Neural networks.

V. Kitov

Yandex School of Data Analysis

Imperial College London
Department of Physics

January 2015

Table of Contents

History

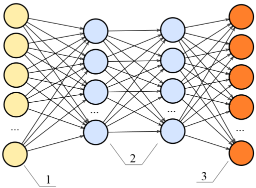
- Neural networks originally appeared as an attempt to model human brain

- 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.

Table of Contents

Definition

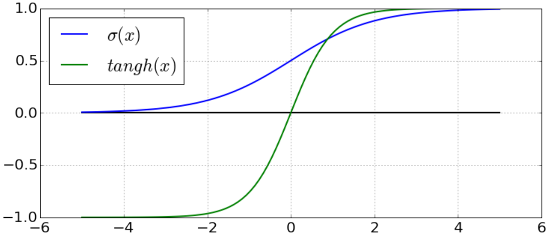
- acyclic directed graph
- verticals called neurons
- edges correspond to certain weighs



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

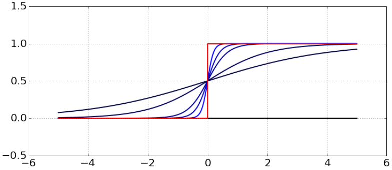
Definition

- Each neuron j is associated a non-linear transformation φ.
- For multilayer perceptron class neural networks φ belongs to a class of activation functions.
- Most common activation functions:
    - sigmoidal: $\sigma(x) = \frac{1}{1+e^{-x}}$
        - 1-layer neural network with sigmoidal activation is equivalent to logistic regression
    - hyperbolic tangent: $\text{tangh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

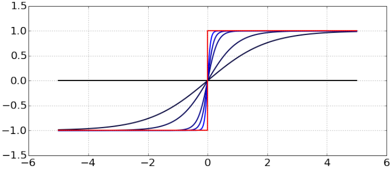


Activation functions

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 $a \to \infty$

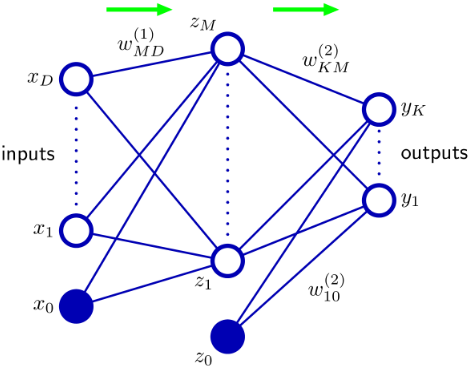
Definition details

- Label each neuron with integer i.
- Denote: I_i - input to neuron i, O_i - output of neuron i
- Output of neuron i: $O_i = A(I_i)$, where A is activation function.
- Input to neuron i: $I_i = \sum_{k \in inc(i)} w_{ki} O_k + w_{k0}$,
    - w_{k0} is the bias term
    - $inc(i)$ is a set of neurons with outgoing edges to neuron i.
    - further we will assume that at each layer there is a vertex with constant output $O_{const} \equiv 1$, so we can simplify notation
    - $$I_i = \sum_{k \in inc(i)} w_{ki} O_k$$

Table of Contents

Output generation

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



Output generation

- Output layer transformations
  - regression: $\varphi(I) = I$
  - classification:
    - 2 classes: sigmoid, indicating target class probability

$$\varphi(I) = \frac{1}{1 + e^{-I}}$$

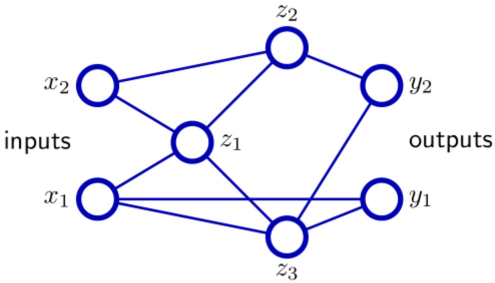
    - multiple classes: softmax, indicating probabilities of each class:

$$\varphi(I_i) = \frac{e^{O_i}}{\sum_{k \in OL} e^{O_k}}, \; i \in OL$$

where OL denotes neuron indices at 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)
- We will consider only continuous activation functions.
- 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 polyhedron
    - 3-layer network can uniformly approximate arbitrary continuous function (as sum of indicators of various 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 fewer amount of neurons
  - model becomes more interpretable and tunable

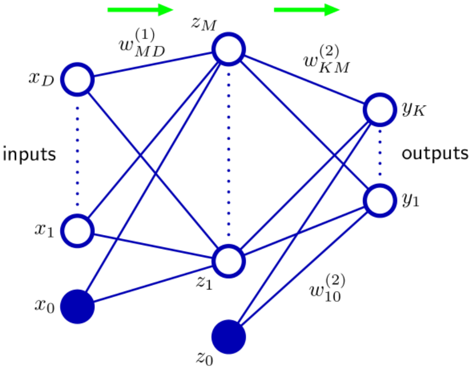
Neural network architecture selection

- Network architecture selection:
  - increasing complexity (control by validation error)
  - decresing complexity (“optimal brain damage”)
    - may be used for feature selection

Table of Contents

Weight space symmetries

- Consider a neural network with 1 hidden layer
  - with *tangh*(x) activation functions
  - consisting of M neurons



Weight space symmetries

- The following transformations in weight space lead to neural networks with equivalent outputs:
  - for any neuron in hidden layer: simultaneous change of sign of input and output weights
    - 2^M possible equivalent transformations of such kind
  - for any pair of neurons in the hidden layer: interchange of input weights between the neurons and simultaneous interchange of output weights
    - this is equivalent to reordering of neurons in the hidden layer, so there are $M!$ such orderings
  - $2^M M!$ equivalent transformations exist in total.
  - For neural network with K hidden layers, consisting of M_k, $k = 1, 2, ...K$ neurons each, we obtain $\prod_{k=1}^{K} 2^{M_k} M_k!$ equivalent neural networks.
  - In general case these are the only symmetries existing in the weights space.

Table of Contents

Network optimization

- Regression (y denotes true value and $\widehat{y}$ - its prediction)
  - single output:
    - $\frac{1}{N}\sum_{n=1}^{N}(\widehat{y}_n(x_n) - y_n)^2 \to \min_w$
  - K outputs
    - $\frac{1}{NK}\sum_{n=1}^{N}\sum_{k=1}^{K}(\widehat{y}_{nk}(x_n) - y_{nk})^2 \to \min_w$
- Classification
  - two class ($y \in \{0, 1\}$ denotes true class, and p is the probability of class 1):
    - $\prod_{n=1}^{N} p(x_n)^{y_n}(1 - p(x_n))^{1-y_n} \to \max_w$ equivalent to $\sum_{n=1}^{N} y_n \ln p(x_n) + (1 - y_n)\ln(1 - p(x_n)) \to \max_w$
  - C classes ($y_{nc} = \mathcal{I}\{y_n = c\}$, $p_c(x_n)$ - estimated probability of class c):
    - $\prod_{n=1}^{N}\prod_{c=1}^{C} p_c(x_n)^{y_{nc}} \to \max_w$ equivalent to $\sum_{n=1}^{N}\sum_{c=1}^{C} y_{nc}\ln p_c(x_n) \to \max_w$

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:

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

- Standardization of features makes gradient descend converge faster
- Other optimization methods are more efficient (conjugate gradients)

Neural network optimization

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

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

or better

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

has complexity $O(W^2)$ [W forward propagations to evaluate W derivatives]

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

Multiple minima problem

- Neural network optimization function has multiple minima
- Solution: select lowest minimum from multiple optimizations with different starting values
- Robust solutions:
  - average outputs of neural networks obtained by using different starting values
  - average outputs of neural networks trained on different bootstrap subsamples

Table of Contents

Backpropagation

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

Backpropagation

- If neuron j belongs to the output node, then error $\delta_j = \frac{\partial E}{\partial O_j}$ is calculated directly.
- For quadratic loss $E = \frac{1}{2}\sum_{j \in OL}(O_j - y_j)^2$: $\delta_j = O_j - y_j$.
- If neuron j belongs some hidden layer, denote $out(j) = \{k_1, k_2, ...k_m\}$ the set of all neurons, receiving output from neuron j.
- The effect of O_j on E is fully absorbed by $I_{k_1}, I_{k_2}, ...I_{k_m}$, so

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

Backpropagation

- For output layer errors are calculated directly:

$$\delta_j = \frac{\partial E}{\partial O_j} \frac{\partial O_j}{\partial I_j} = \frac{\partial E}{\partial O_j} \varphi'(I_i) \tag{1}$$

- For other layers errors are calculated using errors from the next layer:

$$\delta_j = \sum_{k \in out(j)} (\delta_k w_{jk}) \, \varphi'(I_i) \tag{2}$$

- Weight derivatives are calculated using errors and outputs:

$$\frac{\partial E}{\partial w_{ij}} = \delta_j O_i \tag{3}$$

Backpropagation

- Let M be the total number of neurons.
- Backpropagation algorithm:
  1. Forward propagate x_n to the neural network, store all inputs I_i and outputs O_i, $i = 1, 2, \dots M$.
  2. Calculate δ_i for all $i \in OL$ using (1).
  3. Backpropagate δ_i from final layer backwards layer by layer using (2).
  4. Using calculated errors and outputs calculate $\frac{\partial E}{\partial w_{ij}}$ with (3).
- Algorithm complexity: $O(W)$.
- Updates:
  - batch
  - online
    - sequential sampling
    - randomized sampling

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

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

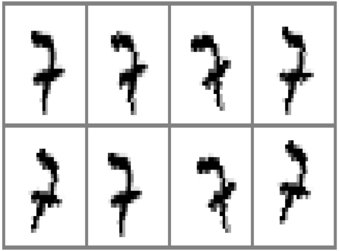
  - alternative regularization (penalizes stronger smaller weights)

$$\tilde{E}(w) = E(w) + \lambda \sum_i w_i^2/(1 + w_i^2)$$

Table of Contents

Invariances

- It may happen that solution should not depend on certain kinds of transformations in the input space.
- Example: character recognition task
  - translation invariance
  - scale invariance
  - invariance to small rotations
  - invariance to small uniform noise

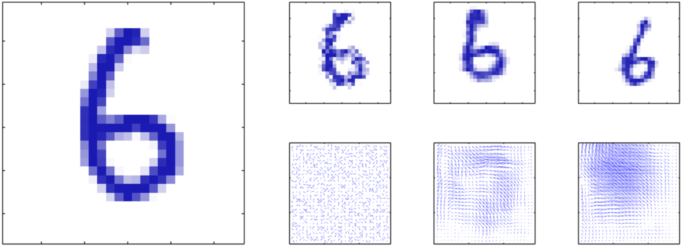


Invariances

- Approaches to build an invariant model:
    - augment training objects with their transformed copies according to given invariances
        - amount of possible transformations grows exponentially with the number of invariances
    - add regularization term to the target cost function, which penalizes changes in output after invariant transformations
        - see tangent propagation
    - extract features that are invariant to transformations
    - build the invariance properties into the structure of neural network
        - see convolutional neural networks

Augmentation of training samples

1. generate a random set of invariant transformations
2. apply these transformations to training objects
3. obtain new training objects



Tangent propagation

- Denote $s(x, \xi)$ be vector x after invariant transformation parametrized by ξ.
- Denote

$$\tau_n = \left.\frac{\partial s(x_n, \xi)}{\partial \xi}\right|_{\xi=0}, \quad J_{ki} = \frac{\partial y_k}{\partial x_i}$$

- We want $\left.\frac{\partial y_k}{\partial \xi}\right|_{\xi=0}$ to be as small, as possible.
- Sensitivity of y_k to small invariant transformation:

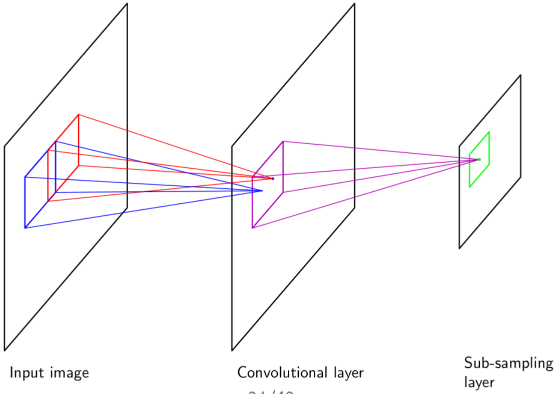
$$\left.\frac{\partial y_k}{\partial \xi}\right|_{\xi=0} = \sum_{i=1}^{D} \frac{\partial y_k}{\partial x_i} \frac{\partial x_i}{\partial \xi} = \sum_{i=1}^{D} J_{ki} \tau_i$$

- Tangent propagation - modify target cost function:

$$\tilde{E} = E + \lambda \sum_n \sum_k \left(\sum_{i=1}^{D} J_{nki} \tau_{ni} \right)^2$$

Convolutional neural networks

- Convolutional neural network:
  - Used for image analysis
  - Consists of a set of convolutional layer / sub-sampling layer pairs and aggregating layer



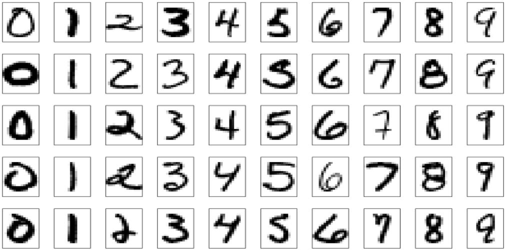
Convolutional neural networks

- Convolutional layer
  - Convolutional layer consists of a number of feature maps
  - Feature map has the same dimensionality as input layer
  - Locality: each neuron in the feature map takes output from small neigborhood of input layer neurons
  - Equivalence: the same transformation is applied by each neuron in the feature map
    - obtained by constraining sets of weights to each feature map layer neuron to be equal
    - similar to convolution with moving adaptive kernel
    - effectively it is feature extraction from a region
- Sub-sampling layer
  - Consists of a number of planes, each corresponding to respective feature map on the previous convolutional layer
  - Locality: Sub-sampling layer neurons take output from small neigborhood of respective feature map neurons
    - neigbourhoods are chosen to be contiguous and

Table of Contents

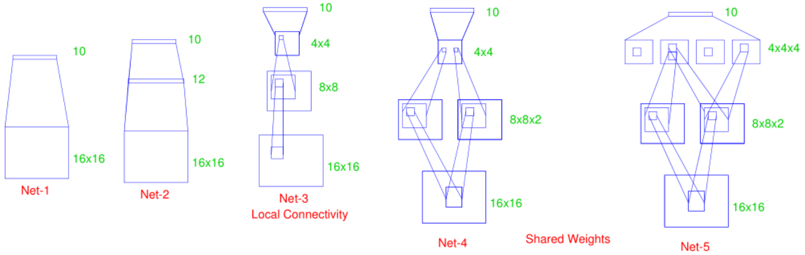
Case study (due to Hastie et al. The Elements of Statistical Learning)

ZIP code recognition task

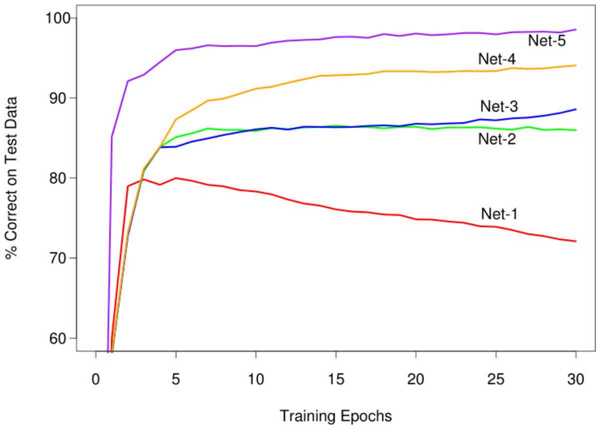


Neural network structures

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

- Neural networks weights may be constrained to belong to mixture density
  - $\tilde{E} \leftarrow E - P(w)$, 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

Further reading

- Further reading on this topic:
  - Pattern Recognition and Machine Learning. Christopher Bishop. Springer. 2007.
  - The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Trevor Hastie, Robert Tibshirani, Jerome Friedman, 2nd Edition, Springer, 2009.