Neural networks

Victor Kitov

v.v.kitov@yandex.ru

Table of Contents

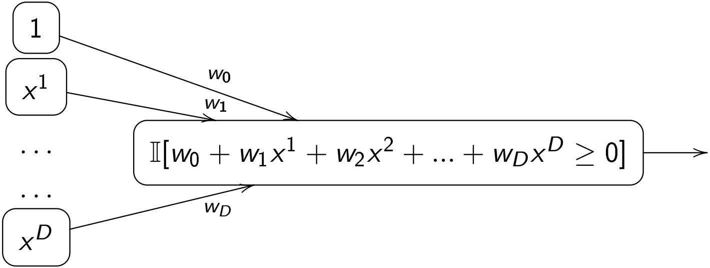
History

- 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.

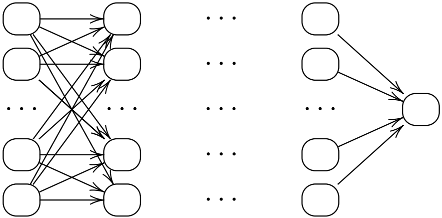
Simple model of a neuron



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

Multilayer perceptron architecture[1]

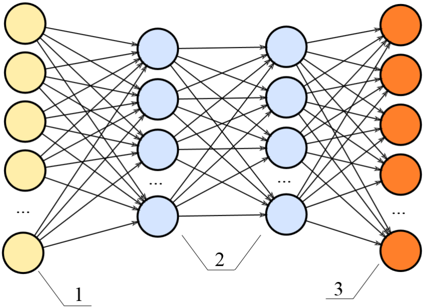
- Hierarchically nested set of neurons.
- Each node has its own weights.



This is structure of multilayer perceptron - acyclic directed graph.

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

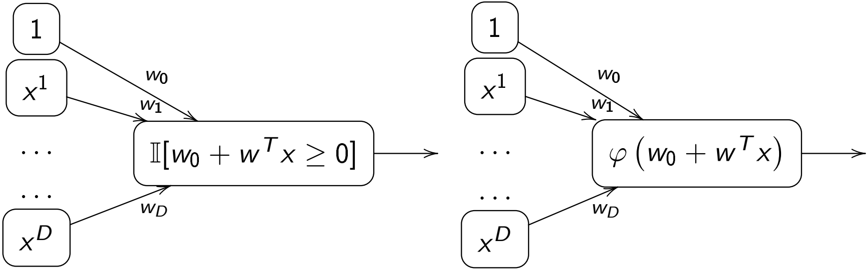
Layers



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

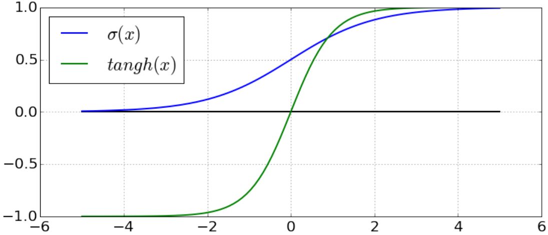
Continious activations

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



Typical activation functions

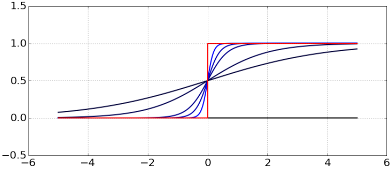
- sigmoidal: $\sigma(x) = \frac{1}{1+e^{-x}}$
  - 1-layer neural network with sigmoidal activation is equivalent to logistic regression
- hyperbolic tangent: $\text{tangh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



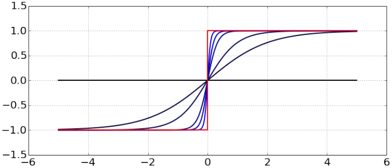
- ReLu: $\varphi(x) = [x]_+$.

Activation functions

Activation functions are smooth approximations of step functions:



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



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

Definition details

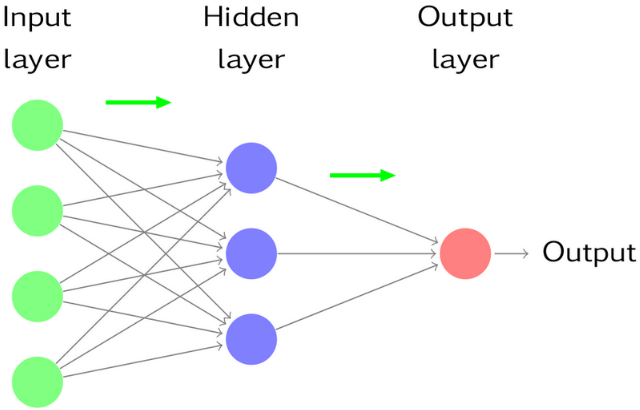
- Label each neuron with integer j.
- Denote: I_j - input to neuron j, O_j - output of neuron j
- Output of neuron j: $O_j = \varphi(I_j)$.
- Input to neuron j: $I_j = \sum_{k \in inc(j)} w_{kj} O_k + w_{0j}$,
    - w_{0j} is the bias term
    - $inc(j)$ is a set of neurons with outging edges incoming 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

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

Table of Contents

Output generation

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



Activations at output layer

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

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

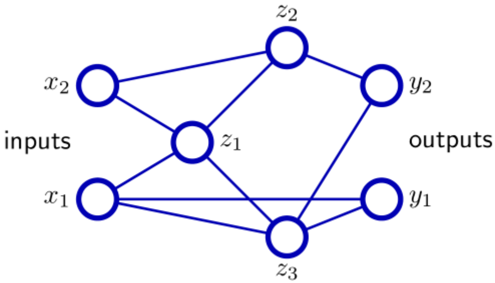
  - multiclass: $y \in 1, 2, ...C$

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

where $O_1, ...O_C$ are outputs of output layer.

Generalizations

- each neuron j may have custom non-linear transformation φ_j
- weights may be constrained:
    - non-negative
    - equal weights
    - etc.
- layer skips are possible



- Not considered here: RBF-networks, recurrent networks.

Number of layers selection

- Number of layers usually denotes all layers except input layer (hidden layers+output layer)
- 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
        - therefore it can approximate almost all sets with well defined volume (Borel measurable)

Number of layers selection

- 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)

Sufficient amount of layers

Any continuous function on a compact space can be uniformly approximated by 2-layer neural network with linear output and wide range of activation functions (excluding polynomial).

- In practice often it is more convenient to use more layers with less total amount of neurons
    - model becomes more interpretable and easy to fit.

Table of Contents

Network optimization: regression

- Single output:

$$\frac{1}{N}\sum_{n=1}^{N}(\widehat{y}_n(x_n) - y_n)^2 \to \min_{w}$$

Network optimization: regression

- Single output:

$$\frac{1}{N}\sum_{n=1}^{N}(\widehat{y}_n(x_n)-y_n)^2 \to \min_w$$

- K outputs

$$\frac{1}{NK}\sum_{n=1}^{N}\sum_{k=1}^{K}(\widehat{y}_{nk}(x_n)-y_{nk})^2 \to \min_w$$

Network optimization: classification

- Two classes ($y \in \{0, 1\}$, $p = P(y = 1)$):

$$\prod_{n=1}^{N} p(y_n = 1|x_n)^{y_n}[1 - p(y_n = 1|x_n)]^{1-y_n} \to \max_{w}$$

Network optimization: classification

- Two classes ($y \in \{0, 1\}$, $p = P(y = 1)$):

$$\prod_{n=1}^{N} p(y_n = 1|x_n)^{y_n}[1 - p(y_n = 1|x_n)]^{1-y_n} \to \max_w$$

- C classes ($y_{nc} = \mathbb{I}\{y_n = c\}$):

$$\prod_{n=1}^{N} \prod_{c=1}^{C} p(y_n = c|x_n)^{y_{nc}} \to \max_w$$

Network optimization: classification

- Two classes ($y \in \{0, 1\}$, $p = P(y = 1)$):

$$\prod_{n=1}^{N} p(y_n = 1|x_n)^{y_n}[1 - p(y_n = 1|x_n)]^{1-y_n} \to \max_w$$

- C classes ($y_{nc} = \mathbb{I}\{y_n = c\}$):

$$\prod_{n=1}^{N} \prod_{c=1}^{C} p(y_n = c|x_n)^{y_{nc}} \to \max_w$$

- In practice log-likelihood is maximized.

Neural network optimization

- Let W denote the total dimensionality of weights space
- Let $E(\widehat{y}, y)$ denote the loss function of output
- We may optimize neural network using gradient descent:

$k = 0$
initialize randomly w^0 # small values for sigmoid and tangh

while stop criteria not met:
$\quad w^{k+1} := w^k - \eta \nabla E(w^k)$
$\quad k := k + 1$

- Standardization of features makes gradient descend converge faster
- Other optimization methods are more efficient (such as conjugate gradients)
- Denote W - total number of edges (and weights) in the neural net.

Gradient calculation

- Direct $\nabla E(w)$ calculation, using

$$\frac{\partial E}{\partial w_i} = \frac{E(w+\varepsilon_i) - E(w)}{\varepsilon} + O(\varepsilon) \tag{1}$$

or better

$$\frac{\partial E}{\partial w_i} = \frac{E(w+\varepsilon_i) - E(w-\varepsilon_i)}{2\varepsilon} + O(\varepsilon^2) \tag{2}$$

has complexity:

Gradient calculation

- Direct $\nabla E(w)$ calculation, using

$$\frac{\partial E}{\partial w_i} = \frac{E(w+\varepsilon_i) - E(w)}{\varepsilon} + O(\varepsilon) \tag{1}$$

or better

$$\frac{\partial E}{\partial w_i} = \frac{E(w+\varepsilon_i) - E(w-\varepsilon_i)}{2\varepsilon} + O(\varepsilon^2) \tag{2}$$

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.

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

Table of Contents

Definitions

- Denote w_{ij} be the weight of edge, connecting i-th and j-th neuron.
- Define $\delta_j = \frac{\partial E}{\partial I_j} = \frac{\partial E}{\partial O_j}\frac{\partial O_j}{\partial I_j}$
- Since E depends on w_{ij} through the following functional relationship $E(w_{ij}) \equiv E(O_j(I_j(w_{ij})))$, using the chain rule we obtain:
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial I_j}\frac{\partial I_j}{\partial w_{ij}} = \delta_j O_i$$
because $\frac{\partial I_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k\in inc(j)} w_{kj} O_k\right) = O_i$, where $inc(j)$ is a set of all neurons with outgoing edges to neuron j.
- $\frac{\partial E}{\partial I_j} = \frac{\partial E}{\partial O_j}\frac{\partial O_j}{\partial I_j} = \frac{\partial E}{\partial O_j}\varphi'(I_j)$, where φ is the activation function.

Output layer

- If neuron j belongs to the output node, then error $\frac{\partial E}{\partial O_j}$ is calculated directly.
- For output layer deltas are calculated directly:

$$\delta_j = \frac{\partial E}{\partial O_j}\frac{\partial O_j}{\partial I_j} = \frac{\partial E}{\partial O_j}\varphi'(I_j) \tag{3}$$

- example for training set = {single point x and true vector of outputs $(y_1, ...y_{|OL|})$}:
  - for $E = \frac{1}{2}\sum_{j\in OL}(O_j - y_j)^2$:

$$\frac{\partial E}{\partial O_j} = O_j - y_j$$

  - for sigmoid $\varphi(I) = \sigma(I)$:

$$\varphi'(I_j) = \sigma(I_j)\,(1 - \sigma(I_j)) = O_j(1 - O_j)$$

  - finally

$$\delta_j = (O_j - y_j)\,O_j(1 - O_j)$$

Inner layer

- If neuron j belongs some hidden layer, denote $out(j) = \{k_1, k_2, ...k_m\}$ the set of all neurons, receiving output of neuron j as their input.
- The effect of O_j on E is fully absorbed by $I_{k_1}, I_{k_2}, ...I_{k_m}$, so

$$\frac{\partial E(O_j)}{\partial O_j} = \frac{\partial E(I_{k_1}, I_{k_2}, ...I_{k_m})}{\partial O_j} = \sum_{k \in out(j)} \left(\frac{\partial E}{\partial I_k} \frac{\partial I_k}{\partial O_j} \right) = \sum_{k \in out(j)} (\delta_k w_{jk})$$

- So for layers other than output layer we have:

$$\delta_j = \frac{\partial E}{\partial I_j} = \frac{\partial E}{\partial O_j} \frac{\partial O_j}{\partial I_j} = \sum_{k \in out(j)} (\delta_k w_{jk}) \varphi'(I_j) \tag{4}$$

- Weight derivatives are calculated using errors and outputs:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial I_j} \frac{\partial I_j}{\partial w_{ij}} = \delta_j O_i \tag{5}$$

Backpropagation

- Backpropagation algorithm:
  1. Forward propagate x_n to the neural network, store all inputs I_i and outputs O_i for each neuron.
  2. Calculate δ_i for all $i \in$output layer using (3).
  3. Backpropagate δ_i from final layer backwards layer by layer using (4).
  4. Using calculated deltas and outputs calculate $\frac{\partial E}{\partial w_{ij}}$ with (5).

Backpropagation

- Backpropagation algorithm:
  1. Forward propagate x_n to the neural network, store all inputs I_i and outputs O_i for each neuron.
  2. Calculate δ_i for all $i \in$output layer using (3).
  3. Backpropagate δ_i from final layer backwards layer by layer using (4).
  4. Using calculated deltas and outputs calculate $\frac{\partial E}{\partial w_{ij}}$ with (5).
- Let be W is total number of edges.
- Calculating complexity: $O(W)$
- Memory complexity: $O(W)$
  - need to store inputs and outputs for each node

Backpropagation - optimization

- Optimization updates:
    - batch (only for small N)
    - stochastic
        - using minibatches of objects
        - minibatches - iterative traversal of shuffled training set
        - minibatch size $\propto$ parallelization of CPU

Backpropagation - comments

- Backpropagation correctness is checked by comparing results with (1), (2).
- Allows to finetune neurons on previous layers
    - all network is optimized
    - in contrast:
        - boosting keeps previous trees fixed
        - stacking keeps base learners fixed.

Regularization

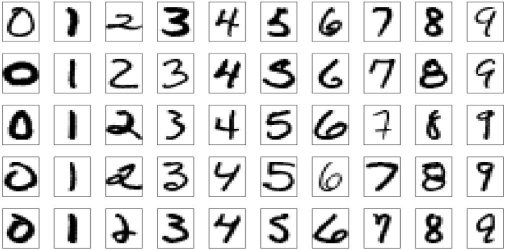
- Constrain model complexity directly
    - constrain number of neurons
    - constrain number of layers
    - impose constraints on weights
- Take a flexible model
    - use early stopping during iterative evaluation (by controlling validation error)
    - quadratic regularization

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

Table of Contents

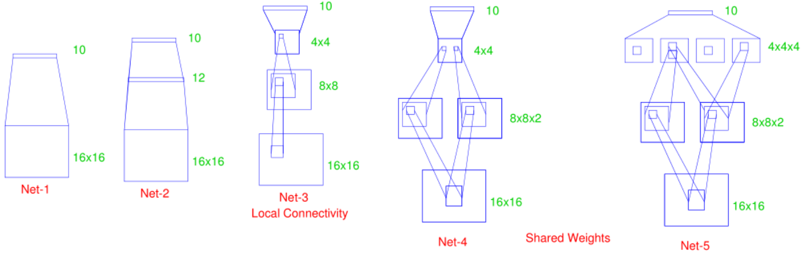
Case study (due to Hastie et al. The Elements of Statistical Learning)

ZIP code recognition task

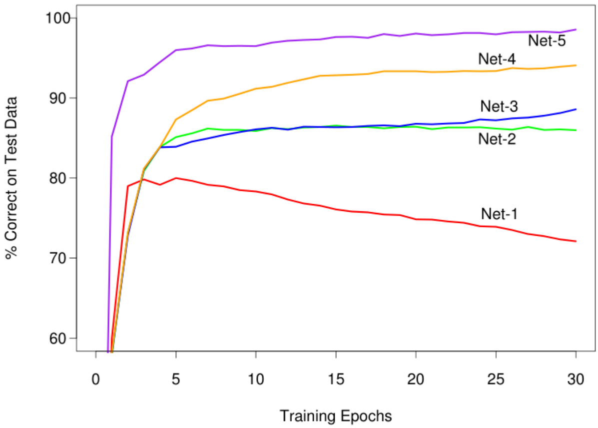


Neural network structures

Net1: no hidden layer
Net2: 1 hidden layer, 12 hidden units fully connected
Net3: 2 hidden layers, locally connected
Net4: 2 hidden layers, locally connected with weight sharing
Net5: 2 hidden layers, locally connected, 2 levels of weight sharing



Results


100
90
80
70
60
% Correct on Test Data
0
5
10
15
20
25
30
Training Epochs
Net-5
Net-4
Net-3
Net-2
Net-1

Addition

- Deep learning
- Neural networks weights may be constrained to belong to mixture density
  - $\tilde{E} \leftarrow E - \lambda P(w)$, where $P(w)$ is the mixture probability of weights
  - soft forcing of weights to group into similar clusters
- 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