Feedforward Neural Networks

Yagmur Gizem Cinar, Eric Gaussier

AMA, LIG, Univ. Grenoble Alpes

30 Septembre 2020

Reference Book

Deep Learning
Ian Goodfellow and Yoshua Bengio and Aaron Courville
MIT Press
2016

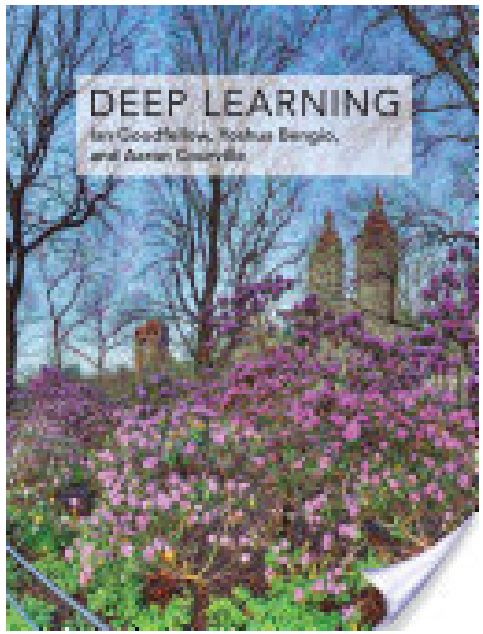


Table of Contents

Multilayer Perceptrons (MLP)

Feedforward Neural Networks, Deep feedforward Networks

Goal

to approximate function f^*

$$y = f^*(\mathbf{x}) \tag{1}$$

- Classification $y \in \{c_1, c_2, \dots c_K\}$
- Regression $y \in \mathbb{R}$

A feedforward network

$$y = f(\mathbf{x}; \theta) \tag{2}$$

Feedforward: $\mathbf{x}$ through f and finally y
No feedback connections as **recurrent neural network**

Multilayer Perceptrons (MLP)

Feedforward Neural Networks

- network: composing different functions
- a directed acyclic graph
- e.g. $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$

  $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$

  $f^{(1)}$ is 1^{st} layer
  $f^{(2)}$ is 2^{nd} layer
- final layer is called **output layer**
- other layers are called **hidden layers**
- length of the chain is the **depth** of the network
- **width** is the dimensionality of the hidden layers

Example: Learning XOR

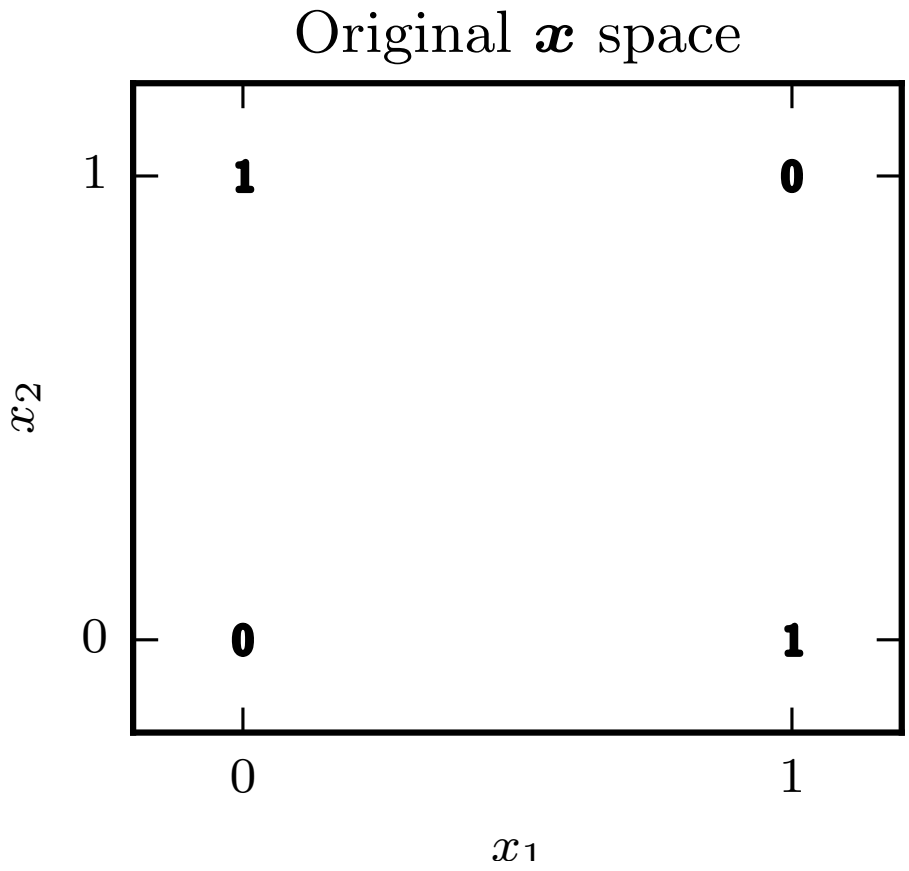


Figure 1: XOR in $\mathbf{x}$ space[1].

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

Example: Learning XOR

- $\mathbb{X} = \{[0,0]^\mathsf{T}, [0,1]^\mathsf{T}, [1,0]^\mathsf{T}, [1,1]^\mathsf{T}\}$
- XOR is not linearly separable
- XOR target function $y = f^*(\mathbf{x})$
- model function $y = f(\mathbf{x}; \theta)$
- XOR MSE loss function

$$J(\theta) = \frac{1}{4} \sum_{\mathbf{x} \in \mathbb{X}} (f^*(\mathbf{x}) - f(\mathbf{x}; \theta))^2$$

- If model is a linear single-layer with one unit

$$f(\mathbf{x}; \theta) = \mathbf{x}^\mathsf{T} \mathbf{w} + b$$

Example: Learning XOR

A single-layer with one hidden unit also called perceptron:

$$f(\mathbf{x};\theta) = \mathbf{x}^{\mathsf{T}}\mathbf{w} + b$$

- cannot separate XOR

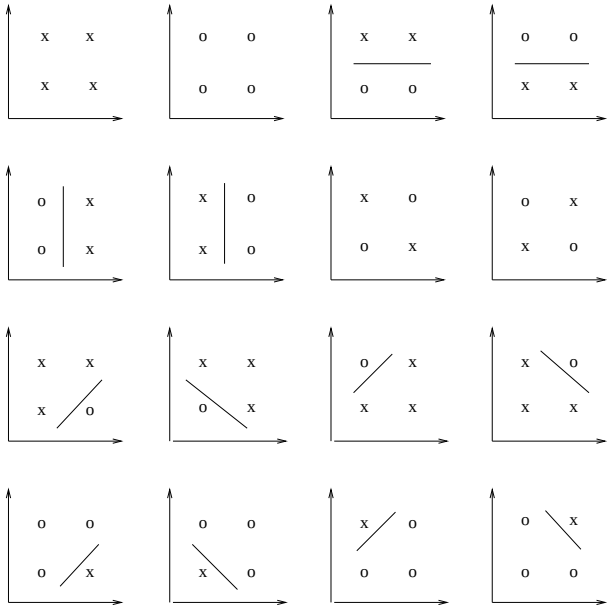


Figure 2: XOR is not linearly separable[2].

[2] Johan Suykens. *Lecture notes in Artificial Neural Networks*. 2015.

Example: Learning XOR

A single layer with two hidden units, and the output layer

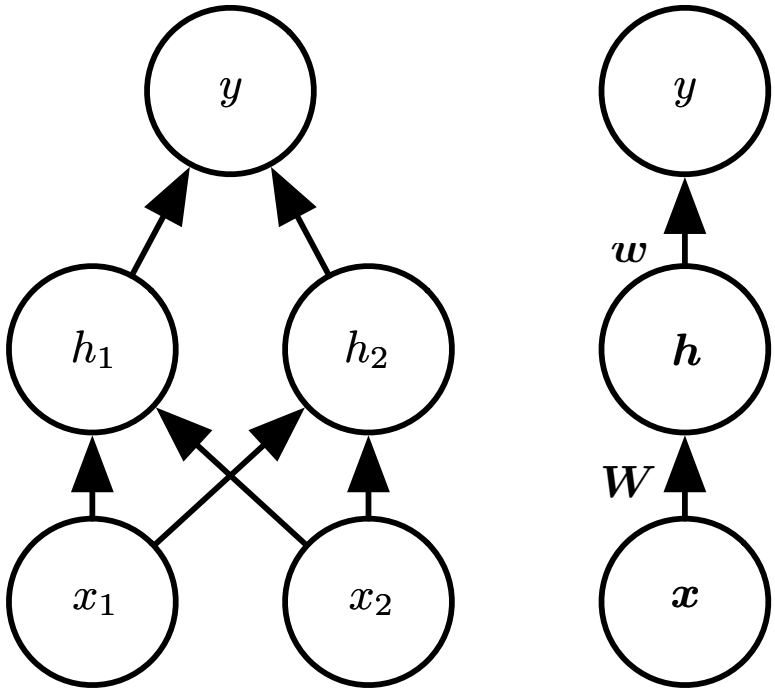


Figure 3: Network diagrams[3].

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

Example: Learning XOR

One hidden layer with two hidden units, and the output layer

$$\mathbf{h} = f^{(1)}(\mathbf{x}; \mathbf{W}, \mathbf{c})$$
$$y = f^{(2)}(\mathbf{h}; \mathbf{w}, b)$$
$$f(\mathbf{x}, \mathbf{W}, \mathbf{c}, \mathbf{w}, b) = f^{(2)}(f^{(1)}(\mathbf{x}))$$

$\mathbf{W}$ and $\mathbf{w}$ weights of a linear transformation
b and $\mathbf{c}$ biases

$$f^{(1)}(\mathbf{x}) = \mathbf{W}^{\mathsf{T}}\mathbf{x}$$
$$f^{(2)}(\mathbf{h}) = \mathbf{h}^{\mathsf{T}}\mathbf{w}$$
$$f(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{W}^{\mathsf{T}}\mathbf{x}$$

(intercept/bias terms ignored)

Example: Learning XOR

For a nonlinearity: **activation function** g

$$\mathbf{h} = g(\mathbf{W}^{\mathsf{T}}\mathbf{x} + c)$$

A rectified linear unit (ReLU) is the activation function for many feedforward networks

$$g(z) = \max\{0, z\}$$

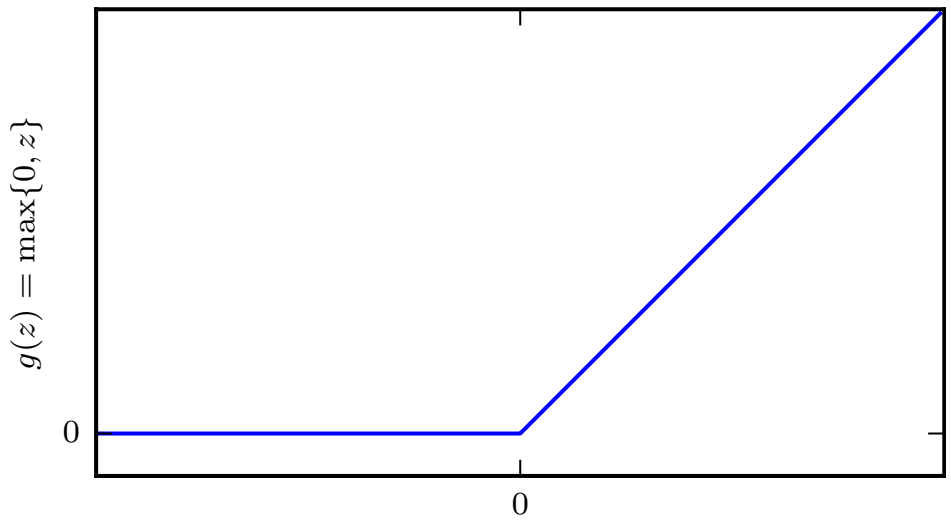


Figure 4: ReLU activation function[4].

Example: Learning XOR

Complete network

$$f(\mathbf{x}, \mathbf{W}, \mathbf{c}, \mathbf{w}, b) = \mathbf{w}^{\mathsf{T}} \max\{0, \mathbf{W}^{\mathsf{T}}\mathbf{x} + c\} + b$$

Let

$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$b = 0$$

Design matrix $\mathbf{X}$

$$\mathbf{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$\mathbf{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$$

$$\mathbf{XW} + \mathbf{c} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

$$\max\{0, \mathbf{XW} + \mathbf{c}\} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

$$\mathbf{w}^{\mathsf{T}} \max\{0, \mathbf{XW} + \mathbf{c}\} + b = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Gradient-Based Learning

- In real life billions of model parameters
- Gradient-based optimization algorithm provide solution with little error
- Trained by iterative gradient-based optimizers

Gradient-Based Learning

- Cost function:

$$J(\mathbf{w}, b) = -\mathbb{E}_{\mathbf{x},y \ \hat{p}_{\text{data}}} \log p_{\text{model}}(y|\mathbf{x})$$

- Mostly negative log-likelihood as a cost function
- So, minimizing the cost leads to maximum likelihood estimation
- Cross entropy between the training data and model's prediction as a cost function
- Typically total cost composed of cross entropy and regularization
- regularization terms (weight decay)

$$J(\mathbf{w}, b) = \lambda ||\mathbf{x}||_2^2 - \mathbb{E}_{\mathbf{x},y \ \hat{p}_{\text{data}}} \log p_{\text{model}}(y|\mathbf{x})$$

- mean squared error (MSE)

$$f^* = \underset{f}{\text{argmin}} \, \mathbb{E}_{x,y \ p_{\text{data}}} ||y - f(\mathbf{x})||^2$$

Gradient-Based Learning

Output units

The choice of output function determines the cross entropy

Output Type	Output Distribution	Output Layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary cross-entropy
Discrete	Multinoulli	Softmax	Discrete cross-entropy
Continuous	Gaussian	Linear	Gaussian cross-entropy (MSE)

Figure 5: Output units[5].

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

Hidden Units

Hidden units

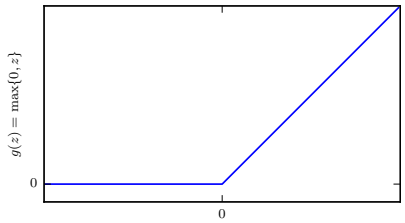
- input vectors $\mathbf{x}$
- computing an affine transformation $z = \mathbf{W}^{\mathsf{T}}\mathbf{x} + \mathbf{b}$
- element-wise nonlinear function $g(z)$

Most hidden units are distinguished by the choice of the activation function g(z)

Activation Functions

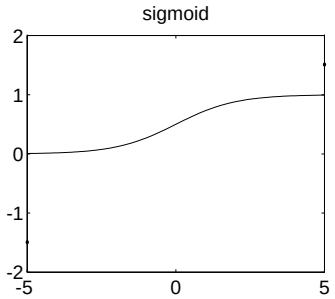
- Rectified Linear units (ReLU)

$$g(z) = \max\{0, z\}$$



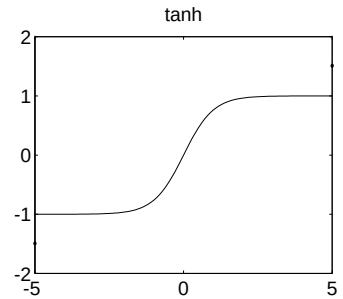
- Logistic sigmoid σ

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



- Hyperbolic tangent tanh

$$\tanh(z) = \frac{1 - \exp(-2z)}{1 + \exp(-2z)}$$



Architecture Design

Architecture

- Mostly in a chain structure

$$\mathbf{h}^{(1)} = g^{(1)}(\mathbf{W}^{(1)\mathsf{T}}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{h}^{(2)} = g^{(2)}(\mathbf{W}^{(2)\mathsf{T}}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$$

- Main design choices
  - depth of the network
  - width of each layer
- Deeper networks
  - fewer units per layer
  - fewer parameters
  - tend to be harder to optimize
- Ideal network architecture via experimentation guided by monitoring the validation error

Back-Propagation

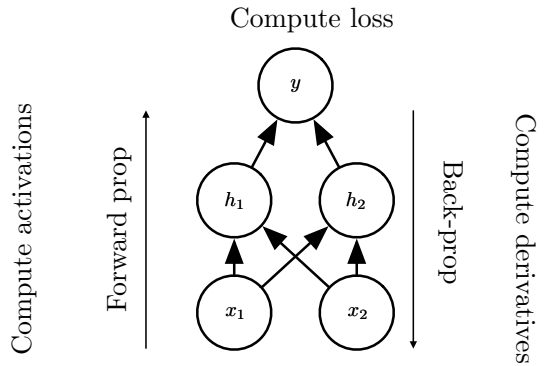
- Forward propagation is flow from $\mathbf{x}$ to $\hat{\mathbf{y}}$
- During training forward propagation continue onward until cost $J(\theta)$
- **backprop** from cost $J(\theta)$ to network backwards to compute the gradient
- Backprop is a method of computing gradient
- Backprop makes it simple and inexpensive

Back-propagation

Computational Graphs

- Each node a variable
- A variable might be scalar, vector, matrix or tensor
- An **operation** a simple function of one or more variables
- Functions more complex, composed of many operations
- directed edge from x to y indicates x used to calculate y

Simple Back-Prop Example



Back-propagation

Examples of Computational Graphs

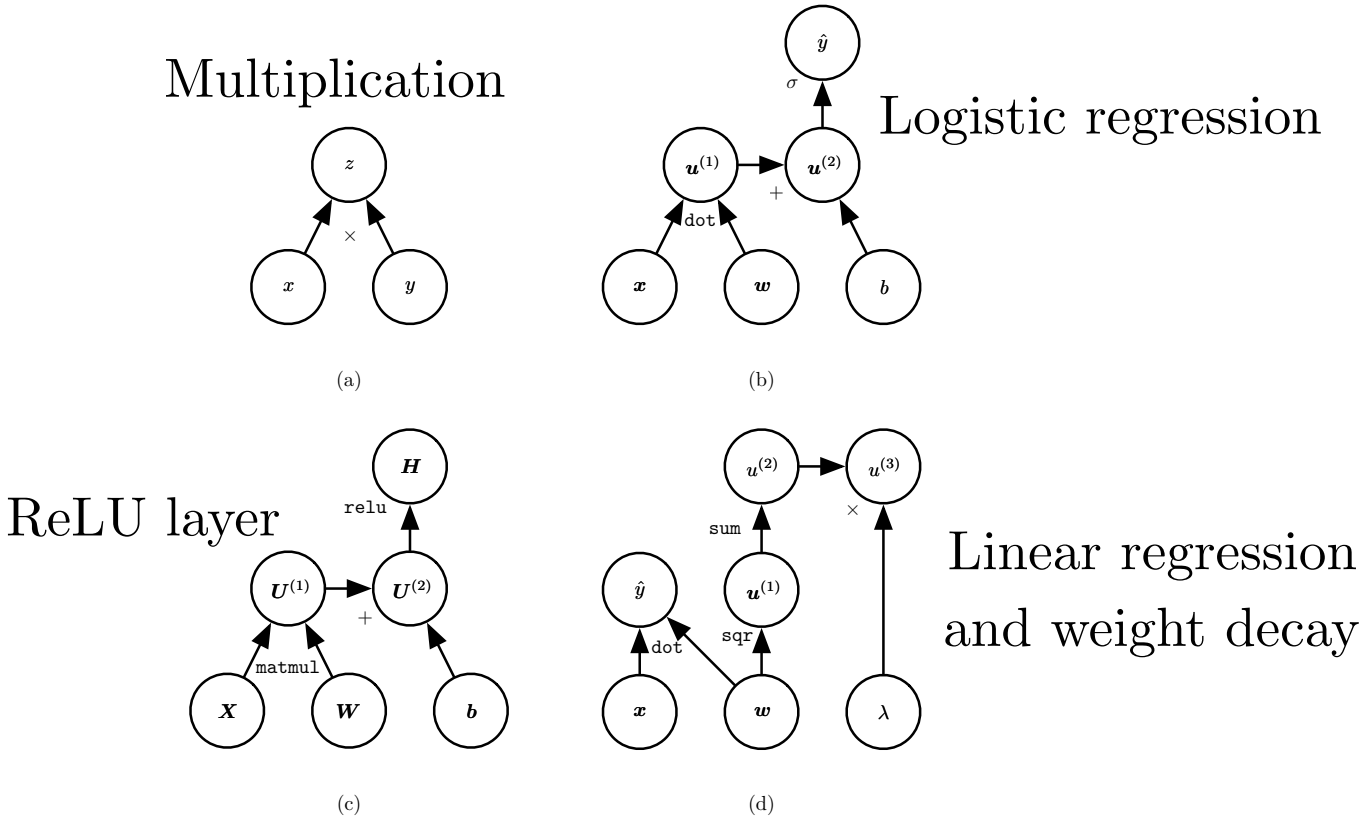


Figure 6: Computation Graphs[6].

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

Back-propagation

Back-propagation is a chain rule of calculus

- Highly efficient
- x is a real number and $f, g : \mathbb{R} \to \mathbb{R}$, $y = g(x)$ and $z = f(g(x)) = f(y)$
- Chain rule:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

- For $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, $g : \mathbb{R}^m \to \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x}$$

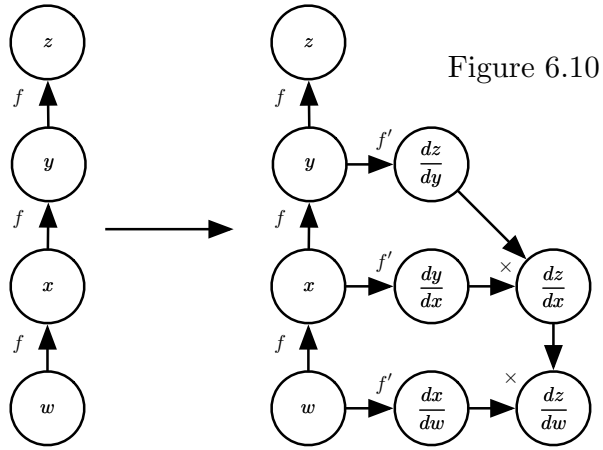


Figure 6.10

Figure 7: Symbol-to-symbol example

Questions?

Thank you!

References


Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.


Johan Suykens. *Lecture notes in Artificial Neural Networks*. 2015.