}{\partial w_{ij}}$ (given that $i \in \text{inc}(j)$):

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}(I_j)}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial w_{ij}} \frac{\partial I_j}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial I_j} \frac{\partial}{\partial w_{ij}} \left(\sum_{k \in \text{inc}(j)} w_{kj} O_k \right) = \frac{\partial \mathcal{L}}{\partial I_j} O_i$$

Backpropagation algorithm

FORWARD PASS: memorize I_j, O_j for all neurons.

BACKWARD PASS:

evaluate $\frac{\partial \mathcal{L}}{\partial O_j}$ for all neurons of output layer, using (1).

evaluate $\frac{\partial \mathcal{L}}{\partial I_j}$ for all neurons of output layer, using (2).

for layer L in $L_{\max} - 1, L_{\max} - 2, \dots, 1$:

 evaluate $\frac{\partial \mathcal{L}}{\partial w_{ij}}$ for all j of layer $L + 1$ and all $i \in \text{inc}(j)$,
 using (4).

 evaluate $\frac{\partial \mathcal{L}}{\partial O_j}$ for all neurons of layer L , using (3).

 evaluate $\frac{\partial \mathcal{L}}{\partial I_j}$ for all neurons of layer L , using (2).

Comments

- Must know $\varphi_j(\cdot)$ and $\varphi'_j(\cdot)$.
- As we may calculate $\frac{\partial \mathcal{L}}{\partial O_j} \forall j$, we may estimate $\frac{\partial \mathcal{L}}{\partial x^d}$ because $x^d = d\text{-th output of input layer}$.
 - May be used for fine-tuning input to have desired properties
 - e.g. style transfer
 - control optimization, having minimal loss (oil production optimization)
- Backpropagation correctness is checked by comparing results with (1), (2).

Complexity

- Denote $W = \#[\text{connections in the network}]$.
- Computational complexity: $O(W)$
 - due to steps (3) and (4).
- Memory complexity: $O(W)$, need to store:
 - I_j, O_j for all neurons.
 - w_{ij} for all connected neuron pairs.

How many objects to consider?

- **Batch mode:** estimate loss on all objects
 - works only for small N
- **Minibatch mode:** estimate loss on K random objects
 - $K \ll N$, works on big data
 - sampling: rolling window of width K over shuffled training set
 - minibatch size \propto parallelization ability of CPU or GPU (better).

Table of Contents

- 1 Architecture
- 2 Sufficient number of layers
- 3 Activation functions
- 4 Optimization
- 5 Comments**

Multilayer perceptron as ensemble method

All network is optimized, in contrast:

- boosting keeps previous trees fixed
- stacking keeps base learners fixed.

Overfitting is more severe - need regularization.

Multiple local optima problem

- Optimization problem for neural nets is **non-convex**.
- Different optima will correspond to:
 - different starting parameter values
 - different training samples
- So we may solve task many times for different conditions and then
 - select best model
 - alternatively: average different obtained models to get ensemble.

Regularization in NNs

- Constrain model complexity directly
 - constrain number of neurons
 - constrain number of layers
 - impose constraints on weights
- Take a flexible model
 - use early stopping during iterative evaluation (by controlling validation error)
 - add L2/L1 regularization:

$$\tilde{\mathcal{L}}(w) = \mathcal{L}(w) + \lambda \sum_i w_i^2$$

- λ may be different for each layer.

Density modelling

Neural networks may model not only real value outputs, but densities

- each output - frequency of histogram bin
- each output - either prior or mean or variance of mixture of parametrized density (normal, beta, etc.)

Conclusion

- Advantages of neural networks:
 - can model accurately complex non-linear relationships
 - easily parallelizable
- Disadvantages of neural networks:
 - hardly interpretable ("black-box" algorithm)
 - optimization requires skill
 - too many parameters
 - may converge slowly
 - may converge to inefficient local minimum far from global one
- Everything in optimization affects final result:
 - starting conditions, optimization algorithm, learning rate size and its decrease schedule.