Velagapudi Ramakrishna Siddhartha Engineering College

M.Tech Degree Examination, September, 2022

II Semester DataScience

**19ITDS2014B- Deep Learning**

**Max. Marks : 60**

**Answer one question from each Unit**

**UNIT-I**

1.What are the limitations of traditional computer programs? How to tackle such programs with a different kind of approach? 15M

**Scheme:** Limitations of traditional computer program with examples –8M

Machine learning approach - 7M

Traditional computer programs are designed to be very good at two things:

1. performing arithmetic really fast and

2) explicitly following a list of instructions

So, Traditional computer programs can do heavy financial number crunching. But it is difficult to write a programs like automatically read someone’s handwriting .

* *For example, It difficult to write* rules that can differentiate one digit from another



It is also difficult to write programs for Object recognition, Speech comprehension, automated translation etc. using traditional computer programming.

To tackle these classes of problems, we’ll have to use a very different kind of approach called machine learning, which is predicated on this idea of learning from example. In machine learning, instead of teaching a computer a massive list of rules to solve the problem, we give it a model with which it can evaluate examples, and a small set of instructions to modify the model when it makes a mistake. We expect that, over time, a well-suited model would be able to solve the problem extremely accurately.

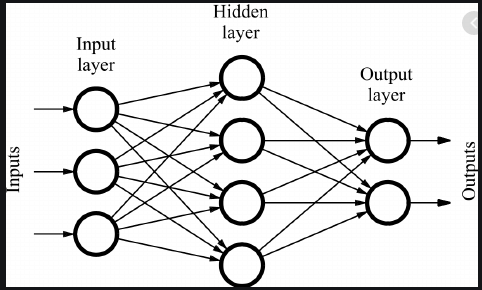
**(OR)**

2. Why single neurons are not expressive to solve complicated learning problems. Discuss the major types of layers that are utilized in feed forward neural networks. 15M

**Scheme:**Discussion on single neuron – 4M

Feed forward neural network with diagram - 11M

Although single neurons are more powerful than linear perceptrons, they’re not nearly expressive enough to solve complicated learning problems. That is why our brain is made of more than one neuron. For example, it is impossible for a single neuron to differentiate handwritten digits. So to tackle much more complicated tasks, we’ll have to take neural networks with multiple layers and multiple neurons.



* Input Layer: The bottom layer of the network pulls in the input data.
* Output Layer: The top layer of neurons (output nodes) computes our final answer.
* Hidden layer: The middle layer of neurons in between input and output layers are called the *hidden layers.*

A neuron takes in some number of inputs, x1, x2, . . . ,xn, each of which is multiplied by a specific weight, w1,w2, . . . ,wn. They are summed together to produce the logit.logit also includes a bias, which is a constant. The logit is then passed through a function f which is called an activation function to produce the output y = f (z). The purpose of an activation function is to add some kind of non-linear property to the function.

**UNIT II**

3. a. Demonstrate with an example of Linear neuron and their limitations. 8M

**Scheme:** Sigmoid description -5M

Limitations -3M

A linear neuron is a neuron that use a linear function in the form of f (z)= ax+b as its activation function.

• The artificial neuron takes in some number of inputs, x1, x2, . . . ,xn, each of which is multiplied

by a specific weight, w1,w2, . . . ,wn.

• . Then we can re-express the outputof the neuron as Where w 0 is the bias term.

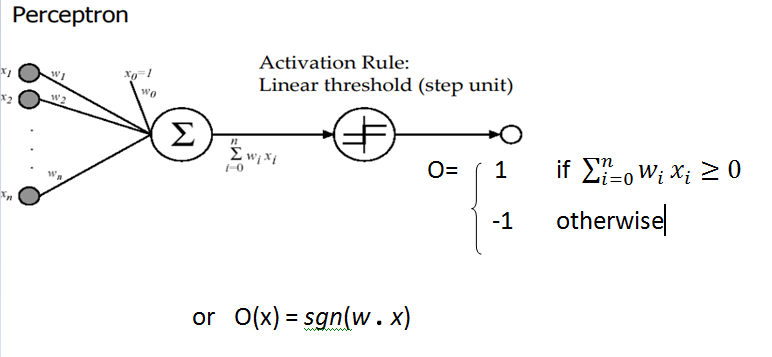
A neural network with perceptron acts as a linear neuron. It uses step function as its activation function.

Linear neurons are easy to compute with, but they run into serious limitations. The following are the disadvantages of linear neurons:

 The function’s derivative is a constant. That means there is constant gradient descent occurring since there is no relation to the value of z.

 A linear model is not really learning as it does not improve upon the error term, which is the whole point of the neural network.

Linear Perceptron



b. Explain how to tackle the problem of training multi layer neural networks. 7M

**Scheme:** Discussion of back propagation - 3M

Derivation for weight update - 4M

To tackle the problem of training multilayer neural networks (instead of just single neurons). To accomplish this task, we’ll use an approach known as back propagation. Each hidden unit can affect many output units. Thus, we’ll have to combine many separate effects on the error in an informative way .

**(OR)**

4. Explain the three major types of neurons that are used in practice and introduce nonlinearities in their computations. 15M

**Scheme:** 3 neurons –Each with explanation, formula and output shape-5M , 3X5=15M

There are three major types of neurons that are used in practice that introduce nonli‐ nearities in their computations. They are sigmoid, tanh and relu neurons

**UNIT III**

5. a. Descibe about sparsity in autoencoders. 8M

**Scheme:** Discussion on interpretability - 5M

Sparisty penalty with formula -3M

Deep models are generally very difficult to interpret because of the nonlinearities and massive numbers of parameters that make up a model. While deep models are generally more accurate, a lack of interpretability often hinders their adoption in highly valuable, but highly risky, applications.

In general, an autoencoder’s representations are dense, and this has implications with respect to how the representation changes as we make coherent modifications to the input. The autoencoder produces a dense representation, that is, the representation of the original image is highly compressed. The activations of the representation combine information from multiple features in ways that are extremely difficult to disentangle. The result is that as we add components or remove components, the output representation changes in unexpected ways. It’s virtually impossible to interpret how and why the representation is generated in the way it is.

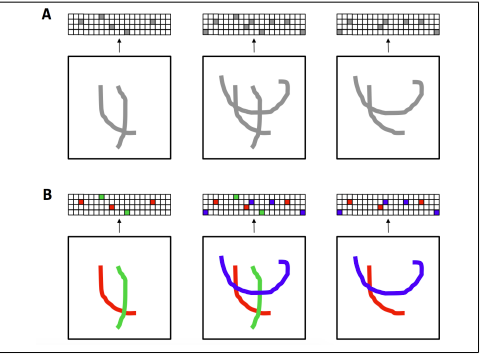


Fig: interpretability

Sparse autoencoders introduce an information bottleneck without *requiring* a reduction in the number of nodes at our hidden layers. Rather, we'll construct our loss function such that we penalize *activations* within a layer. For any given observation, we'll encourage our network to learn an encoding and decoding which only relies on activating a small number of neurons. We achieve this by modifying the objective function with a sparsity penalty, which increases the cost of any representation that has a large number of nonzero components.



The value of β determines how strongly we favor sparsity at the expense of generating better reconstructions.

**(OR)**

6.a. Why labeled data is scarce and expensive to generate. What is the basic concept behind PCA.

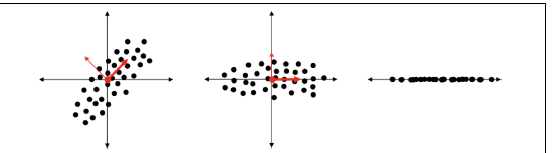
8M

**Scheme:** Discussion on labeled data and why scarce -2M

PCA explanation with figure - 6M

Labeled data is scarce and expensive as it requires human effort especially expertise in the domain to generate labels.

The basic concept behind PCA is that we’d like to find a set of axes that communicatesthe most information about our dataset. More specifically, if we have d dimensionaldata, we’d like to find a new set of m < d dimensions that conserves asmuch valuable information from the original dataset. For simplicity, let’schoose d = 2,m = 1. Assuming that variance corresponds to information, we can performthis transformation through an iterative process. First we find a unit vectoralong which the dataset has maximum variance. Because this direction contains themost information, we select this direction as our first axis. Then from the set of vectorsorthogonal to this first choice, we pick a new unit vector along which the datasethas maximum variance. This is our second axis. We continue this process until wehave found a total of d new vectors that represent new axes. We project our data onto this new set of axes. We then decide a good value for m and toss out all but thefirst *m* axes (the principal components, which store the most information).

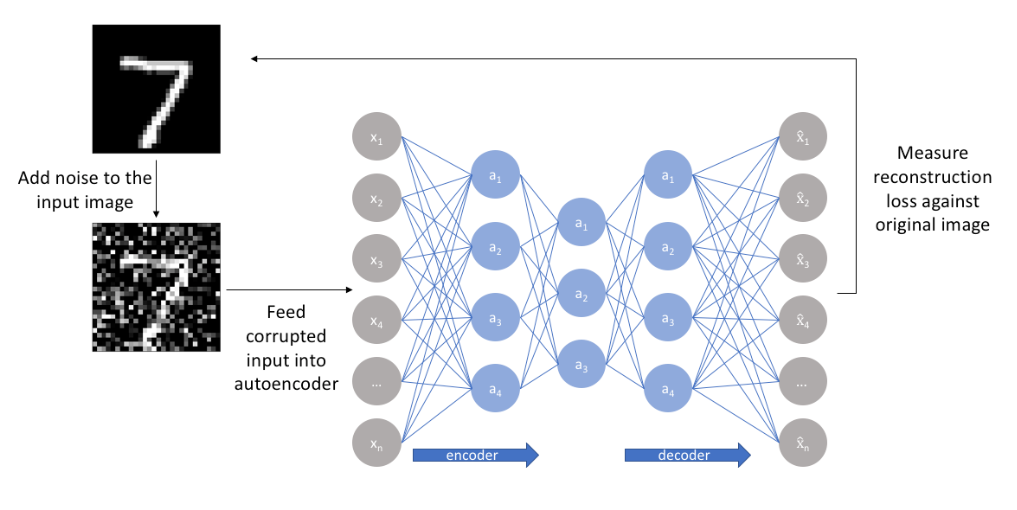


b. Explore mechanisms to improve the ability of the autoencoder to generate embeddings that are resistant to noise. 7M

**Scheme:** Denoisy auto encoder – diagram -2M, explanation -5M

Even though a corrupted version of a digit hits our retina, our brain is still able to reproduce the set of activations (i.e., the code or embedding) that we normally would use to represent the image of that digit. Denoisy autoencoders are also made to generate embeddings that are resistant to noise . We corrupt some fixed percentage of the pixels in the input image by setting them to zero. Given an original input X, let’s call the corrupted version C(X).

The input to the encoder network is the corrupted C( X ) instead of X. In other words, the autoencoder is forced to learn a Embedding and Representation Learning code for each input that is resistant to the corruption mechanism and is able to interpolate through the missing information to recreate the original, uncorrupted image.

. 

**UNIT-IV**

7.a. Design a wide variety of recurrent neural networks. 8M

**Scheme:** 3 design patterns – Listing -2M

Each diagram and explanation -2M, 3X2=6M

RNN design patterns- 3 types

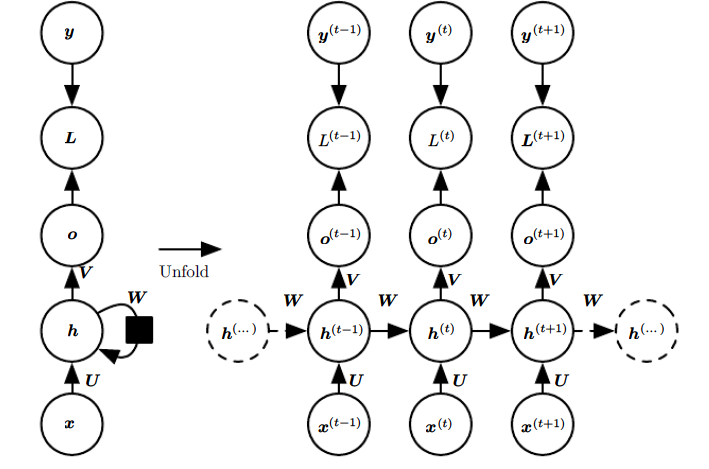
* Recurrent networks that produce an output at each time step and have recurrent connections between hidden units
* Recurrent networks that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step
* Recurrent networks with recurrent connections between hidden units, that read an entire sequence and then produce a single output

b. Illustrate how feed forward neural networks break when analyzing sequences. 7M

**Scheme:** (Any RNN model can be explained.)

Diagram - 3M

Explanation - 4M



* A recurrent network that maps an input sequence of *x* values to a corresponding sequence of output *o* values
* A loss *L* measures how far each *o* is from the corresponding training target *y*
* The RNN has input to hidden connections parameterized by a weight matrix *U*
* hidden-to-hidden recurrent connections parameterized by a weight matrix *W*

and hidden-to-output connections parameterized by a weight matrix *V*

**(OR)**

8.a. Explain the common ways of providing an extra input to an RNN. 8M

**Scheme:** Discussion – 4M

Explanation with diagram - 4M

* RNNs take a sequence of vectors *x*(*t*) for  *t* = 1*, . . . , τ* as input. Another option is to take only a single vector *x* as input.
* When *x* is a fixed-size vector, we can simply make it an extra input of the RNN that generates the y sequence. Some common ways of providing an extra input to an RNN are:

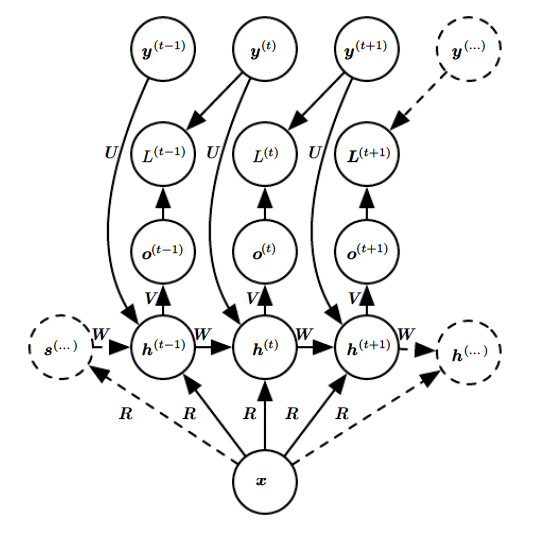
1. as an extra input at each time step, or

2. as the initial state *h*(0), or

3. both.

* The RNN shown in figure is appropriate for tasks such as image captioning, where a single image is used as input to a model that then produces a sequence of words describing the image.
* Each element *y*(*t*) of the observed output sequence serves both as input (for the current

time step) and, during training, as target (for the previous time step).



b. Explain the challenge associated with long term dependencies. 7M

**Scheme:** Mentioning of vanishing and gradient descent – 3M

Explanation of each -2M, 2X2= 4M

The basic problem is that gradients propagated over many stages tend to either vanish (most of the time) or explode (rarely, but with much damage to the optimization).

Vanishing Gradient occurs when the derivative or slope will get smaller and smaller as we go backward with every layer during backpropagation.When weights update is very small or exponential small, the training time takes too much longer, and in the worst case, this may completely stop the neural network training.

Exploding gradient occurs when the derivatives or slope will get larger and larger as we go backward with every layer during backpropagation. This situation is the exact opposite of the vanishing gradients. This problem happens because of weights, not because of the activation function. Due to high weight values, the derivatives will also higher so that the new weight varies a lot to the older weight, and the gradient will never converge. So it may result in oscillating around minima and never come to a global minima point.

Even if we assume that the parameters are such that the recurrent network is stable (can store memories, with gradients not exploding), the difficulty with long-term dependencies arises from the exponentially smaller weights given to long-term interactions (involving the multiplication of many Jacobians) compared to short-term ones.

Suppose a computational graph consists of repeatedly multiplying by a matrix W. After t steps this is equivalent to multiplying by Wt.

• Suppose W has an eigendecomposition W=Vdiag(λ)V-1 . • In this case it is straightforward to see that

Wt=(Vdiag(λ)V-1)t =Vdiag(λ)t V-1

• Any eigenvalues λi that are not near an absolute value of 1 will either explode if they are greater than 1 in magnitude and vanish if they are less than 1 in magnitude. Vanishing gradients make it difficult to know which direction the parameters should move to improve cost.