Deep Learning's Notes

Daniele Bertagnoli

2023/2024

Contents

1 Neural Network **3**
- 1.1 Prediction Function . . . 3
  - 1.1.1 Notation . . . 4
  - 1.1.2 Consideration about the dimensionalities . . . 5
- 1.2 Cost Function's Optimization . . . 5
- 1.3 Backpropagation . . . 6
- 1.4 HyperNetworks . . . 6

2 Convolutional Neural Network **7**
- 2.1 CNN Architecture . . . 7
  - 2.1.1 Convolutional Layer . . . 8
  - 2.1.2 Pooling Layer . . . 9
- 2.2 More on Convolution . . . 9
- 2.3 Standard Spatial CNN . . . 9

These notes are about the NYU's Deep Learning course, it can be found on YouTube at the following link.

1 Neural Network

A **Neural Network (NN) is a machine learning model that is capable to capture non-linear relationship between** x **and** θ, where x are the features and θ are the parameters of the model. These models are very powerfull since the non-linearity guarantees an higher level of expressivity.

As for the other classical ML models, **NNs aim to model a function**:

$$h_\theta(x) : \mathbb{R}^d \to \mathbb{R}$$

that outputs a certain prediction y. This **function is created by the network** itself with a mechanism that we will describe in next paragraphs. Once that the function is created, the NN tries to **find the best parameters based on the cost function**,typically the Least Mean Square. The cost function is optimized **using the Gradient Descent method** (can be either the classical one or its stochastic version). To perform Gradient Descent, we must be able to compute the gradient of the cost function with respect to all the parameters, this step is called **backpropagation** and will be also discussed in a dedicated section.

1.1 Prediction Function

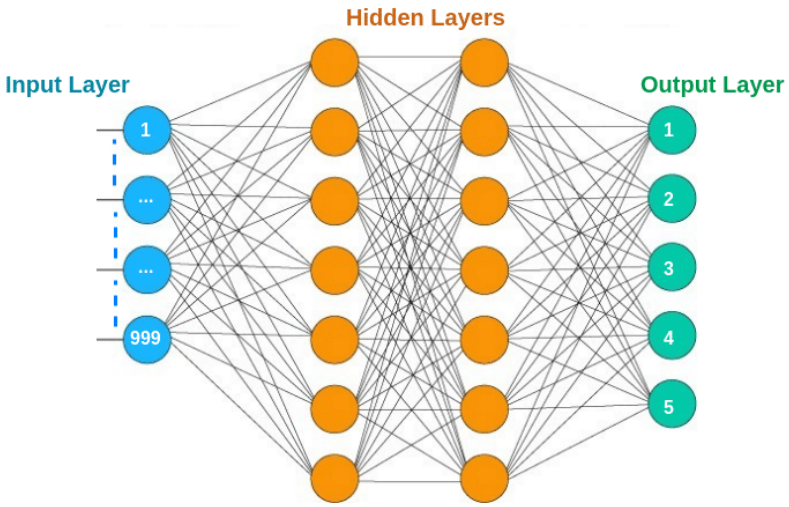
The h_θ function can be expressed using the following form:

$$h_\theta(x) = \sigma(\omega^T x + b)$$

where $x \in \mathbb{R}^d$ is a **feature** of d-dimensionality and $b \in \mathbb{R}$ is called **bias**. The ω is the **weight vector**. The σ function is called **activation function** and it's used to achieve non-linearity, typically it is the $ReLU$ function defined as follows:

$$ReLU(x) = \max(0, x)$$

The **NN is composed by neurons** (also known as perceptrons), each of those implements a h_θ function with different weights. These **neurons are organized in layers such that, each neuron of the layer** i **is connected with all the neurons in the layer** $i + 1$. Therefore, the output of a neuron's function is the input of the next layer.



1.1.1 Notation

To be more compact we will introduce the following notation to express all the functions in the NN.

- a_i^j: The i-th neuron in the j-th layer.
- x_i: i-th feature in the d-dimensional feature vector x.
- w_i^j: The weight vector of the a_i^j neuron. The size of the weight vector is the same as the number of features in input (e.g. for the firs hidden layer is d).
- b_i^j: The bias vector of the a_i^j neuron.
- $z_i^j = {\omega_i^j}^T x + b_i^j$

According with this notation, each neuron's function can be expressed as:

$$a_i^j(x) = ReLU(z_i^j) = ReLU({\omega_i^j}^T x + b_i^j)$$

when $j = 1$ then x coincides with the features vector, otherwise x is represented by a^{j-1}. For convenience, we will express most of the times not the single neuron's function, but the entire layer functions as follows:

$$a^j(x) = ReLU(z^j) = ReLU({W^j}^T x + b^j)$$

$$W^j = \begin{bmatrix} - & w_1^{j\ T} & - \\ - & w_2^{j\ T} & - \\ & \vdots & \\ - & w_m^{j\ T} & - \end{bmatrix}$$

the $\mathbb{R}^{m \times d}$ matrix representing the weights of all the m neurons in the j-th layer.

1.1.2 Consideration about the dimensionalities

Let's briefly discuss the dimensionality of all the new parameters we have introduced in the previus part.

- $z^j \in \mathbb{R}^m$ with $m = \#$ neurons in the j-th layer.
- $x^j \in \mathbb{R}^k$ with $k = \#$ neurons in the $(j-1)$-th layer.
- $b^j \in \mathbb{R}^m$ with $m = \#$ neurons in the j-th layer.
- $W \in \mathbb{R}^{m \times k}$ with m number of the neurons in the j-th layer and k the number of neurons in the $(j-1)$-th layer.

1.2 Cost Function's Optimization

As mentioned in the introduction of the Neural Networks, after that the final function is calculated by the model, we still have to assign real values to the parameters to get a prediction. The cost function that we will discuss is the LMS defined as follows:

$$J^i(\theta) = (y^i - h_\theta(x^i))^2, \text{ considering the } i\text{-th example.}$$

The optimization step consists into find the best θs that minimizes the cost function, this can be done using the Gradient Descent:

$$\theta : \theta - \alpha \nabla_\theta J(\theta)$$

The **Stochastic Gradient Descent** algorithm works as follows:

Algorithm 1 Stochastic Gradient Descent

1: Hyperparameter: learning rate α, number of total iteration n_{iter}.
2: Initialize θ randomly.
3: **for** $i = 1$ to n_{iter} **do**
4: Sample j uniformly from $\{1, \dots, n\}$, and update θ by

$$\theta := \theta - \alpha \nabla_\theta J^{(j)}(\theta)$$

We can also use the **Mini-batch Stochastic Gradient Descent**, this allows to compute multiple bathes in parallel (hence useful for the GPUs):

Algorithm 2 Mini-batch Stochastic Gradient Descent

1: Hyperparameters: learning rate α, batch size B, # iterations n_{iter}.
2: Initialize θ randomly
3: **for** $i = 1$ to n_{iter} **do**
4: Sample B examples $j_1, \ldots, j_B$ (without replacement) uniformly from $\{1, \ldots, n\}$, and update θ by

$$\theta := \theta - \frac{\alpha}{B} \sum_{k=1}^{B} \nabla_\theta J^{(j_k)}(\theta)$$

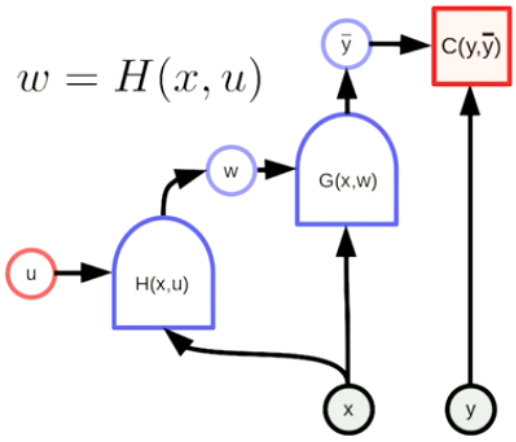
Stanford

1.3 Backpropagation

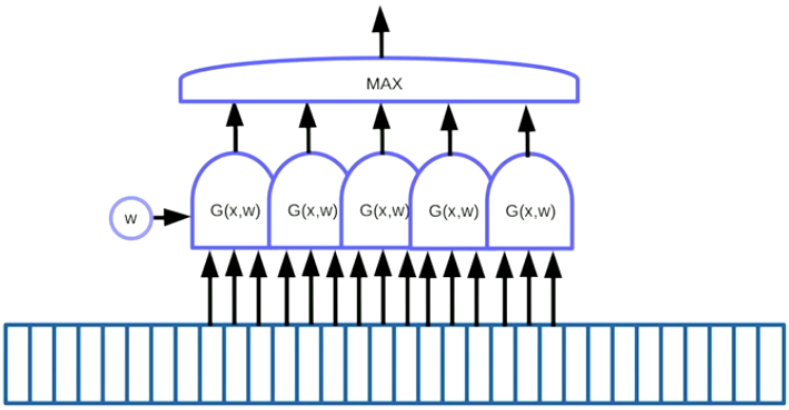
In both the algorithms presented in the optimization step, we need to compute the gradient of the cost function. This involves in computing the derivative of J with respect of θs. The parameters which the J depends on are all the weight matrixes and the bias vectors in the network. Therefore, to solve the gradient of the

1.4 HyperNetworks

In the standard NN architecture, the weights are determined by the network itself during the backpropagation step. However, we can also add more complexity to the structure by using as weight vectors the output of another network or function. In the **HyperNetworks, the weights of a network H are dynamically configured by another network G.**



Another way of thinking these models is to create multiple models that shares the same weights, therefore during the backpropagation step, each of these model will contribute to the weights. This kind of structure is used for example to detect motifs in images, soundwaves, etc...



2 Convolutional Neural Network

A **Convolutional Neural Network (CNN) is a neural network composed of multiple layers that is based upon the convolution operator concept**. The idea behind these networks is to have a **kernel** (can be seen as a sliding window) **which applies a set of filters to obtain higher-level features**. These new features are used in the next layers to perform tasks such as pattern recognition, image classification, etc...

2.1 CNN Architecture

The layers composing the a CNN are the following:

1. **Input Layer**: The input to a CNN is typically an image or a multidimensional array representing the data.

2. **Convolutional Layer**: The convolutional layer is the core building block of a CNN. It consists of a set of filters (also called kernels) that slide across the input image. This process produces a feature map that highlights certain patterns or features in the input data. The filters are learned during the training process to detect various features such as edges, textures, or shapes.

3. **Pooling Layer**: The pooling layer reduces the spatial dimensions (width and height) of the feature maps while retaining important information.

4. **Dense Layer** (Fully-Connected Layer): In the final layers of the CNN, the feature maps are flattened into a one-dimensional vector and connected to a fully connected layer. This layer performs classification tasks by learning weights associated with each feature and combining them to produce an output vector.

5. **Output Layer**: The output layer represents the final prediction or classification of the input data.

The convolutional layers and pooling layers can be repeated more than once.

2.1.1 Convolutional Layer

This layer is responsible for applying filters to the input data. The filter is applied across the input data using the convolution operation, where the filter's weights are multiplied with the input values at each position, and the results are summed up to produce an output feature map. The **general form of a 1-dimensional convolutional operation** is given by:

$$y_i = \sum_j (\omega_j \cdot x_{i-j})$$

- y_i: Output feature at position i
- x_{i-j}: Input features within the kernel
- ω_j: Weights of the filter

The **output features produced by the convolutional layer can be seen as higher-level features compared to the raw input data**. These features **capture patterns and structures** in the data and are used by subsequent layers (such as pooling and fully connected layers) for tasks like pattern recognition and classification.

The kernel is typically shifted by one pixel at time, however this behaviour can be changed by modifying the **stride** which indicates how much the kernel must be shifted between the two iterations. **For each convolutional layer we have to specify both the number of filters that we want to apply during the convolutional step of and their size**. Another important parameter that must be set is the **padding** used to preserve the spatial dimensions of the input volume when applying convolution operations. It involves adding extra border pixels (or values) around the input data before applying the convolution operation. Typically the padding is a zero-padding, hence the values are filled with zeros.

Each feature then is typically passed through a non-linear activation function such as ReLU.

2.1.2 Pooling Layer

This layer aims to reduce the feature maps' dimensions trying to keep only the relevant and useful features. There many pooling operations that can be performed over the features:

- **Max Pooling**: Keeps the maximum value in a window.
- **Average Pooling**: Calculates the average on all the values in a window.
- **L-p Norm**: Applies the L-p Norm over the window.

These operations helps also to lower in some cases the overfitting and the computational complexity of the task.

2.2 More on Convolution

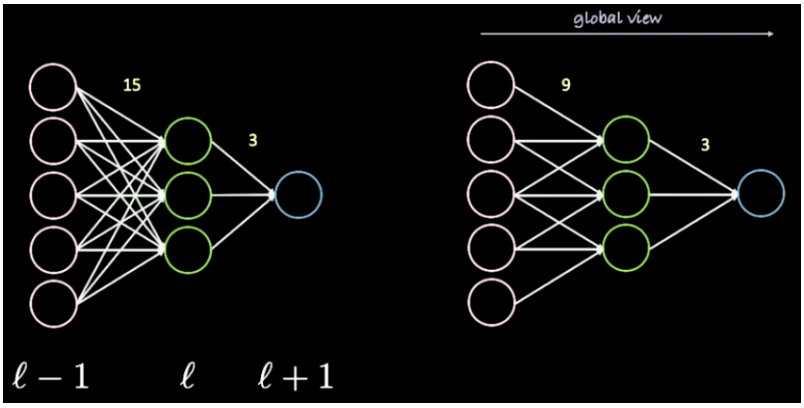
The convolution step in the convolutional layer can be also seen as a sequence of linear operations. A filter, indeed is a matrix that is applied to the input to produce an output:

$$z = Ax$$

- z: is the output vector s.t. $z \in \mathbb{R}^m$
- x: is the input vector s.t. $z \in \mathbb{R}^n$
- A: is the convolution matrix s.t. $z \in \mathbb{R}^{m \times m}$

2.3 Standard Spatial CNN

A **Standard Spatial CNN is a modified CNN designed to model and exploit spatial information and relationships within the input data**. For specific types of data, we can take advantage of the local features to speed up and improve the results of a CNN. In practice, this can involve removing some connections in the neurons of the convolutional layers to save computations. Therefore, the **neurons will not have "global access" to all the input directly, but as they climb the hierarchy of the layers, the final neurons will still have the same amount of information**.



The final architecture can still see the whole input while each single neuron sees a some part of the input.

This is called **sparsity** and can be done if and only if the data that we are using shows **locality**, hence close input data are likely to be realted to each other.

Another important aspect of the SSCNN is the **parameters sharing**, instead of having a number of weights for each linked neuron, a single neuron has a single weight. This technique can be applied if the data are **stationary**, hence the same pattern is repeated multiple times within the dataset. This lead to a faster convergence, better generalization and kernel independece (hence they can be parallelized) and also reduces the number of computations per step.