Deep Learning

4. Fully connected neural nets. Backpropagation. Stochactic gradient descent.

Viacheslav Dudar

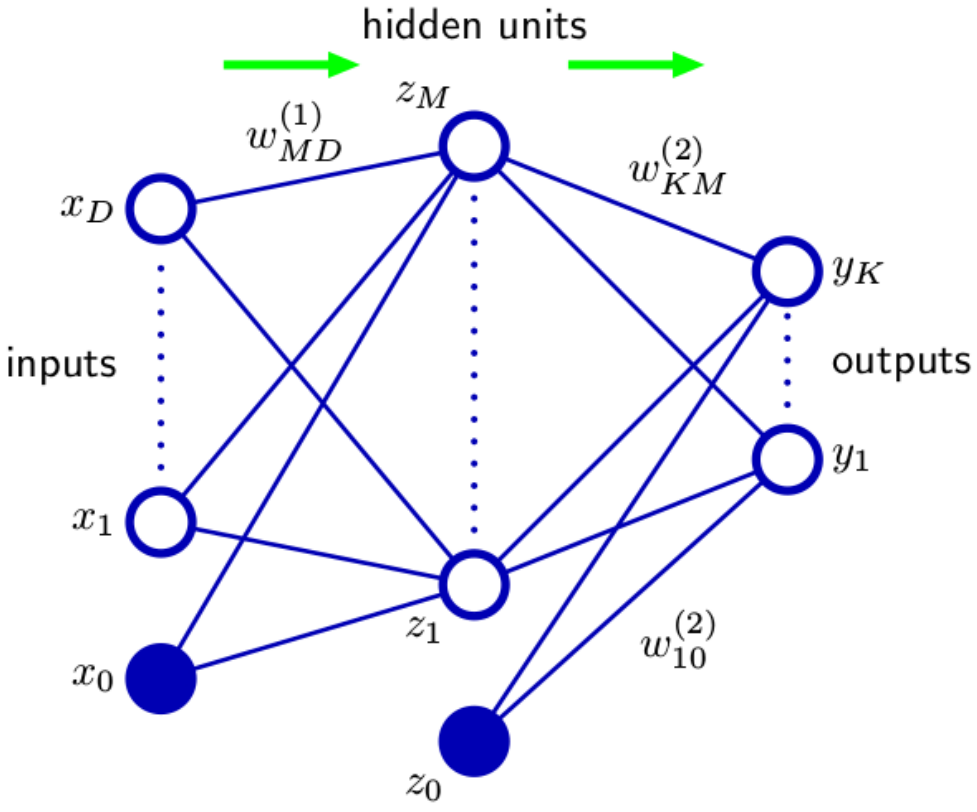
Taras Shevchenko National University of Kyiv

2018

Why linear classifiers are not enough?

We can find nonlinear mappings between input and output with linear classifiers by going to nonlinear feature space.
But it is complicated to find appropriate low-dimensional feature space.
Can be train also feature space?

2-layer neural net for regression



Activations:

$$a_j = \sum_{i=1}^{D} w^{(1)}_{ji} x_i + w^{(1)}_{j0}$$

Hidden units:

$$z_j = h(a_j).$$

Outputs:

$$y_k = \sum_{j=1}^{M} w^{(2)}_{kj} z_j + w^{(2)}_{k0}$$

Matrix form:

$$y = W^{(2)} h(W^{(1)} x)$$

Neural nets for classification

2-layer neural net for binary classification:

$$y(x, w) = \sigma\left(\sum_{j=1}^{M} w_j^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i\right)\right)$$

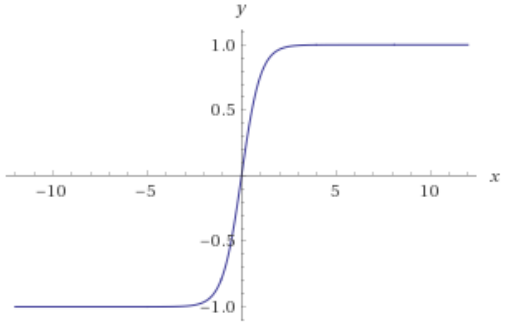
2-layer neural net for multiclass classification:

$$y(x, w) = softmax\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i\right)\right)$$

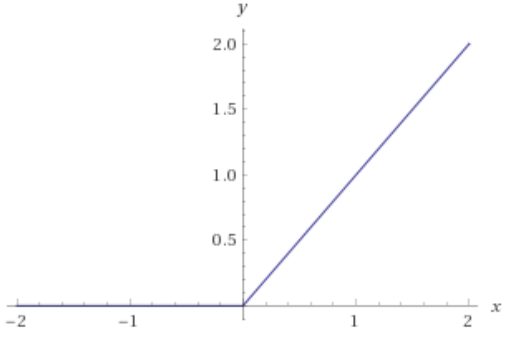
$$y(x, w) = softmax\left(W^{(2)} h(W^{(1)} x)\right)$$

Activation functions

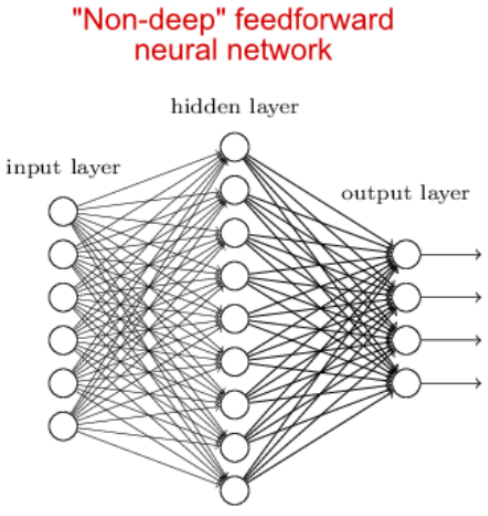
- tanh

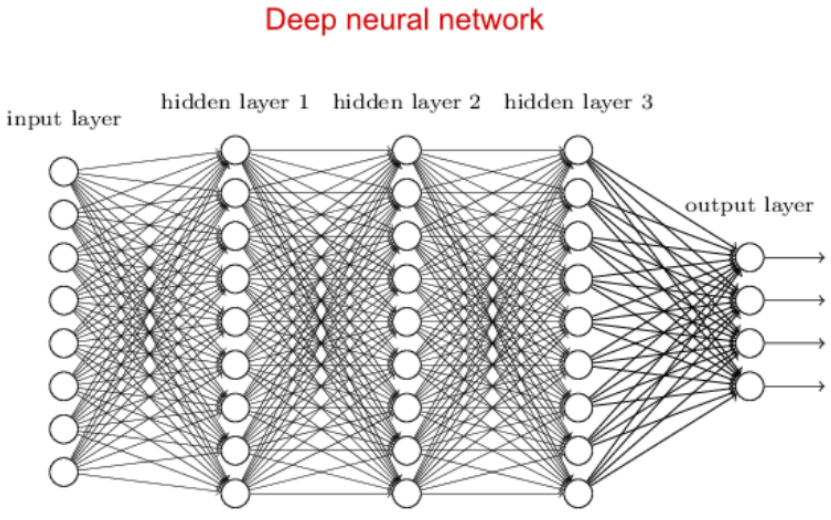


- RELU



Deeper nets





2-layer feed-forward network
"shallow" network

4-layer feed-forward network
"deep" network

Backpropagation

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}).$$

$\Rightarrow$ find gradients for all training elements and sum up.

Stages of gradient calculation:
1) Forward pass (evaluate network output for given input)
2) Backward pass (find derivatives of error function with respect to hidden units)

Backward pass

Starts after forward pass from the last layer and goes backward to the input.

t: target vector

Forward pass	Backward pass
$y = softmax(x)$ (in the last layer)	$\delta_x = y - t$
$y = h(x)$	$\delta_x = h'(x)\delta_y$
$y = Wx$	$\nabla W = \delta_y x^T$ $\delta_x = W^T \delta_y$
General rule for each connection: $y \mathrel{+}= wx$	$\delta_x \mathrel{+}= w\delta_y$ $\nabla w \mathrel{+}= \delta_y x$

These rules work also in case there are skip-layer connections and weights are shared.

Example: gradient calculation for 2-layer net

2-layer feed-forward neural net for classification

Input: x, target vector: t

a_0: activations. z: hidden units, a_1: output activations, y: output

W_0, W_1: matrices of parameters

$\nabla W_0, \nabla W_1$: gradients

Step #	Forward Pass	Backward pass	Step #
0	$a_0 = W_0 x$	$\nabla W_0 = \delta_{a_0} x^T$	8
1	$z = h(a)$	$\delta_{a_0} = h'(a_0)\delta_z$	7
2	$a_1 = W_1 z$	$\delta_z = W_1^T \delta_{a_1}$ $\nabla W_1 = \delta_{a_1} z^T$	6 5
3	$y = softmax(a_1)$	$\delta_{a_1} = y - t$	4

General properties

- Universal approximator:
  2-layer net can approximate any continuous function on a compact set with arbitrary accuracy (but number of hidden units can be large)
- In general higher number of layers leads to higher expressive capability, but there is no strict relation
- Input space is divided into non-linear regions (regions could be disjoint and non-convex)
- RELU activation function: piecewise-linear decision regions and piecewise-linear function

Weight space symmetries

2-layer network with M hidden units:

- Interchange of weights coming in and going out of 2 different hidden units leaves output unchanged $\Rightarrow$ M! equivalent weight vectors.
- Since $tanh(-x) = -tanh(x)$, so changing sign of input and output weights for each hidden unit leaves output unchanged $\Rightarrow 2^M M!$ equivalent weights.

Properties of error function

- Non-convex error function
- Weight symmetry $\Rightarrow$ multiple global minima
- Existence of local minima
- tanh: Vanishing-exploding gradient problem
- RELU: non-differentiable error function
- Huge plateaus of low gradient

SGD

Stochastic gradient descent (SGD):
At each iteration select random subset of size S from training set (called **minibatch**) and evaluate gradient for it:

$$\{j_1 \ldots j_S\} \subset \{1 \ldots N\}$$

$$\widehat{E}(w) = \sum_{s=1}^{S} E_{j_s}(w)$$

Update with stochastic gradient:

$$w_{k+1} = w_k - \alpha_k \nabla \widehat{E}_k(w_k)$$

Epoch: effective pass of training data

Tips for SGD

- Random initiation of weights: network initiated at zero will not train!
- Choose reasonable minibatch size: small $\Rightarrow$ big oscilations, big $\Rightarrow$ slow movement
- Usually minibatch size is 100 - 1000
- Balance minibatches for classes
- Calculate error function on validation set to adjust learning rate during training

RMSPROP

Problem with momentum: gets stucked on plateaus of low gradient.
Solution: normalize gradient.
Direct normalization $\Rightarrow$ unstable trajectory.
Solution: running average.

$$MS_{k+1} = 0.9MS_k + 0.1\left(\widehat{E}_k(w_k)\right)^2$$

$$w_{k+1} = w_k - \alpha_k \frac{\widehat{E}_k(w_k)}{\sqrt{MS_{k+1} + \varepsilon}}$$

ADAM

Combine gradient normalization and momentum.

$$MS_{k+1} = 0.9MS_k + 0.1\left(\widehat{E}_k(w_k)\right)^2$$

$$v_{k+1} = 0.9v_k + 0.1\widehat{E}_k(w_k)$$

$$w_{k+1} = w_k - \alpha_k \frac{v_{k+1}}{\sqrt{MS_{k+1} + \varepsilon}}$$

Most popular method for training all kinds of neural nets.
Practical values of α : $0.0001 - 0.05$.
Works well if α is slowly decreased with iterations.

Speeding up tips

- Normalize inputs: make means 0 and variances 1 for each input component over the training set, or make full decorrelation
- Explore more advanced methods for initialization of weights
- Use parallel computing and GPUs! (will be explored in next lectures)