# Deep Learning Summary

### 

1. **What is Deep Learning?**

Deep learning (DL) is a machine learning technique that uses a biologically inspired neural network architecture to learn features and tasks directly from data, making connections and weighing input for the best results. Traditional machine learning algorithms require domain expertise, and human intervention, and are only capable of what they're designed for. On the other hand, deep learning algorithms can automate tasks that are complicated for AI practitioners and the rest of the world. These algorithms can learn to detect faces from raw data, by detecting lines and edges, and then composing the features together. DL algorithms have existed for quite a long time, but have only recently gained popularity due to the availability of large amounts of data and powerful hardware. Also, with the increasing popularity of open-source software like TensorFlow and Pytorch, deep learning models refer to the training of things called neural networks.

1. **Introduction to Neural Network and how do they learn?**

Neural networks are a set of neurons take in data, recognize patterns in this data, and predict outputs for a new set of similar data.

The learning process is divided into two processes: forward propagation and backward propagation.

FORWARD PROPAGATION:

The inputs are multiplied by the weights, and the sum is then sent to the hidden layer neurons. The neurons are each associated with a numerical value called the bias, which is added to the input sum and passed through a non-linear function called the activation function.

BACKWARD PROPAGATION:

In backpropagation, a neural network evaluates its own performance and sends back information about errors to the hidden layers so that their accuracy level increases.

Example:

Let's suppose we have a data set that tells us if vehicles are cars or trucks. We want to train a neural network to predict vehicles. To start with, we initialize the neural network by giving it random weights and biases. We then move the inputs through the neural network, multiplying the inputs by their weights and adding a bias, and finally using an activation function to generate a guess. In backpropagation, we adjust the initial weights and biases of the neural network by calculating the loss between the model’s predicted and the actual value to better fit the prediction model.

LEARNING ALGORITHM:

* We initialize the network with random values.
* Pass data through it.
* Compare the predicted value to the ground truth.
* Calculate the loss.
* Perform backpropagation to update the weights and biases of the neural network.

1. **TERMS USED IN DEEP LEARNING:**
2. **ACTIVATION FUNCTION:**

The activation function decides whether a neuron can fire or activate and introduces non-linearity into the network.

1. Step function:

It is essentially a step function that is activated when the value is greater than some threshold and not activated otherwise. But the drawback of this function is if you want to classify multiple neurons into classes, and more than one neuron is activated, you would want the network to activate only one neuron and the other should be zero. We can train our models with this activation function, but it is harder to train and converge it this way.

1. Linear Function:

A linear function gives us a range of activations, so we can connect a few neurons together and decide based on the maximum value.

But the problem here is that when gradient descent is calculated the derivative is a constant (x), and this means that the adjustments made to the weights and biases are not dependent on the input (x).

And if all of the layers are linear, the final layer is just a linear function of the input of the first layer, and the entire neural network can be replaced by a single layer.

1. Sigmoid Function:

A “sigmoid” function is a non-linear activation function that outputs an analog activation unlike the step function, and also has a smooth gradient. It has several advantages over a linear activation function but also has some disadvantages.

When the input to the activation function is very large, the “sigmoid” is going to squish that down to a value between zero and one, and the gradient of this function becomes really small which gives rise to vanishing gradient problem.

And when the input to the activation is between x=-2 and x=2 the value of y or output changes very rapidly.

1. Tanh Function:

The “tanh” function is another activation function that is used and it almost have the same properties as the sigmoid have. The changes are that it can output in range [-1,1] unlike sigmoid that outputs in range [0,1].

But it also has the same disadvantages as the “tanh” function also like the vanishing gradient problem. And further more it has much steeper gradient than the sigmoid.

1. ReLU Function:

The rectified linear unit or “ReLU” function is also a non-linear function that can be stacked. However, it is not bounded, so there is a chance of blowing up the activation. But the most important advantage of ReLU function is that is can make the neural network sparser. How and why? Why because if we use analog activation functions like sigmoid or tanh then all of the neuron would be activated which would have our neural network more dense and computationally expensive so by using ReLU activation function we can limit the activation of some neurons as it completely shuts neuron off if the output of neuron is negative making our neural network sparse.

But as some neuron never get activated, they also don’t get updated during the backpropagation process as the gradient of straight line is zero in the negative region and this is called the dying value problem and several neurons just die and do not respond. There are workarounds, including making the horizontal line into a non-horizontal component by adding a slope.

All in all, “ReLU” activation function is less computationally expensive than functions like “tan h” and “sigmoid”, so you should consider using it for everything. However, you can also consider “sigmoid” and “tanh” if you know the nature of the function you're trying to approximate.

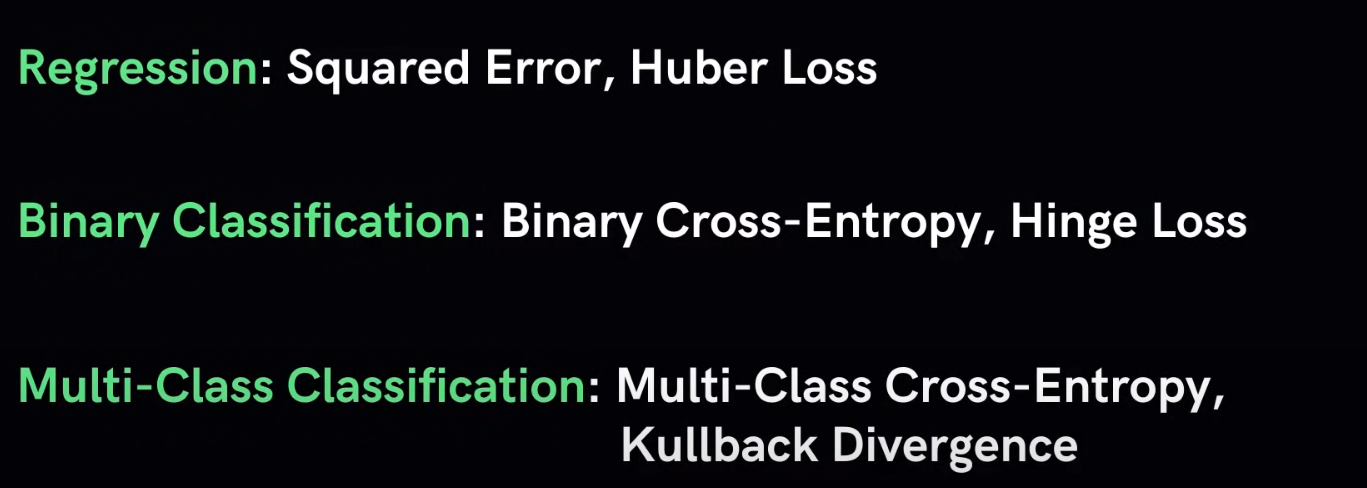
**Why we use non-linear activation functions as opposed to linear ones?**

Non-linear activation functions are polynomials of a degree greater than one, and when graphed, they don't form straight lines rather than more curved ones.

1. **Loss Function:**

In the learning process of neural networks, we started with random weights and biases and compared the predicted output to the expected output. A loss function is used to calculate the deviation.

There are plenty of loss functions out there, and the choice depends on what kind of project you're working on.

****

Loss functions are mathematical ways of measuring how wrong predictions made by the neural network are.

1. **Optimizers:**

Optimizers tie together the loss function and model parameters and adjust the weights and biases in response to the output of the loss function.

In this analogy, you resemble a neural network going down a hill. The loss functions measure whether you're going in the right way or the wrong way.

Grading Descent is an iterative algorithm that travels down the slope of a loss function in steps until it reaches the lowest point or the minimum of the function.

A gradient is a vector of partial derivatives with respect to all independent variables that points in the direction of the steepest increase in a function. When dealing with high dimensional data sets, it's possible to get stuck in local minima so make sure you use the proper learning rate.

Changing our weights too fast by adding or subtracting too much can hinder our ability to minimize the loss function. In math, taking steps that are too large or too small can lead to the algorithm not converging to an optimum.

1. Stochastic Gradient Descent:

Stochastic Gradient Descent is a less computationally expensive implementation of gradient descent because it uses data in batches and also uses momentum to dictate what might happen in the next steps.

1. Adagrad:

Adagrad adapts the learning rate specifically to individual features but has a major issue with the adaptive learning rate getting really small over time. Used normally for sparse datasets.

1. RMSprop:

Professor Jeffrey Hinton developed Rms Prop, a special version of Adagrad, which accumulates gradients in a fixed window instead of accumulating gradient for momentum to solve some of the issues that Adagrad leaves.

1. Adam:

Adam stands for Adaptive Momentum Estimation. It is a neural network optimizer that uses past gradients to calculate the current gradient. It also utilizes the concept of momentum.

1. **Parameters and Hyperparameters:**

Parameters are variables that are internal to the neural network and whose values can be estimated from the data itself. They are often not manually set by the practitioner. Example: weights, bias etc.

A hyperparameter is a configuration of the model that cannot be estimated from data. It is usually found by trial and error. Example: Learning rate, activation function etc.

1. **Epochs, Batches, Batch size & Iterations:**
2. **Epoch:**

When the data is too big to pass to the computer at once, we need to divide it into smaller chunks and update the weights at the end of every step.

One epoch means, we pass the entire dataset through the network once. In a majority of deep learning models, we use more than one epoch, so the neural network can generalize better.

Gradient descent is an iterative process when updating the parameters and backpropagation in a single pass or one epoch is not enough. The number of epochs depends on the size of the data set and the number of batches needed to complete one epoch.

1. **Batch & Batch Size:**

Here we divide large data sets into smaller batches and feed those batches into the neural network.

Batch Size = Total number of training examples in a Batch

1. **Iterations:**

Number of batches needed to complete one epoch.

Alternatively,

Number of batches = Number of iterations for one epoch

Example:

Suppose you have a dataset of 34000 training examples and you divide the dataset into batches of 500. Then to complete 1 epoch, we would need 68 iterations.

In the end in deep learning, you'll often have a bunch of different choices to make, and it's difficult to know what's the right combination to use for your project. Dabble along with the material shown, try various combinations, and see what works for you best.

1. **TYPES OF LEARNING:**
2. Supervised learning:

Supervised machine learning algorithms are designed to learn, and are typically used when you're new to machine learning. In this learning method we train our models on well-labeled data and then take in new unseen inputs and predict the correct label for them based on prior training data. It connects input features to a predicted output by creating a mapping function.

Supervised Learning can be classified into two categories:

1. Classification:

During training, a classification algorithm will be given data points with an assigned category and will assign a value to a class. A binary classification problem is used to determine if an email is a spam or not. The algorithm will use training data and a mapping function to determine whether the email is spam or not.

There are numerous algorithms to solve classification problems, and the best one depends on the data and the situation.

1. Regression:

Regression is a predictive statistical process that attempts to find the important relationship between dependent and independent variables. If we plot data and we can determine that a student's test grade is correlated with how many hours they studied then we can use the line of best fit to predict how well a student would do with five hours of study.

1. Unsupervised Learning:

Unsupervised learning is a branch of machine learning that does not use labeled data and instead focuses on the data's features. It can be used to find hidden patterns in data.

1. Clustering:

Clustering is the process of grouping data points into different clusters. It can be broken down into partitional clustering and hierarchical clustering and is organized as a tree diagram.

1. Association:

It attempts to find relationships between different entities.

All things considered; unsupervised learning finds applications almost everywhere. For example, Airbnb uses unsupervised learning algorithms to recommend stays and experiences that fall under the same group of clusters. Credit card fraud detection is another unsupervised learning algorithm that learns patterns of a user's usage and generates an alarm if the patterns are not matched.

1. Reinforcement Learning:

Reinforcement Learning is a type of machine learning technique that uses rewards and punishments to encourage positive and negative behavior.

When you compare reinforcement learning with unsupervised learning, reinforcement learning aims to maximize the total cumulative reward.

Reinforcement Learning algorithms are goal-oriented algorithms that learns how to attain a complex objective or goal. They are penalized for making the wrong decisions and rewarded for making the right ones.

A reinforcement learning problem involves an agent that learns from its environment by receiving rewards and mapping its actions to its states. It can be best explained through the game of Pac-man. The agent or Pac-man receives a reward for eating food and receives punishment if it gets killed by a ghost.

1. **Regularization:**

A central problem in deep learning is overfitting, where your model performs exceptionally well on training data but not on testing data.

A line of some random slope in intercept is not the best model for a data set and is actually called underfitting. And where the line that fits the data set the best but is actually called overfitting.

Overfitting occurs when a network memorizes patterns only on the training data and doesn't perform well on new data.

**Tackling Overfitting:**

1. **Dropout:**

Dropout is a regularization technique used in deep learning to tackle overfitting. It randomly selects some nodes and removes them along with their incoming and outgoing connections, resulting in a different set of outputs at every iteration.

1. **Dataset Augmentation:**

To make a deep learning model generalize better, we can create new fake data and add it to the training set. A classifier needs to be invariant to a wide variety of transformations, so it can generate new input to output pairs easily.

Dataset augmentation can be very effective in improving generalization, especially for high-dimensional classification problems. You must be careful not to apply a transformation that would change the correct class.

1. **Early Stopping:**

When training large models, we can stop training at the point where the validation set error starts to increase, and thus obtain a model with better validation set error.

1. **Neural Network Architectures:**
2. **Fully Connected Neural Networks**
3. **Convolution Neural Networks**
4. **Recurrent Neural Networks**
5. **Fully Connected Neural Networks:**

A fully connected feed-forward neural network has no cycles or loops in the connections. Each neuron in a neural network contains an activation function that changes the output of a neuron when given its input. There are several types of activation functions, and we use in various layers in a deep neural network based on the problem.

Using everything we've learned so far, we can create a wide variety of neural networks. These networks can solve a wide array of problems. The more neurons we add to a neural network, the more complex the network becomes, and the more computational resources are required to train the network.

All the neural networks we've discussed so far are Feed Forward Neural Networks, which take an input and give an output. But these networks aren't able to model every single problem.

If I show you a picture of a ball and ask you to predict its next position in two or three seconds, you probably won't give me an accurate answer. If I say the word dog, you probably won't understand it either.

If I give you the previous position states of a ball and ask you to predict its future trajectory, you can do this accurately.

Neural networks can't model sequential data, such as a sentence, because they are feed-forward models. This is a major shortcoming of traditional neural networks. The disadvantage of modeling sequences with traditional neural networks is that they don't share parameters across time.

1. **Recurrent Neural Networks:**

This type of neural network shares parameters to look for a given feature everywhere in the sequence. They are RNNs (Recurrent neural networks) which can be used to process sequences of data with variable input lengths, maintain sequence order, and keep track of long-term dependencies. They also share parameters across the sequence so that we don't reload things.

Neural networks can propagate information by using knowledge of their previous states and can therefore model certain areas of data that standalone neural networks can't model. Recurrent Neural Networks remember their past, while basic feed-forward networks remember things they learned during training. Therefore, to train an RNN, we apply the backpropagation through time algorithm to every sequence data point. They predict the next letter a person is likely to type based on the previous letters they've already typed. It uses the letter the user just typed as well as the state of the first hidden neuron to compute the next prediction.

But RNNs have short-term memory issues because it processes more words, it has trouble retaining information from previous steps.

Backpropagation through time which is used with recurrent neural networks gives rise to a problem that the gradient values of the backpropagation through time are exponentially shrinking, which causes small adjustments in the neural network weights. And because of the vanishing gradient, the RNN can't learn the long-range dependencies across time steps, which causes the network to have a short-term memory.

We can combat the short-term memory of an RNN by using two variants of recurrent neural networks: Gated RNNs and Long short-term Memory RNNs. These networks use mechanisms called gates to learn long-term dependencies.

1. **Convolutional Neural Networks:**

A Convolutional Neural Network is a type of deep neural network architecture that was inspired by the visual cortex of the animal brain. It consists of an input layer, an output layer, and several hidden layers. The hidden layers use convolution and pooling functions instead of traditional activation functions.

Convolution is a technique that allows us to extract visual features from a 2d array in small chunks. Each neuron in a convolution layer is responsible for a small cluster of neurons in the preceding layer.

Pooling, also known as sub-sampling or down-sampling, is the next step in a convolutional neural network. It can be performed in two different ways: max pooling and min pooling.

We use multiple filters (also called kernel) in a convolution layer to create a feature matrix, and pool the results to produce a down sample feature matrix. We then add a few fully connected hidden layers to help classify the image.

1. **Step to create a Deep Learning Model:**

Data is at the core of what deep learning is all about. The amount of data you need for a well-performing model should be 10 times the number of parameters in that model. Quantity of data matters, but quality matters more. A model trained on reliable data is more likely to yield useful predictions. There are plenty of resources on the web that offers good data sets for free.

First, we split the data set into subsets. Then, we evaluate the module on the validation set and finally test it on the testing data set.

To develop a model, you must choose certain values for the hyperparameters and biases, and do this with the feedback received from the validation set.

To eliminate skewing, you have to split the data set into three parts. This depends on the number of samples and the model you're trying to train.

For models with few hyperparameters, you can reduce the size of your validation set, but for models with many hyperparameters, you should have a large validation set as well as consider cross-validation. Cross-validation is a method for avoiding overfitting in machine learning models. It involves splitting your data set into two, the train and the test, and iteratively training and validating the model on both sets.

When working with time series data, a frequent technique is to split the data by time. This ensures your validation set mirrors the lag between training and serving.

There are a couple of ways to format data in pre-processing, such as changing the format from database to CSV.

It is quite common in real-world problems to miss some values of our data samples. There are a couple of ways to deal with missing values, but be smart as handling missing data in the wrong way can spell disaster. In real-world datasets, you're going to come across imbalanced data, which is classification data that has skewed class proportions. To avoid biasing a model towards the majority class, you should down-sample and up weight the data. After down-sampling, we add example weights to help the model converge faster and to keep the data set in a similar proportion. This helps the model perform better in real-world situations.

Feature scaling is a crucial step in the pre-processing phase of deep learning algorithms. The most common techniques are normalization and column standardization. Once our data has been prepared, we feed it into our neural network to train. The parameters are adjusted based on the losses incurred.

Your model has successfully trained, now you need to test it against data it has never seen before. After the evaluation process, your model could be optimized further.

There are a few ways to optimize your model, including showing the model the entire data set multiple times, increasing the number of epochs, and adjusting the learning rate. Initial conditions can also play a significant role in determining the outcome of training.

The adjustment of hyperparameters remains an art and depends on the specifics of your data set, model, and training process.

When your model performs well on training data, but terribly on unseen data, it is probably overfitting. Getting more data is usually the best solution, but reducing the model size by reducing the number of learnable parameters is another way.

The second method to addressing overfitting is by applying weight regularization to the model. This is done by adding to the loss function a cost associated with having larger weights.

When working with images, there's always a chance that your model won't perform as well as you'd like it, no matter how much data you have. Data augmentation is a good way of increasing your data set without really increasing it.

Dropout is a technique used in deep learning that randomly drops out units or neurons in the network during the training phase. This reduces overfitting by ignoring these units during a particular forward or backward pass.

-=-END OF VIDEO-=-