Machine Learning Diploma

DEEP LEARNING 3 :DEEP NEURAL NETWORK

Agenda

- TensorFlow
- Underfitting and Overfitting
- Batch Normalization

TensorFlow Framework

TensorFlow

- Created by the Google Brain team and initially released to the public in 2015, TensorFlow is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning models and algorithms (neural networks) and makes them useful by way of common programmatic metaphors. It uses Python or JavaScript to provide a convenient front-end API for building applications, while executing those applications in high-performance C++.
- TensorFlow allows developers to create a graph of computations to perform. Each node in the graph represents a mathematical operation, and each connection represents data. Hence, instead of dealing with low details like figuring out proper ways to hitch the output of one function to the input of another, the developer can focus on the overall logic of the application.
- TensorFlow is, at present, the most popular software library. There are several real-world applications of deep learning that make TensorFlow popular. Being an Open-Source library for deep learning and machine learning, TensorFlow plays a role in text-based applications, image recognition, voice search, and many more. DeepFace, Facebook’s image recognition system, uses TensorFlow for image recognition. It is used by Apple’s Siri for voice recognition. Every Google app has made good use of TensorFlow to improve your experience.

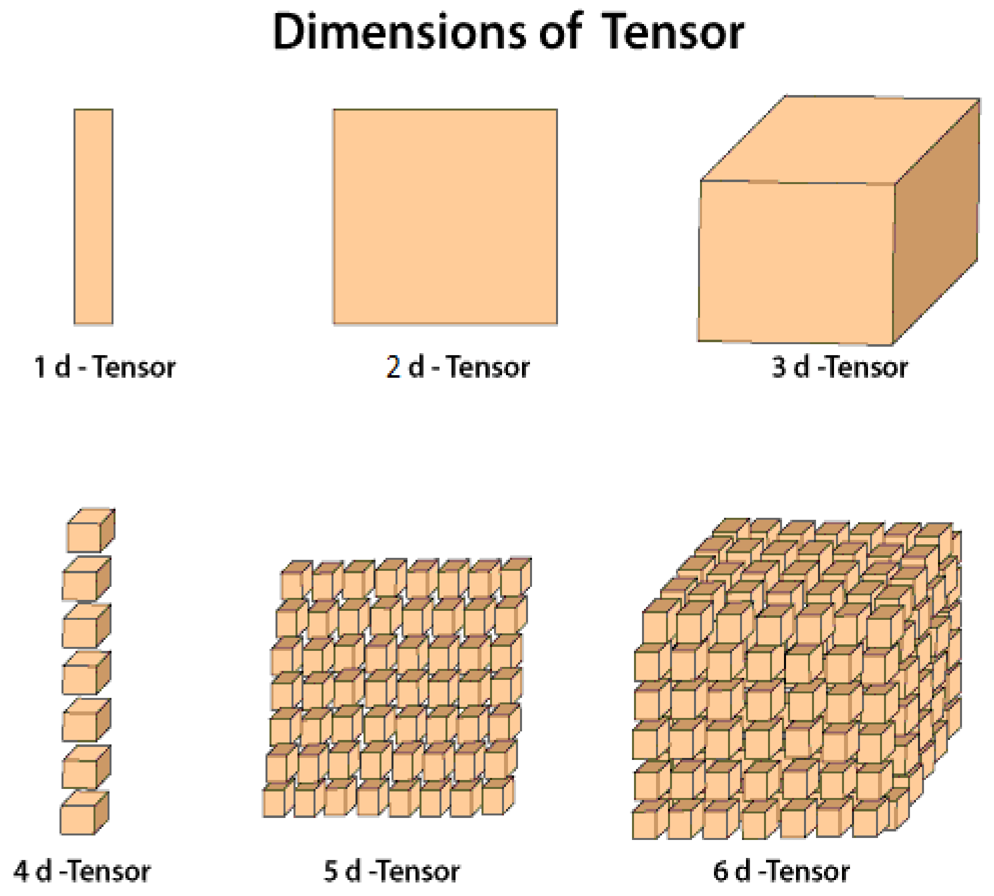
TensorFlow

- **What Is Tensor?**
- All the computations associated with TensorFlow involve the use of tensors.
- A tensor is a vector/matrix of n-dimensions representing types of data. Values in a tensor hold identical data types with a known shape, and this shape is the dimensionality of the matrix.

  A vector is a one-dimensional tensor.

  A matrix is a two-dimensional tensor.

  A scalar is a zero-dimensional tensor.



TensorFlow 1.0 Vs TensorFlow 2.0

- The key differences are:
- **Eager Execution**: In TensorFlow 1.x. The writing of code was divided into two parts: building the computational graph and later creating a session to execute it. this was quite cumbersome, especially if in the big model that you have designed, a small error existed somewhere in the beginning. TensorFlow2.0 Eager Execution is implemented by default, i.e. you no longer need to create a session to run the computational graph, you can see the result of your code directly without the need of creating Session.
- **The Data pipeline simplified**: TensorFlow2.0 has a separate module TensorFlow DataSets that can be used to operate with the model in more elegant way. Not only it has a large range of existing datasets, making your job of experimenting with a new architecture easier – it also has well defined way to add your data to it.
- In TensorFlow 1.x for building a model we would first need to declare placeholders. These were the dummy variables which will later (in the session) used to feed data to the model. There were many built-in APIs for building the layers like tf.contrib, tf.layers and tf.keras, one could also build layers by defining the actual mathematical operations.
- TensorFlow 2.0 you can build your model defining your own mathematical operations, as before you can use math module (tf.math) and linear algebra (tf.linalg) module. However, you can take advantage of the high level Keras API and tf.layers module. ***The important part is we do not need to define placeholders any more.***

TensorFlow 1.0 Vs TensorFlow 2.0

- **Model Building and deploying made easy:** With TensorFlow2.0 providing high level TensorFlow Keras API, the user has a greater flexibility in creating the model. One can define model using Keras functional or sequential API. The TensorFlow Estimator API allows one to run model on a local host or on a distributed multi-server environment without changing your model. Computational graphs are powerful in terms of performance, in TensorFlow 2.0 you can use the decorator **tf.function** so that the following function block is run as a single graph. This is done via the powerful Autograph feature of TensorFlow 2.0. This allows users to optimize the function and increase portability. And the best part you can write the function using natural Python syntax. Effectively, you can use the decorator tf.function to turn plain Python code into graph. While the decorator @tf.function applies to the function block immediately following it, any functions ***called*** by it will be executed in graph mode as well. Thus, in TensorFlow 2.0, users should refactor their code into smaller functions which are called as needed. In general, it’s not necessary to decorate each of these smaller functions with tf.function; only use tf.function to decorate high-level computations – for example, one step of training, or the forward pass of your model. (source stack overflow and TF2 documentation)
- To expand this idea, In TensorFlow 1.x we needed to build the computational graph. TensorFlow 2.0 does not build graph by default. However, as every Machine Learning engineer knows, graphs are good for speed. TensorFlow 2.0 provides the user to create a callable graph using a python function @tf.function. The tf.function() will create a separate graph for every unique set of input shapes and datatypes.

TensorFlow 1.0 Vs TensorFlow 2.0

- **Ease of use:** Many old libraries (example tf.contrib) were removed, and some consolidated. For example, in TensorFlow1.x the model could be made using Contrib, layers, Keras or estimators, so many options for the same task confused many new users. TensorFlow 2.0 promotes TensorFlow Keras for model experimentation and Estimators for scaled serving, and the two APIs are very convenient to use.

How Install TensorFlow?

- Assuming you have a setup, TensorFlow can be installed directly via pip:

```
pip3 install --upgrade tensorflow
```

or

```
pip install tensorflow
```

- If you need GPU support, you will have to install by tensorflow-gpu tensorflow .
- To test installation ,simply run :

```
import tensorflow
print(tensorflow.__version__)
```

Go to Jupyter Notebook

Tasks:

1. Implement linear regression using tensorflow

2. Implement logistic regression using tensorflow (assignment)

Overfitting and Underfitting

Underfitting and Overfitting

- When we talk about the Machine Learning model, we actually talk about how well it performs and its accuracy which is known as prediction errors. Let us consider that we are designing a machine learning model. A model is said to be a good machine learning model if it generalizes any new input data from the problem domain in a proper way. This helps us to make predictions about future data, that the data model has never seen. Now, suppose we want to check how well our machine learning model learns and generalizes to the new data. For that, we have overfitting and underfitting, which are majorly responsible for the poor performances of the machine learning algorithms.
- Before diving further let's understand two important terms:

✓ **Bias:** Assumptions made by a model to make a function easier to learn. It is actually the error rate of the training data. When the error rate has a high value, we call it High Bias and when the error rate has a low value, we call it low Bias.

✓ **Variance:** The difference between the error rate of training data and testing data is called variance. If the difference is high then it's called high variance and when the difference of errors is low then it's called low variance. Usually, we want to make a low variance for generalized our model.

Underfitting and Overfitting

- **Underfitting:** A statistical model or a machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data, i.e., it only performs well on training data but performs poorly on testing data. *(It's just like trying to fit undersized pants!)* Underfitting destroys the accuracy of our machine learning model. Its occurrence simply means that our model or the algorithm does not fit the data well enough. It usually happens when we have fewer data to build an accurate model and also when we try to build a linear model with fewer non-linear data. In such cases, the rules of the machine learning model are too easy and flexible to be applied to such minimal data and therefore the model will probably make a lot of wrong predictions. Underfitting can be avoided by using more data and also reducing the features by feature selection.
- In a nutshell, Underfitting refers to a model that can neither performs well on the training data nor generalize to new data.
- High training error and high testing error.

Underfitting and Overfitting

- **Reasons for Underfitting:**
  a. High bias and low variance
  b. The size of the training dataset used is not enough.
  c. The model is too simple.
  d. Training data is not cleaned and also contains noise in it.
- **Techniques to reduce underfitting:**
  a. Increase model complexity
  b. Increase the number of features, performing feature engineering
  c. Remove noise from the data.
  d. Increase the number of epochs or increase the duration of training to get better results.

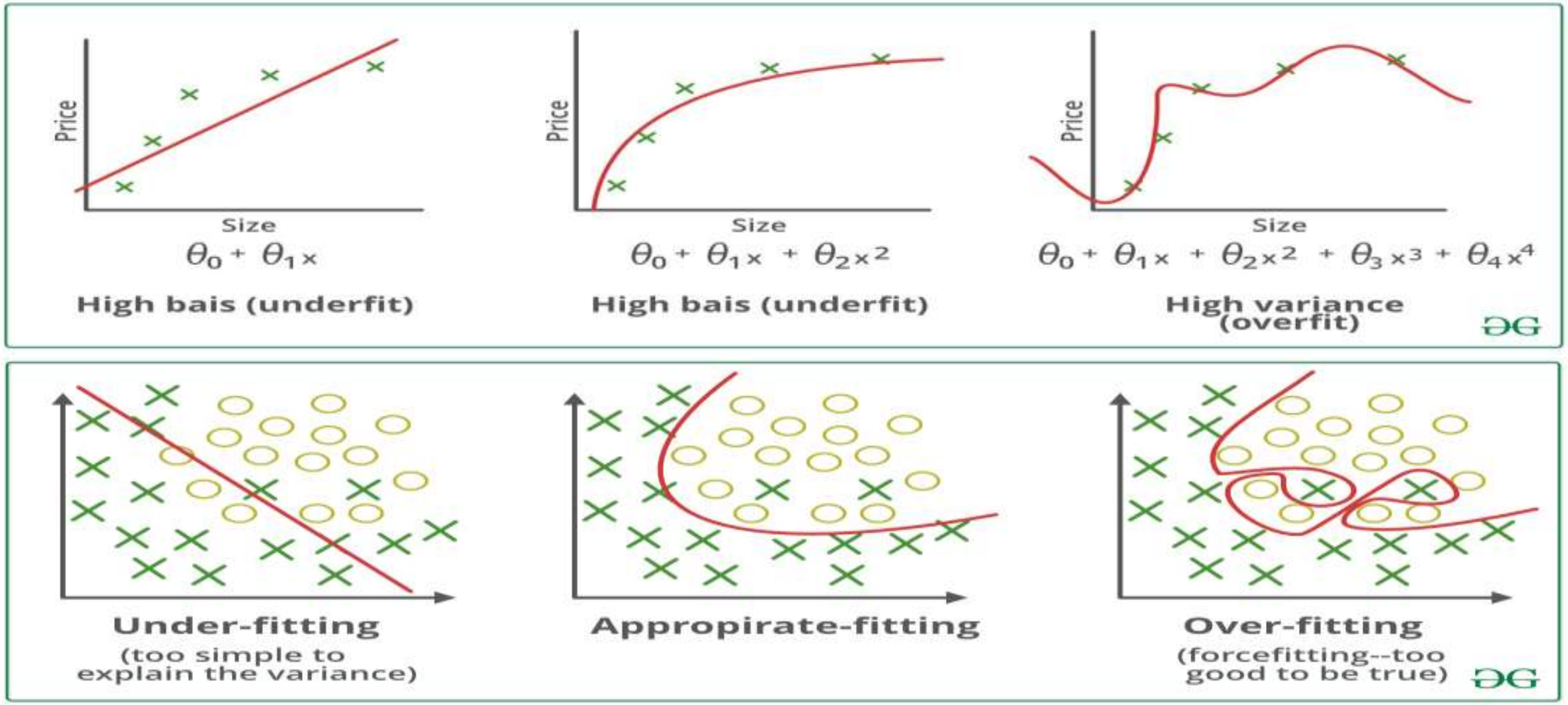
Underfitting and Overfitting

- **Overfitting:** A statistical model is said to be overfitted when the model does not make accurate predictions on testing data. When a model gets trained with so much data, it starts learning from the noise and inaccurate data entries in our data set. And when testing with test data results in High variance. Then the model does not categorize the data correctly, because of too many details and noise. The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore they can really build unrealistic models. A solution to avoid overfitting is using a linear algorithm if we have linear data or using the parameters like the maximal depth if we are using decision trees.
- In a nutshell, Overfitting is a problem where the evaluation of machine learning algorithms on training data is different from unseen data.
- Low training error and high test error.

Underfitting and Overfitting

- **Reasons for Overfitting are as follows:**
  a. High variance and low bias
  b. The model is too complex
  c. The size of the training data (small)
- **Techniques to reduce overfitting:**
  a. Increase training data.
  b. Reduce model complexity.
  c. Early stopping during the training phase (have an eye over the loss over the training period as soon as loss begins to increase stop training).
  d. Ridge Regularization and Lasso Regularization
  e. Use dropout for neural networks to tackle overfitting.

Underfitting and Overfitting


Price
Size
$\theta_0 + \theta_1 x$
High bais (underfit)
Price
Size
$\theta_0 + \theta_1 x + \theta_2 x^2$
High bais (underfit)
Price
Size
$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$
High variance (overfit)
Under-fitting
(too simple to explain the variance)
Appropirate-fitting
Over-fitting
(forcefitting--too good to be true)

Techniques for Avoid Overfitting

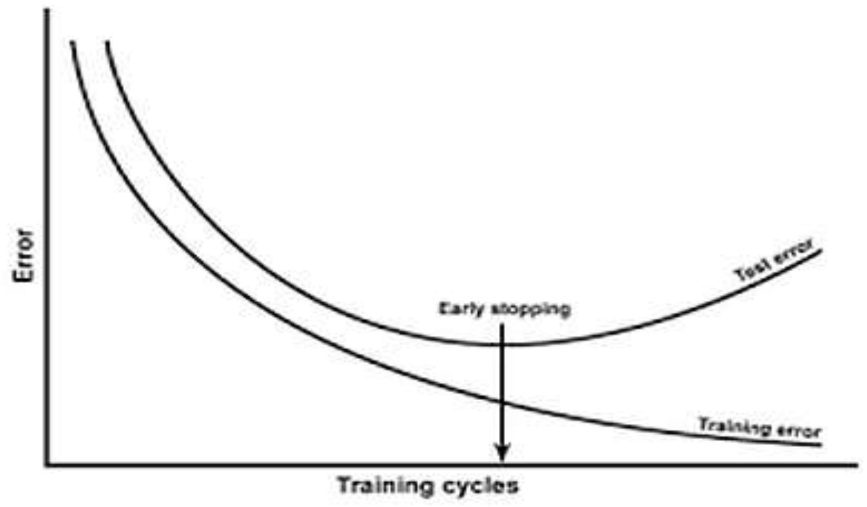
1. Reduce model complexity

- The first step when dealing with overfitting is to decrease the complexity of the model. To decrease the complexity, we can simply remove layers or reduce the number of neurons to make the network smaller. While doing this, it is important to calculate the input and output dimensions of the various layers involved in the neural network. There is no general rule on how much to remove or how large your network should be. But, if your neural network is overfitting, try making it smaller.

Techniques for Avoid Overfitting

2. Early Stopping

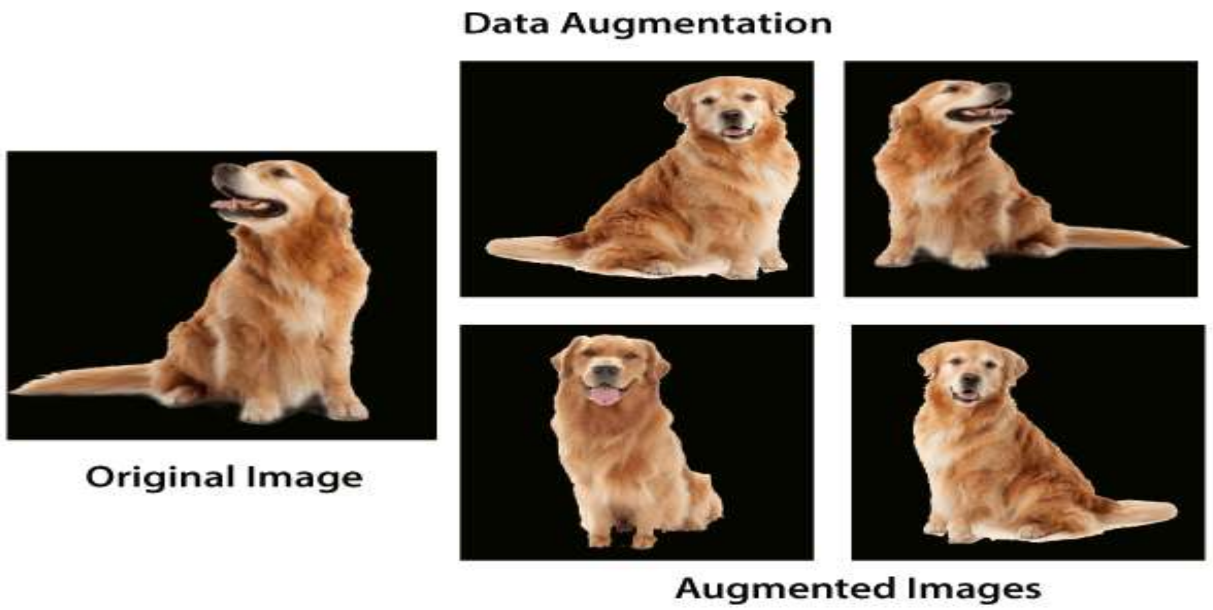
- Early stopping is a form of regularization while training a model with an iterative method, such as gradient descent. Since all the neural networks learn exclusively by using gradient descent, early stopping is a technique applicable to all the problems. This method update the model so as to make it better fit the training data with each iteration. Up to a point, this improves the model's performance on data on the test set. Past that point however, improving the model's fit to the training data leads to increased generalization error. Early stopping rules provide guidance as to how many iterations can be run before the model begins to overfit.



Techniques for Avoid Overfitting

3. Use Data Augmentation

- In the case of neural networks, data augmentation simply means increasing size of the data that is increasing the number of images present in the dataset. Some of the popular image augmentation techniques are flipping, translation, rotation, scaling, changing brightness, adding noise etcetera.



Techniques for Avoid Overfitting

4. Use Regularization

- Regularization is a technique to reduce the complexity of the model. It does so by adding a penalty term to the loss function. The most common techniques are known as L1 and L2 regularization:
- The L1 penalty aims to minimize the absolute value of the weights. This is mathematically shown in the below formula.

$$L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \boxed{\lambda \sum_{i=1}^{n}|\theta_i|}$$

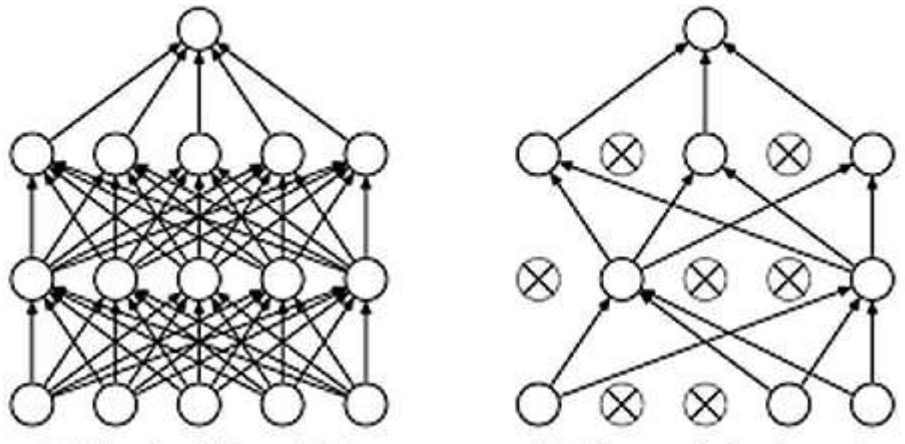
- The L2 penalty aims to minimize the squared magnitude of the weights. This is mathematically shown in the below formula

$$L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \boxed{\lambda \sum_{i=1}^{n}\theta_i^2}$$

Techniques for Avoid Overfitting

5. Use Dropouts

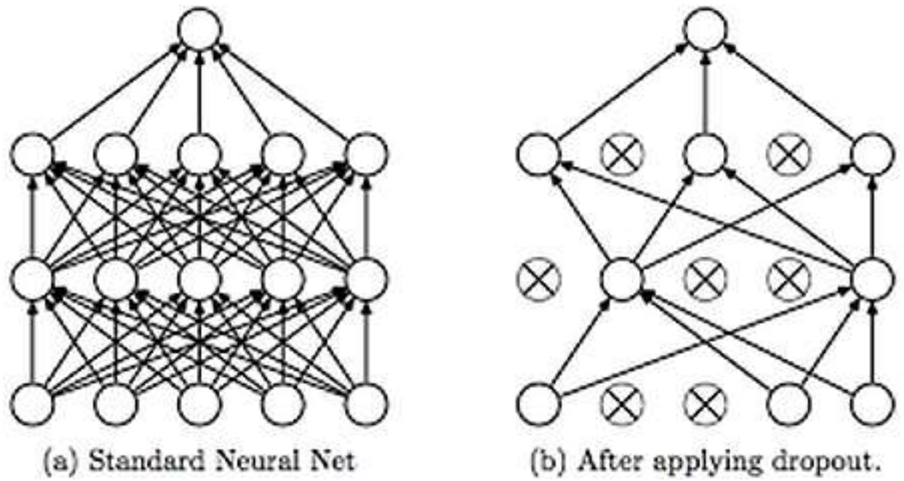
- Dropout is a regularization technique that prevents neural networks from overfitting. Regularization methods like L1 and L2 reduce overfitting by modifying the cost function. Dropout on the other hand, modify the network itself. It randomly drops neurons from the neural network during training in each iteration. When we drop different sets of neurons, it’s equivalent to training different neural networks. The different networks will overfit in different ways, so the net effect of dropout will be to reduce overfitting.



(a) Standard Neural Net (b) After applying dropout.

Techniques for Avoid Overfitting

5. Use Dropouts



- This technique is shown in the above diagram. As we can see, dropouts are used to randomly remove neurons while training of the neural network. This technique has proven to reduce overfitting to a variety of problems involving image classification, image segmentation, word embeddings, semantic matching etcetera.

Batch Normalization

What is Batch Normalization?

- Normalization is a data pre-processing tool used to bring the numerical data to a common scale without distorting its shape.

- Batch normalization, it is a process to make neural networks faster and more stable through adding extra layers in a deep neural network. The new layer performs the standardizing and normalizing operations on the input of a layer coming from a previous layer.

- what is the reason behind the term "Batch" in batch normalization? A typical neural network is trained using a collected set of input data called **batch**. Similarly, the normalizing process in batch normalization takes place in batches, not as a single input.

How does Batch Normalization work?

- It is a two-step process. First, the input is normalized, and later rescaling and offsetting is performed.

- **Normalization of the Input:**

✓ Normalization is the process of transforming the data to have a mean zero and standard deviation one. In this step we have our batch input from layer h, first, we need to calculate the mean of this hidden activation.

$$\mu = \frac{1}{m} \sum h_i$$

- Here, m is the number of neurons at layer h.

How does Batch Normalization work?

- ✓ Once we have meant at our end, the next step is to calculate the standard deviation of the hidden activations.

$$\sigma = \left[\frac{1}{m} \sum (h_i - \mu)^2 \right]^{1/2}$$

- ✓ Further, as we have the mean and the standard deviation ready. We will normalize the hidden activations using these values. For this, we will subtract the mean from each input and divide the whole value with the sum of standard deviation and the smoothing term (*ε*).
- ✓ The smoothing term(*ε*) assures numerical stability within the operation by stopping a division by a zero value.

$$h_{i(norm)} = \frac{(h_i - \mu)}{\sigma + \varepsilon}$$

How does Batch Normalization work?

- Rescaling of Offsetting

✓ In the final operation, the re-scaling and offsetting of the input take place. Here two components of the BN algorithm come into the picture, γ(gamma) and β (beta). These parameters are used for re-scaling (γ) and shifting(β) of the vector containing values from the previous operations.

$$h_i = \gamma \, h_{i(norm)} + \beta$$

✓ These two are learnable parameters, during the training neural network ensures the optimal values of γ and β are used. That will enable the accurate normalization of each batch.

Batch Normalization

- **Benefits of BN:**

✓ It solved entirely the problem of vanishing gradient descent.

✓ Faster training by applying higher learning rate without the fear of activation going very high or low.

✓ Lead to use some of saturating functions like tanh and sigmoid.

✓ Since it solved the problem of vanishing gradient descent, so now we can have deeper networks.

✓ Reduced the sensitivity to the weights initialization

✓ Batch normalization was discovered to have a regularization effect that reduced the need of other regularizes as dropout.

Any Question?

THANK YOU!

AMIT