



Learning &
Adaptive Systems

Neural Networks

Zhenrong Lang¹ and Rajesh Sharma²

Introduction to Machine Learning

¹Department of Information Technology and Electrical Engineering, ETH Zürich

²Department of Computer Science, ETH Zürich

Where we are

- Lectures
 - Week 5 Kernels
 - Week 6 (**This Week!**) **Neural Networks I, II**
 - Week 7 Neural Networks III, IV
- Tutorials
 - Week 5 Classification and Model Selection
 - Week 6 (**Today!**) **Neural Network Basics**
 - Week 7 Homework Review, CNN and other Networks

Outline

- First session (Zhenrong)
 - Motivation for neural networks
 - Multilayer perceptrons
 - Activation functions
 - Forward and backward propagation
 - Gradient Descent
- Second session (Rajesh)
 - Code a simple neural network
 - [Click here for Notebook](#)
 - Data Generation and Visualization
 - Setting up Weights and Biases
 - Training Loop
 - Updating the network
 - Visualizing the results
 - Keras/Tensorflow demo

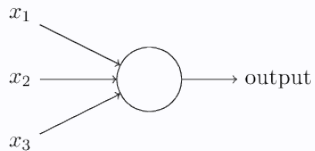
Motivation for neural networks

Motivation for neural networks

Up till now:

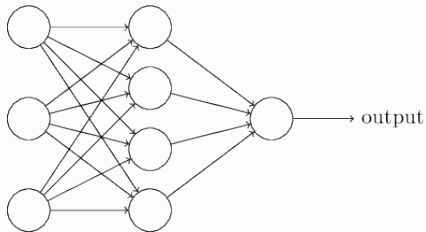
Linear:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$



Non-linear:

$$f(\mathbf{x}) = g(\phi(\mathbf{x})) \text{ with } g(\mathbf{z}) = \mathbf{w}^T \mathbf{z}$$

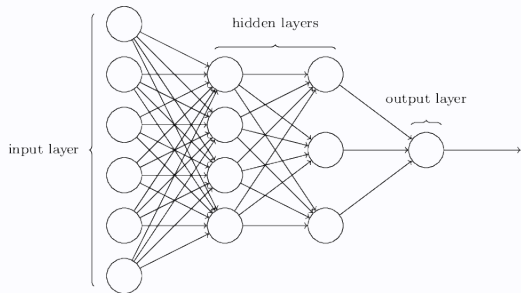


The features are hand-designed (chosen manually).

Goal: learn features from data automatically.

Multilayer Perceptrons

Multilayer Perceptrons



Node (neuron/perceptron)

pre-activation $z = \mathbf{w}^T \mathbf{v}$

apply activation: $v = \varphi(z)$

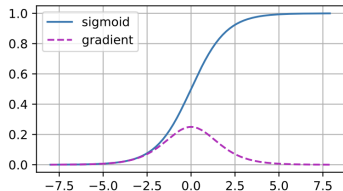
Bias-trick:

$$\mathbf{w} = (w_0, w_1, \dots, w_{d-1})^T$$

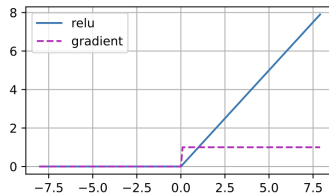
$$\mathbf{x} = (1, x_1, \dots, x_{d-1})^T$$

Activation Functions

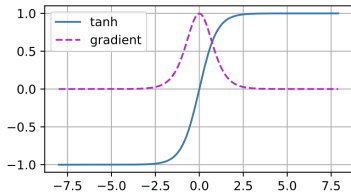
sigmoid



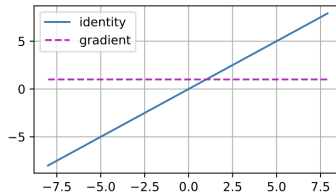
relu



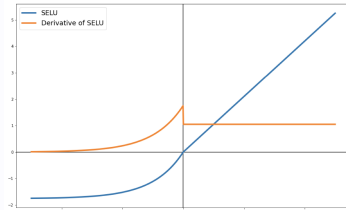
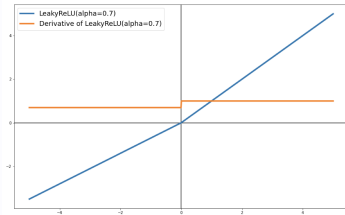
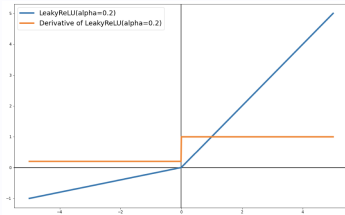
tanh



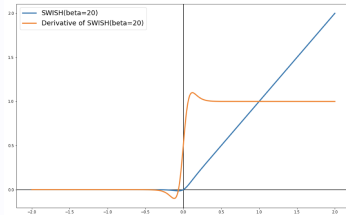
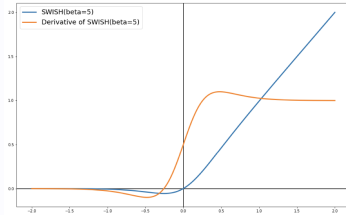
identity (is this a good activation function?)



Activation Functions

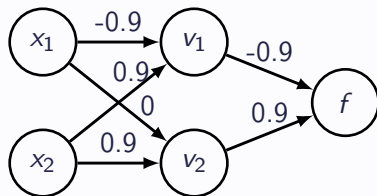


$$\alpha = 1.6733; \lambda = 1.0507$$



Forward and backward propagation

Forward propagation with a numerical example



activation function: ReLU

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

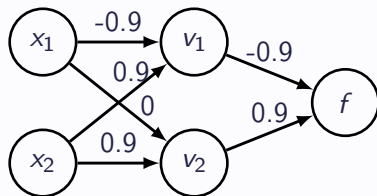
Pre-activation:

$$\mathbf{z} = W^{(1)}\mathbf{x}$$
$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} W_{11}^{(1)} & W_{12}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$
$$\begin{pmatrix} 0 \\ 0.9 \end{pmatrix} = \begin{pmatrix} -0.9 & 0.9 \\ 0 & 0.9 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Hidden Layer:

$$\mathbf{v} = \varphi(\mathbf{z})$$
$$\begin{pmatrix} 0 \\ 0.9 \end{pmatrix} = \begin{pmatrix} \varphi(0) \\ \varphi(0.9) \end{pmatrix}$$

Forward propagation with a numerical example (cont.)



activation function: ReLU

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

Output

$$f = W^{(2)} \mathbf{v}$$

$$f = \begin{pmatrix} W_{11}^{(2)} & W_{12}^{(2)} \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

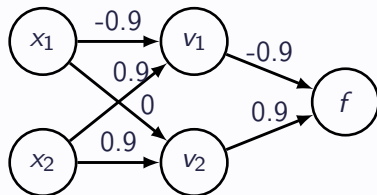
$$0.81 = \begin{pmatrix} -0.9 & 0.9 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0.9 \end{pmatrix}$$

Mean Square Error:

$$l = (y - f)^2$$

$$0.0361 = (1 - 0.81)^2$$

Backward propagation with a numerical example

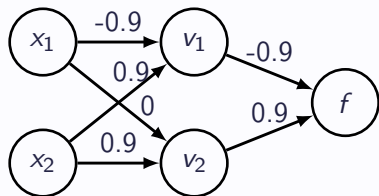


activation function: ReLU

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

$$\begin{aligned}\frac{\partial l}{\partial W^{(2)}} &= \frac{\partial l}{\partial f} \cdot \frac{\partial f}{\partial W^{(2)}} \\ &= \frac{\partial (y - f)^2}{\partial f} \cdot \frac{\partial W^{(2)} \mathbf{v}}{\partial W^{(2)}} \\ &= \begin{pmatrix} 0 \\ 0.9 \end{pmatrix} \cdot 2(0.81 - 1) = \begin{pmatrix} 0 \\ -0.342 \end{pmatrix} \\ W_{new}^{(2)} &= W^{(2)} - \eta \nabla_{W^{(2)}} l(W, x, y) \\ &= W^{(2)} - \eta \left(\frac{\partial l}{\partial W^{(2)}} \right)^T \\ &= \begin{pmatrix} -0.9 & 0.9 \end{pmatrix} - 0.2 \cdot \begin{pmatrix} 0 & -0.342 \end{pmatrix} \\ &= \begin{pmatrix} -0.9 & 0.9684 \end{pmatrix}\end{aligned}$$

Backward propagation with a numerical example (cont.)

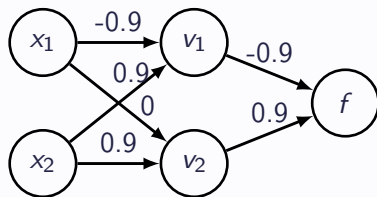


activation function: ReLU

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

$$\begin{aligned}\frac{\partial l}{\partial W^{(1)}} &= \frac{\partial l}{\partial f} \cdot \frac{\partial f}{\partial \mathbf{v}} \cdot \frac{\partial \mathbf{v}}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial W^{(1)}} \\&= \frac{\partial (y - f)^2}{\partial f} \cdot \frac{\partial W^{(2)} \mathbf{v}}{\partial \mathbf{v}} \cdot \frac{\partial \varphi(\mathbf{z})}{\partial \mathbf{z}} \cdot \frac{\partial W^{(1)} \mathbf{x}}{\partial W^{(1)}} \\&= \mathbf{x} \cdot 2(f - y) \cdot W^{(2)} \cdot \text{diag}(\varphi'(\mathbf{z})) \\&= \begin{pmatrix} 1 \\ 1 \end{pmatrix} \cdot 2(0.81 - 1) \cdot \begin{pmatrix} -0.9 & 0.9 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \\&= -0.38 \cdot \begin{pmatrix} -0.9 & 0.9 \\ -0.9 & 0.9 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -0.342 \\ 0 & -0.342 \end{pmatrix}\end{aligned}$$

Backward propagation with a numerical example (cont.)



activation function: ReLU

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

$$\begin{aligned}W_{new}^{(1)} &= W^{(1)} - \eta \nabla_{W^{(1)}} l(W, x, y) \\&= W^{(1)} - \eta \left(\frac{\partial l}{\partial W^{(1)}} \right)^T \\&= \begin{pmatrix} -0.9 & 0.9 \\ 0 & 0.9 \end{pmatrix} - 0.2 \cdot \begin{pmatrix} 0 & 0 \\ -0.342 & -0.342 \end{pmatrix} \\&= \begin{pmatrix} -0.9 & 0.9 \\ 0.0684 & 0.9684 \end{pmatrix} \\&\begin{pmatrix} -0.9 & 0.9 \\ 0.0684 & 0.9684 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1.0368 \end{pmatrix} \\&\begin{pmatrix} -0.9 & 0.9684 \end{pmatrix} \cdot \begin{pmatrix} \varphi(0) \\ \varphi(1.0368) \end{pmatrix} = 1.004\end{aligned}$$

Gradient Descent

Gradient Descent

Recall the following Gradient Descent Methods:

(Batch) Gradient Descent: uses all examples in the training data.

Stochastic Gradient Descent: uses one randomly chosen example.

Mark the following as (T) rue or (F) alse:

- Batch Gradient Descent is suited for large number of training data.
- Batch Gradient Descent gives the global optimal solution given sufficient time.
- Stochastic Gradient Descent tends to escape local minima.
- Stochastic Gradient Descent reaches convergence faster than Batch Gradient Descent.
- Batch Gradient Descent is faster and less computationally expensive.

Neural Networks and Deep Learning by Michael Nielsen

Dive into Deep Learning by Aston Zhang

Deep Learning by Ian Goodfellow

Backup slide: numerator layout

Scalar by vector

$$\frac{\partial y}{\partial \mathbf{x}} = \left[\frac{\partial y}{\partial x_1} \quad \frac{\partial y}{\partial x_2} \quad \cdots \quad \frac{\partial y}{\partial x_n} \right]$$

$$\frac{\partial \mathbf{a}^\top \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}^\top$$

Vector by vector

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

$$\frac{\partial A \mathbf{x}}{\partial \mathbf{x}} = A$$

Scalar by matrix

$$\frac{\partial y}{\partial X} = \begin{bmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{21}} & \cdots & \frac{\partial y}{\partial x_{p1}} \\ \frac{\partial y}{\partial x_{12}} & \frac{\partial y}{\partial x_{22}} & \cdots & \frac{\partial y}{\partial x_{p2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial x_{1q}} & \frac{\partial y}{\partial x_{2q}} & \cdots & \frac{\partial y}{\partial x_{pq}} \end{bmatrix}$$

$$\frac{\partial \mathbf{a}^\top X \mathbf{b}}{\partial X} = \mathbf{b} \mathbf{a}^\top$$