Mathematics for Machine Learning

Lecture 11
(04.07.2024)

Multi-layer Perceptron

Mahammad Namazov

Table of contents

Introduction

DNN

UAT

Architecture

Deep Neural Networks

- Deep Neural Networks can be seen as cascade combination of neural network blocks, where one’s output is the input for following. In such representation:
  - Each block has a predefined structure, and is defined based on **unknown parameters**;
  - We train the model to find these **unknown parameters**;
  - Training’s purpose is to minimize the **loss**;
  - **Minimization** is done by **backpropagation**

- Model architecture can be written as composition of functions as below:

$$\hat{\mathbf{y}} \leftarrow (\sigma \circ f) \circ (\sigma \circ f) \circ (\sigma \circ f) \circ \cdots \circ (\sigma \circ f) \circ (\sigma \circ f)\mathbf{x}$$

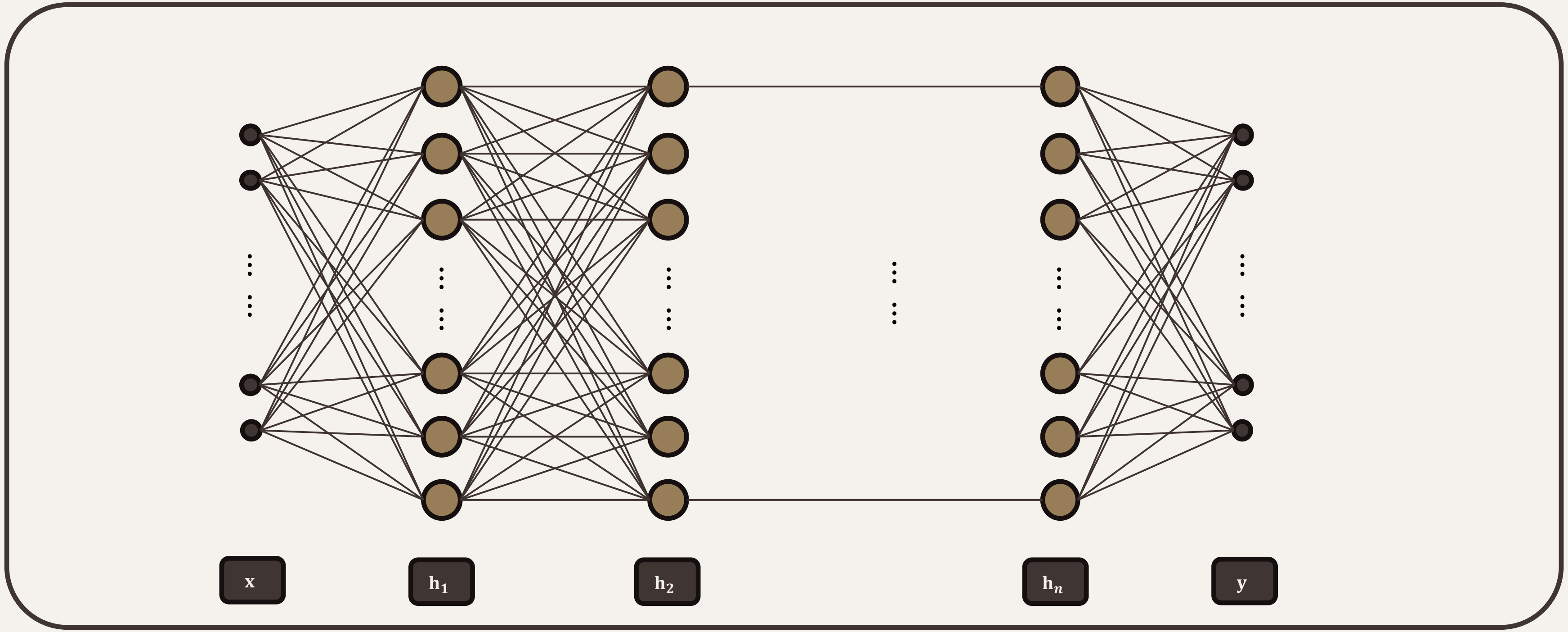
- σ brings non-linearity to the table, and $\sigma \circ f$ is considered as a block;

- Note: σ here is not necessarily a sigmoid function, but represents activation functions;

- Once we include unknown parameters to the equation:

$$\hat{\boldsymbol{y}}_{\boldsymbol{\theta}} \leftarrow \left(\sigma \circ f_{\boldsymbol{\theta}_n}\right) \circ \left(\sigma \circ f_{\boldsymbol{\theta}_{n-1}}\right) \circ \cdots \circ \left(\sigma \circ f_{\boldsymbol{\theta}_2}\right) \circ \left(\sigma \circ f_{\boldsymbol{\theta}_1}\right)\mathbf{x}$$

Architecture



Universal Approximation Theorem

- Theorem says:
- For any compact set $\Omega \subset \mathbb{R}^p$, the space spanned by the functions $\phi(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$ is dense in $\mathcal{C}(\Omega)$ for the uniform convergence. Thus, for any continuous function f and any $\epsilon > 0$, there exists $q \in \mathbb{N}$ and weights stating that:

$$|f(\mathbf{x}) - \sum_{k}^{q} u_k \phi(\mathrm{x})| \leq \epsilon \text{, for all } \mathbf{x} \in \Omega$$

- If the given function is continuous, we will find such weights for a network that will approximate the given function;

- Increasing number of layer can decrease the error;

Working Principle

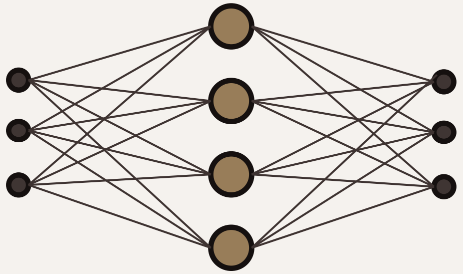
Feedforward

Loss Computation

Backpropagation

Feed forward

- You have 3-dimensional input vector, and you want to classify your data with respect to 3 labels;
- For simplicity (?) we will have one hidden layer with sigmoid function as activation function:

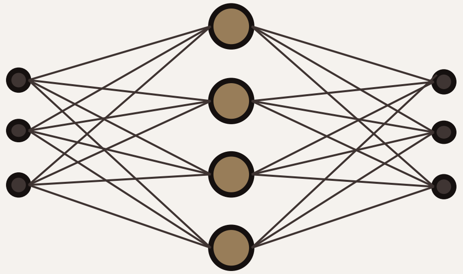


Let's set notation:

- Input layer will be noted as h_0;
- Output layer will be noted as h_{n+1};
- W^k: weight matrix to connect $(k-1)^{th}$ layer to k^{th} layer ;
- $w_{i,j}^{(k)}$: weight connects i^{th} unit of $(k-1)^{th}$ layer to j^{th} unit of k^{th} layer

Feed forward

- You have 3-dimensional input vector, and you want to classify your data with respect to 3 labels;
- For simplicity (?) we will have one hidden layer with sigmoid function as activation function:



Computation

- $\mathbf{z}^{(1)} = \left(\mathbf{W}^{(1)}\right)^{\mathbf{T}}\mathbf{x} + \mathbf{b}^{(1)}$;
- $\mathbf{h}^{(1)} = \sigma\left(\mathbf{z}^{(1)}\right)$;
- $\mathbf{z}^{(2)} = \left(\mathbf{W}^{(2)}\right)^{\mathbf{T}}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}$
- $\mathbf{y} = softmax(\mathbf{z}^{(2)})$

Activation Functions

- ReLU, Sigmoid, Softmax, Leaky ReLU and etc.;
- We focus on 2 of them:

Sigmoid

- **Given input vector produces values between 0 and 1;**
- For $\mathbf{x} \in \mathbb{R}^n$:

$$\mathbf{sigmoid}(x) = \frac{\mathbf{1}}{\mathbf{1} + \boldsymbol{exp}(-x)}$$

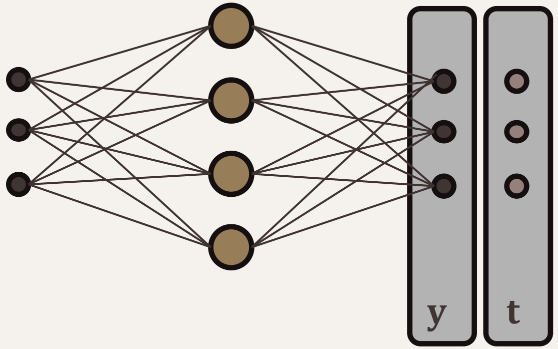
Softmax

- **Given input vector, generates probability distribution;**
- For $\mathbf{x} \in \mathbb{R}^n$:

$$\mathbf{softmax}(\mathbf{x})_i = \frac{exp(x_i)}{\sum_k^n exp(x_k)}$$

Loss Computation

- When we have classification tasks, we usually use Categorical Cross Entropy Loss as an objective function to minimize;
- Once you have k labels and you want to classify your data according to them:



- Loss will be computed for chosen data:

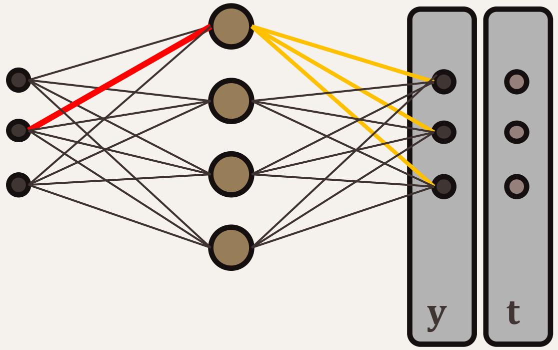
$$L_{CE} = -\sum_{i}^{k} t_i \log(y_i)$$

- If chosen data belongs to the class which index is $j \in [1, k]$:

$$\mathbf{t} = \begin{bmatrix} 0 & 0 & \cdots & \underset{\uparrow \\ \boxed{t_j}}{1} & \cdots & 0 \end{bmatrix}^T$$

Backward computation

- Once we computed the loss function, we need to minimize it, since we need to find such parameters that make our model fit to the data;



- How does $w_{2,1}^{(1)}$ impact our loss?

$$\frac{\partial L}{\partial w_{2,1}^{(1)}} = \frac{\partial L}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial w_{2,1}}$$

Summary

- Universal Approximation Theorem says that NN can approximate any continuous function;
- Co-domain of the NN is defined by the last block of the neural network;
- The more the number of layers is the deeper the network;
- Increasing model complexity can decrease the error, but also might cause the overfitting;
- Feedforward is used to compute the output for the given input data (prediction);
- Backpropagation can be called as the reverse mode of the automatic differentiation which is applied to deep neural networks;

Takeaways

- Jacobians will be used decrease computational expenses;
- UAT is not only theorem for approximation by DNNs;
- Experiments show that more neurons in a layer and using SGD as optimizer, approximation gets more accurate;
- There are vast amount of possibility depending on:
    - Your choice of learning rate;
    - Your choice of optimizer;
    - Your choice of loss function;
    - Your choice of regularization technique (?);
    - Choice of model complexity (is less more?).

The End

Thanks for your attention!

Mahammad Namazov