Deep Learning with PyTorch Workshop

CHAPTER 3: A DEEPER LOOK INTO THE BUILDING BLOCKS OF NEURAL NETWORKS

BY MUHAMMAD RAJABINASAB

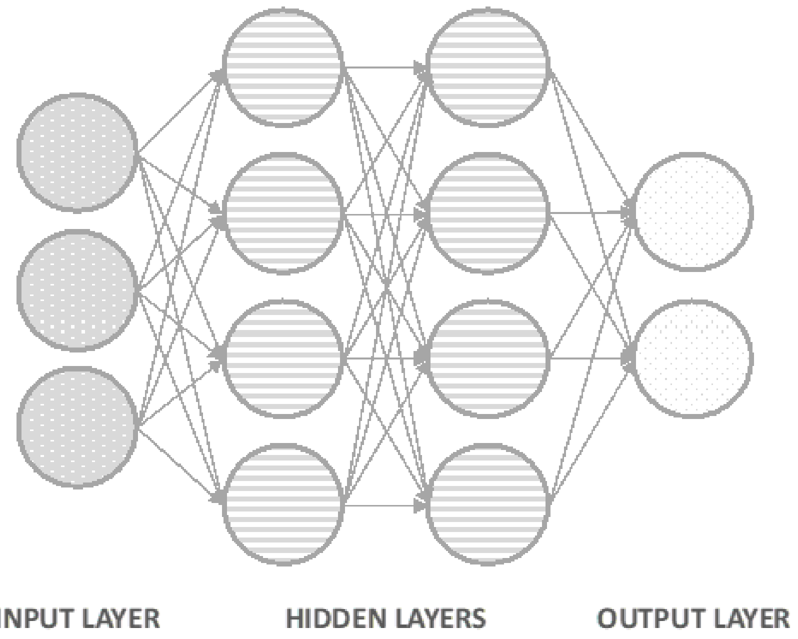
BASED ON THE DEEP LEARNING WITH PYTORCH WORKSHOP

BY HYATT SALEH

Artificial Neural Networks

ANNs are typically composed of three main elements.

1-Input Layer 2-Hidden Layer 3-Output Layer



Introduction to Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are mostly used in the field of computer vision, where, in recent decades, machines have achieved levels of accuracy that surpass human ability.

CNNs create models that use subgroups of neurons to recognize different aspects of an image. These groups should be able to communicate with each other so that, together, they can form the complete image.

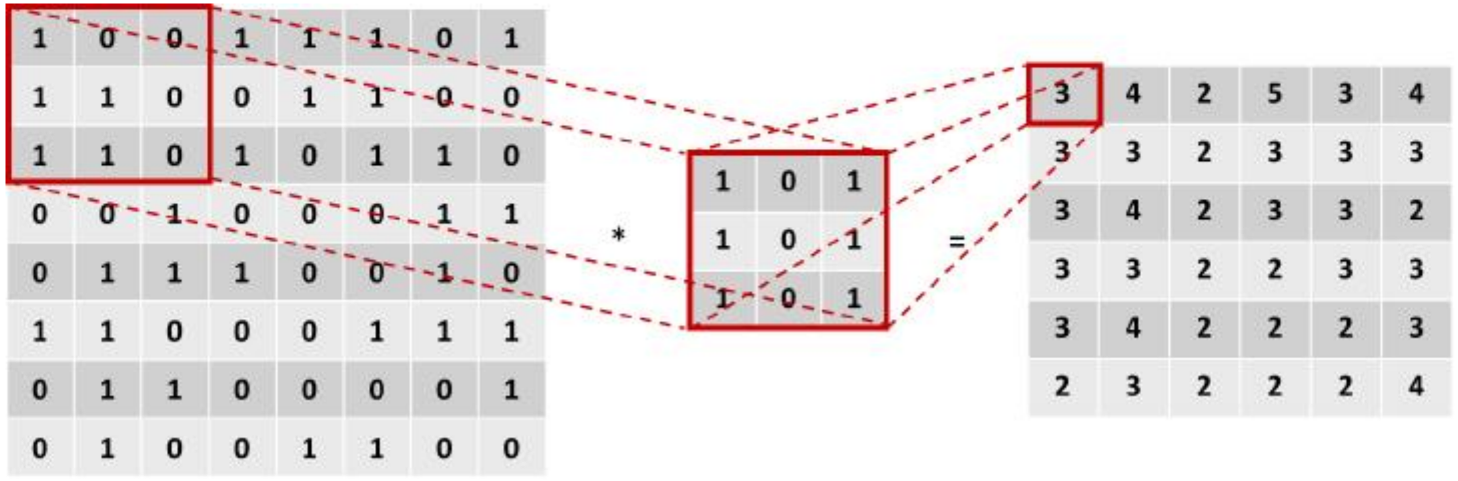
Considering this, the layers in the architecture of a CNN divide their recognition tasks. The first layers focus on trivial patterns, while the layers at the end of the network use that information to uncover more complex patterns.

Convolutional Layers

In these layers, a convolutional computation occurs between an image (represented as a matrix of pixels) and a filter. This computation produces a feature map as output that ultimately serves as input for the next layer.

The computation takes a subsection of the image matrix of the same shape of the filter and performs a multiplication of the values. Then, the sum of the product is set as the output for that section of the image, as shown in the following diagram:

Convolutional Layers



1	0	0	1	1	1	0	1
1	1	0	0	1	1	0	0
1	1	0	1	0	1	1	0
0	0	1	0	0	0	1	1
0	1	1	1	0	0	1	0
1	1	0	0	0	1	1	1
0	1	1	0	0	0	0	1
0	1	0	0	1	1	0	0

*

1	0	1
1	0	1
1	0	1

=

3	4	2	5	3	4
3	3	2	3	3	3
3	4	2	3	3	2
3	3	2	2	3	3
3	4	2	2	2	3
2	3	2	2	2	4

$$1 * 1 + 1 * 1 + 1 * 1 + 0 * 0 + 1 * 0 + 1 * 0 + 0 * 1 + 0 * 1 + 0 * 1 = 3$$

Convolutional Layers

One important notion of convolutional layers is that they are invariant in such a way that each filter will have a specific function, which does not vary during the training process. For instance, a filter in charge of detecting ears will only specialize in that function throughout the training process.

Moreover, a CNN will typically have several convolutional layers, considering that each of them will focus on identifying a particular feature or set of features of the image, depending on the filters that are used. Commonly, there is one pooling layer between two convolutional layers.

Pooling Layers

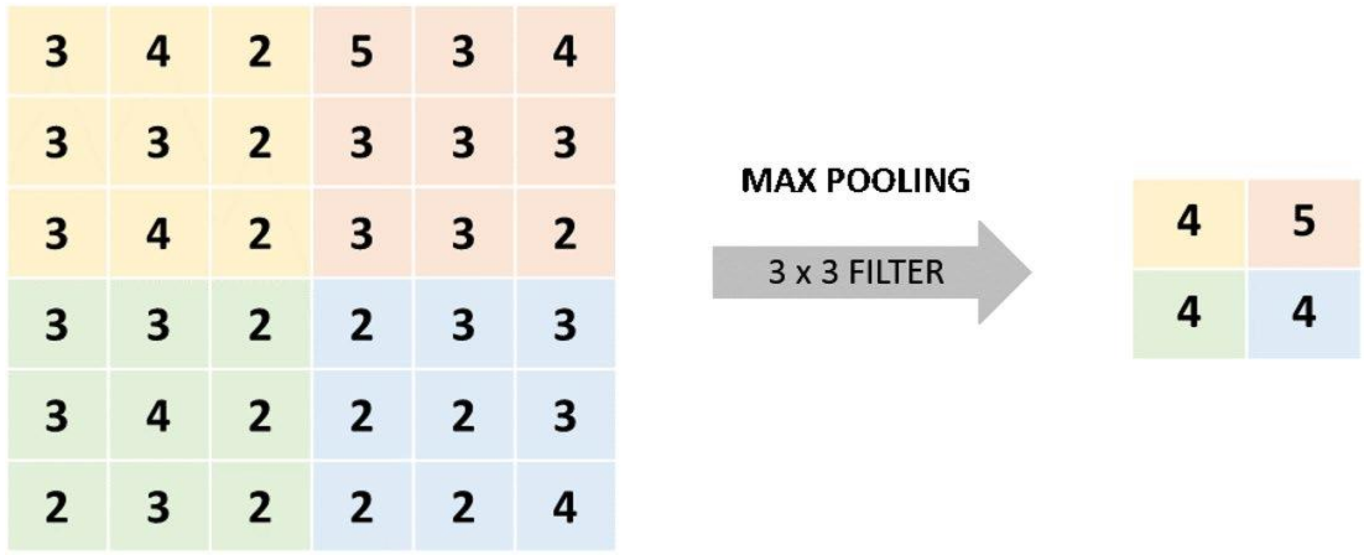
Although convolutional layers are capable of extracting relevant features from images, their results can become enormous when analyzing complex geometrical shapes, which would make the training process impossible in terms of computational power, hence the invention of pooling layers.

These layers not only accomplish the goal of reducing the output of the convolutional layers, but also achieve the removal of any noise that's present in the features that have been extracted, which ultimately helps to increase the accuracy of the model.

There are two main types of pooling layers that can be applied, and the idea behind them is to detect the areas that express a stronger influence in the image so that the other areas can be overlooked.

Max Pooling

This operation consists of taking a subsection of the matrix of a given size and taking the maximum number in that subsection as the output of the max pooling operation:

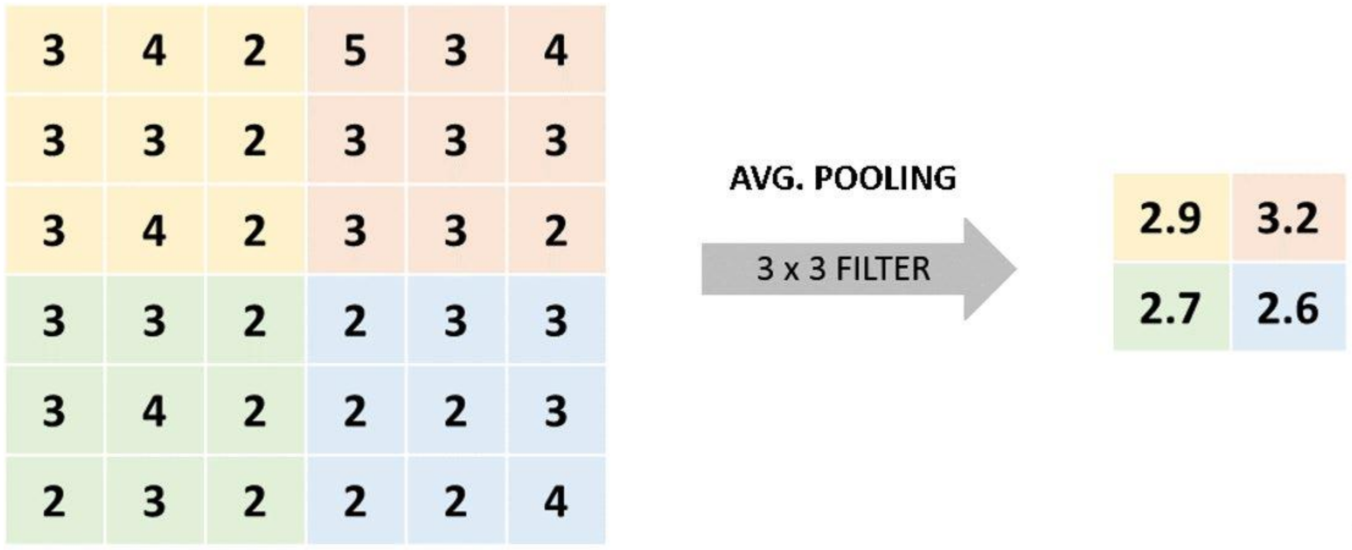


3	4	2	5	3	4
3	3	2	3	3	3
3	4	2	3	3	2
3	3	2	2	3	3
3	4	2	2	2	3
2	3	2	2	2	4

4	5
4	4

Average Pooling

Similarly, the average pooling operation takes subsections of the matrix and takes the number that meets the rule as output, which, in this case, is the average of all the numbers in the subsection in question:



3	4	2	5	3	4
3	3	2	3	3	3
3	4	2	3	3	2
3	3	2	2	3	3
3	4	2	2	2	3
2	3	2	2	2	4

AVG. POOLING

3 x 3 FILTER

2.9	3.2
2.7	2.6

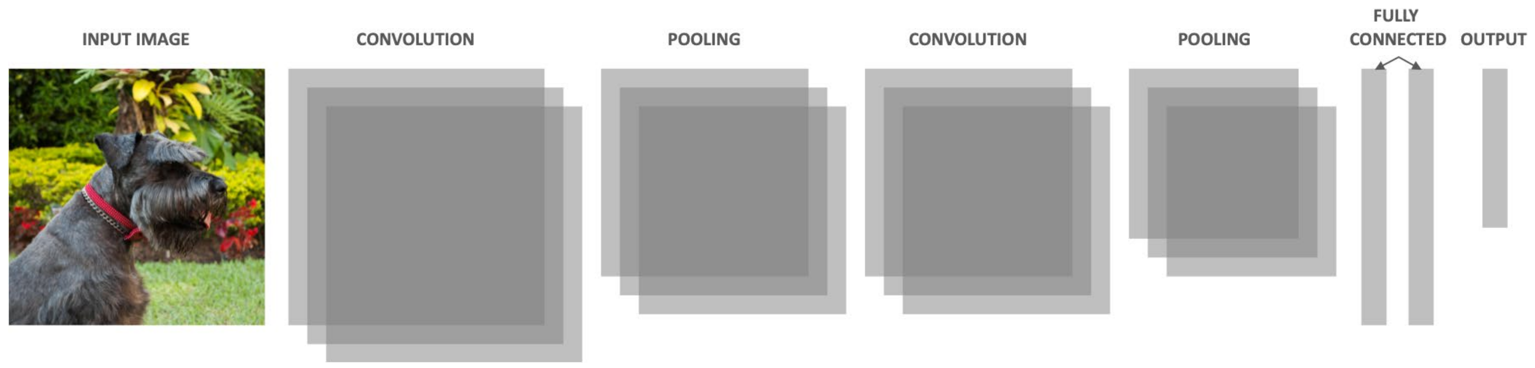
Fully Connected Layers

Finally, considering that the network would be of no use if it was only capable of detecting a set of features without having the capability of classifying them into a class label, fully connected layers are used at the end of CNNs to take the features that were detected by the previous layer (known as the feature map) and output the probability of that group of features belonging to a class label, which is used to make the final prediction.

Like ANNs, fully connected layers use perceptrons to calculate an output based on a given input. Moreover, it is crucial to mention that CNNs typically have more than one fully connected layer at the end of the architecture.

Introduction to Convolutional Neural Networks

By combining all of these concepts, the conventional architecture of CNNs is obtained. There can be as many layers of each type as desired, and each convolutional layer can have as many filters as desired (each for a particular task). Additionally, the pooling layer should have the same number of filters as the previous convolutional layer, as shown in the following image:



Introduction to Recurrent Neural Networks

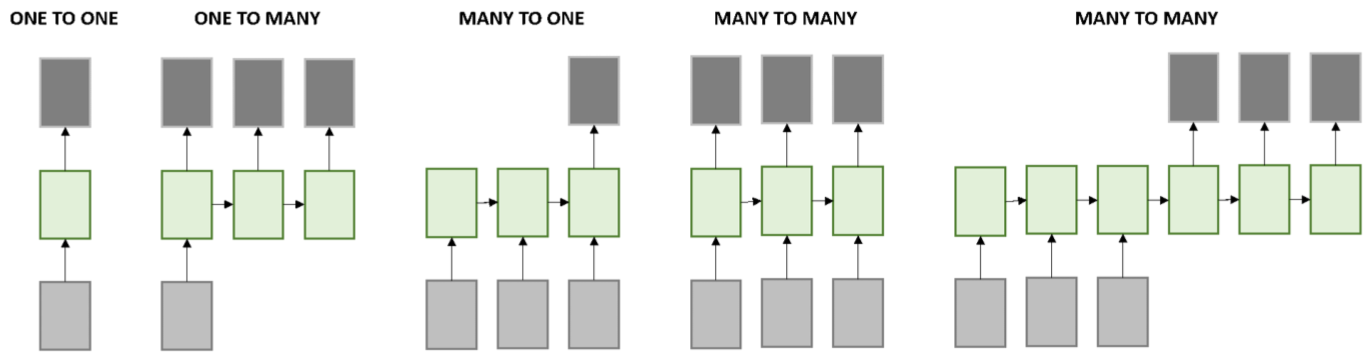
taking into account the fact that neural networks aim to optimize several processes that are traditionally done by humans, it is crucial to think of a network that's able to consider a sequence of inputs and outputs, hence the creation of recurrent neural networks (RNNs). They are a robust type of neural network that allow solutions to be found for complex data problems through the use of internal memory.

Simply put, these networks contain loops in them that allow for the information to remain in their memory for longer periods, even when a subsequent set of information is being processed. This means that a perceptron in an RNN not only passes over the output to the following perceptron, but it also retains a bit of information to itself, which can be useful for analyzing the next bit of information. This memory-keeping capability allows them to be very accurate in predicting what’s coming next.

Introduction to Recurrent Neural Networks

The learning process of an RNN, similar to other networks, tries to map the relationship between an input (x) and an output (y), with the difference being that these models also take into consideration the entire or partial history of previous inputs.

RNNs allow sequences of data to be processed in the form of a sequence of inputs, a sequence of outputs, or even both at the same time, as shown in the following diagram:



Data Preparation

The first step in the development of any deep learning model – after gathering the data, of course – should be preparation of the data. This is crucial if we wish to understand the data at hand to outline the scope of the project correctly.

Many data scientists fail to do so, which results in models that perform poorly, and even models that are useless as they do not answer the data problem to begin with.

This task mainly consists of performing exploratory data analysis (EDA) to understand the data available, as well as to detect potential issues that may affect the development of the model.

Data Preparation

The process of preparing the data can be divided into three main tasks:

1. Understanding the data and dealing with any potential issues
2. Rescaling the features to make sure no bias is introduced by mistake
3. Splitting the data to be able to measure performance accurately

The EDA process is useful as it helps the developer uncover information that's crucialto the definition of the course of action:

Quantity of Data   The Target Feature   Noisy Data/Outliers

Missing values   Qualitative Features

Exercise 2.02: Dealing with Messy Data

In this exercise, we will use pandas, which is a popular Python package, to explore the data at hand and learn how to detect missing values, outliers, and qualitative values.

Data Rescaling

Although data does not need to be rescaled to be fed to an algorithm for training, it is an important step if you wish to improve a model's accuracy. This is basically because having different scales for each feature may result in the model assuming that a given feature is more important than others as it has higher numerical values.

Take, for instance, two features, one measuring the number of children a person has and another stating the age of the person. Even though the age feature may have higher numerical values, in a study for recommending schools, the number of children feature may be more important.

Data Rescaling

Normalization: This consists of rescaling the values so that all the values of all the features are between zero and one. This is done using the following equation:

$$x_{norm} = \frac{x_i - min(x)}{max(x) - min(x)}$$

Standardization: In contrast, this rescaling methodology converts all the values so that their mean is 0 and their standard deviation is equal to 1. This is done using the following equation:

$$x_{standardization} = \frac{x_i - mean(x)}{std(x)}$$

Exercise 2.03: Rescaling Data

In this exercise, we will rescale the data from the previous exercise.

Splitting Data

The purpose of splitting the dataset into three subsets is so that the model can be trained, fine-tuned, and measured appropriately, without the introduction of bias. We have three sets:

- Training Set
- Validation Set
- Testing Set

Exercise 2.04: Splitting a Dataset

In this exercise, we will rescale the data from the previous exercise.

In this exercise, we will split the dataset from the previous exercise into three subsets. For the purpose of learning, we will explore two different approaches. First, the dataset will be split using indexing. Next, scikit-learn's train_test_split() function will be used for the same purpose, thereby achieving the same result with both approaches.

Disadvantages of Failing to Prepare Your Data

Although the process of preparing the dataset is time-consuming and may be tiring when dealing with large datasets, the disadvantages of failing to do so are even more inconvenient:

- Longer training times
- Introduction of bias
- Avoid generalization

Activity 2.01: Performing Data Preparation

In this activity, we will prepare a dataset containing a list of songs, each with several attributes that help determine the year they were released. This data preparation step is crucial for the next activity in this chapter. Let's look at the following scenario.

You work at a music record company and your boss wants to uncover the details that characterize records from different time periods, which is why they have put together a dataset that contains data on 515,345 records, with release years ranging from 1922 to 2011. They have tasked you with preparing the dataset so that it is ready to be fed to a neural network.

Building a Deep Neural Network

Building a neural network, in general terms, can be achieved either on a very simple level using libraries such as scikit-learn (not suitable for deep learning), which perform all the math for you without much flexibility, or on a very complex level by coding every single step of the training process from scratch, or by using a more robust framework, which allows great flexibility.

PyTorch was built considering the input of many developers in the field and has the advantage of allowing both approximations in the same place. As we mentioned previously, it has a neural network module that was built to allow easy predefined implementations of simple architectures using the sequential container, while at the same time allowing for the creation of custom modules that introduce flexibility to the process of building very complex architectures.

Building a Deep Neural Network

In this section, we will discuss the use of the sequential container for developing deep neural networks in order to demystify their complexity. Nevertheless, in later sections, we will move on and explore more complex and abstract applications, which can also be achieved with very little effort.

As we mentioned previously, the sequential container is a module that was built to contain sequences of modules that follow an order. Each of the modules it contains will apply some computation to a given input to arrive at an outcome.

Building a Deep Neural Network

Some of the most popular modules (layers) that can be used inside the sequential container to develop regular classification models are explained here:

• Linear layer: This applies a linear transformation to the input data while keeping internal tensors to hold the weights and biases. It receives the size of the input sample (the number of features of the dataset or the number of outputs from the previous layer), the size of the output sample (the number of units in the current layer, which will be the number of outputs), and whether to use tensor of biases during the training process (which is set to True by default) as arguments.

Building a Deep Neural Network

• Activation functions: They receive the output from the linear layer as input in order to break the linearity. There are several activation functions, as explained previously, that can be added to the sequential container. The most commonly used ones are explained here:

ReLU: This applies the rectified linear unit function to the tensor containing the input data. The only argument it takes in is whether the operation should be done in-situ, which is set to False by default.

$$RELU(x) = \begin{cases} 0 \ if \ x < 0 \\ x \ if \ x >= 0 \end{cases}$$

Building a Deep Neural Network

Tanh: This applies the element-wise tanh function to the tensor containing the input data. It does not take any arguments.

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

Sigmoid: This applies the previously explained sigmoid function to the tensor containing the input data. It does not take any arguments.

$$\text{Sigmoid } S(x) = \frac{1}{1+e^{-x}}$$

Building a Deep Neural Network

Softmax: This applies the softmax function to an n-dimensional tensor containing the input data. The output is rescaled so that the elements of the tensor lie in a range between zero and one, and sum to one. It takes the dimension along which the softmax function should be computed as an argument.

$$\text{Softmax}\ \sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Building a Deep Neural Network

Dropout layer: This module randomly zeroes some of the elements of the nput tensor, according to a set probability. It takes the probability to use for the random selection, as well as whether the operation should be done in-situ, which s set to False by default, as arguments. This technique is commonly used for dealing with overfitted models, which will be explained in more detail later.

Normalization layer: There are different methodologies that can be used to add a normalization layer in the sequential container. Some of them include BatchNorm1d, BatchNorm2d, and BatchNorm3d. The idea behind this is to normalize the output from the previous layer, which ultimately achieves similar accuracy levels at lower training times.

Exercise 2.05: Building a Deep Neural Network Using PyTorch

In this exercise, we will use the PyTorch library to define the architecture of a deep neural network of four layers, which will then be trained with the dataset we prepared in the previous exercises.

Activity 2.02: Developing a Deep Learning Solution for a Regression Problem

In this activity, we will create and train a neural network to solve the regression problem we mentioned in the previous activity. Let's look at the scenario.

You continue to work at the music record company and, after seeing the great job you did preparing the dataset, your boss has trusted you with the task of defining the network's architecture, as well as training it with the prepared dataset.

Summary

In this chapter, we discussed the building blocks of neural networks.

The main architecture of the three main types of neural networks was discussed: the artificial neural network, the convolutional neural network, and the recurrent neural network. The first is used to solve traditional classification or regression problems, the second one is widely popular for its capacity to solve computer vision problems (for instance, image classification), and the third one is capable of processing data in sequence, which is useful for tasks such as language translation.

We then discussed several other properties of neural networks such as activation functions and dropout layers.

Finally we built deep neural networks using PyTorch.

Thanks For Your Attention!

Feel free to ask any questions

☺