CS 181 LECTURE 2/15/24

Scribe Notes

Contents

1 Neural Networks

1.1 History of Neural Networks

Rosenblatt invented the perceptron machine. The inspiration for many of these models come from the brain. Many times in history, neural networks have been overhyped. In the 50's, because of the connection to the brain, people are promised things that the neural network can't live up to.

In the 80's and 90's, people discovered that neural networks are quite powerful in image recognizing. As part of the project, the people at the Bell Labs created the MNIST dataset, which is still used today. Datasets that are publicly available and easy to use have allowed for the growth of neural networks.

A more recent project is the sequence to sequence model from Google. Neural networks have been very quickly adopted and used in practice. Google translate in 2014 to after, there was a sudden spike in quality because of neural networks and the code base was simplified.

1.2 Intuition

We have talked about linear models but using basis function to allow for greater complexity.

$$w^\top x + w_0$$

to

$$w^\top \phi(x) + w_0$$

The idea that you can change the representation of data is key to neural networks. We looked at a variety of basis functions – polynomial, cosine, etc – but how do you choose which one to use? In neural network, we learn the ϕ. So, instead of choosing your representation, you learn what the best representation is.

Let's start with our function f.

$$f = w^\top \phi(x) + w_0$$

Let us represent $\phi(x)$.

$$\phi(x)_j = \sigma(\tilde{w_j}^\top x + \tilde{w}_{0,j})$$

for $j \in 1, 2, ..., J$.

1.3 Graphical Representation

We have a neural network with one hidden layer - which is also known as a two layer NN.

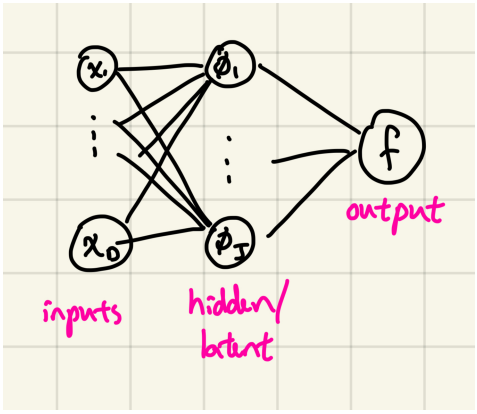

Figure 1: 1-hidden layer network.

1.4 Matrix Representation

X is dimensions (N x D), Φ is dimensions (N x J), and we have our final f that is dimension (N x 1).

1.5 How powerful are these NNs?

With the neural network that we just described with one hidden layer, we can approximate any linear function! This is called the universal approximation. Lets see why this is the case through some examples.

2 Example 1: Decision Boundary

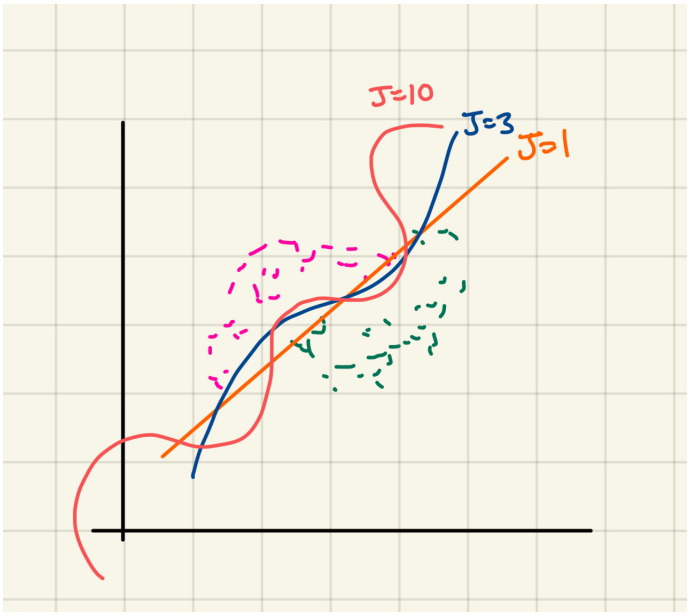

Figure 2: Two moons dataset.

What would happen if we use the model with the single basis function? We would get a linear function!

If $J = 3$, then we get a squiggly line. If we start adding more and more bases, we end up getting a very squiggly line. By playing with the width of the network, we get more complex functions. This highlights a problem when making a model too wide: overfitting.

3 Example 2: XOR

x_1	x_2	XOR
0	0	0
1	0	1
0	1	1
1	1	0

The following functions in neural net can be used to represent the XOR function:

$\phi_1 = \sigma(10x_1 + 10x_2 - 5)$ $\phi_2 = \sigma(-10x_1 - 10x_2 + 15)$ $f = 10\phi_1 + 10\phi_2 - 15$
Why does this work? $\phi_1 = \sigma(10x_1 + 10x_2 - 5)$ picks out [1,1]

$\phi_2 = \sigma(-10x_1 - 10x_2 + 15)$ picks out [0, 0]

$f = 10\phi_1 + 10\phi_2 - 15$ combines the two ϕ functions
We do this in neural networks where we use functions to pick out aspects of the model that we desire.

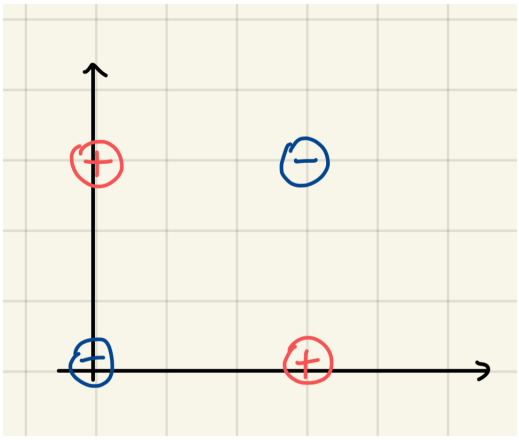

Figure 3: Graphical representation of XOR function.

4 Example 3: Rings

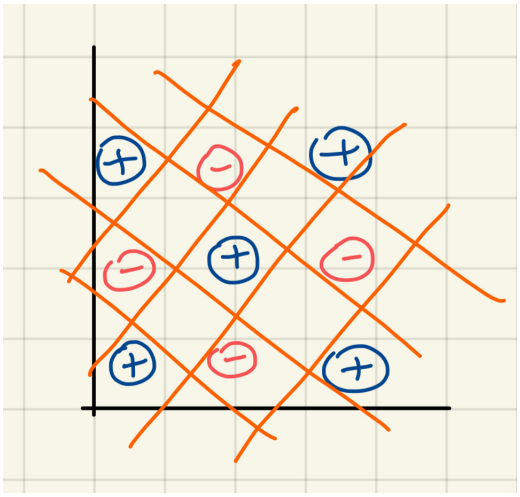

Figure 4: Rings.

There is no way you can solve this with a linear classifier. You want something that has multiple linear boundaries. There needs to be a nested sequence of classifiers. By playing with the width of the network, we can classify something complex like this problem.

What happens when we are doing multiclassification? We can map to k dimensions. Now, the output of the vector is size k and we use softmax to get probabilities.

Note on softmax: is a mathematical function that converts a vector of real numbers into a vector of probabilities. It is commonly used in multi-class classification problems to transform the raw output scores (logits) of a model into a probability distribution over multiple classes.

5 Deep Neural Networks

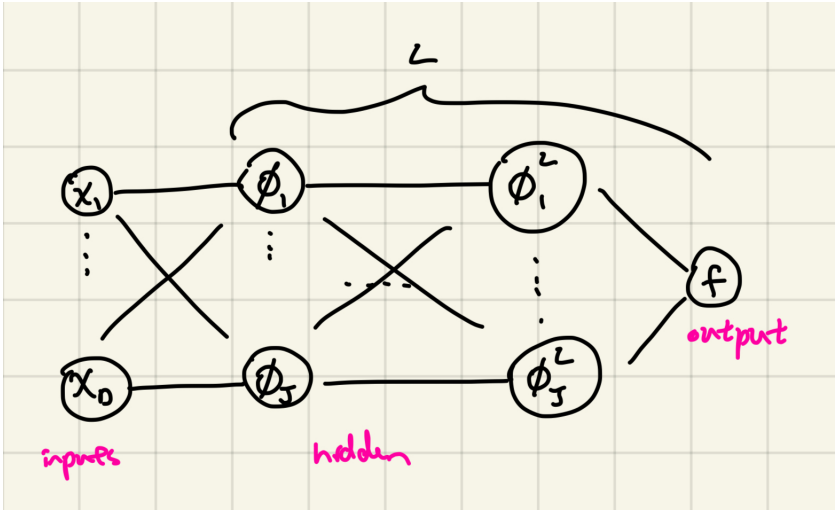

Figure 5: Graph of neural network where we have L hidden layers.

$$f(x) = W^L \sigma(W^{L-1} + ...\sigma(W_x^1 + W_0^1) + W_0^{L-1}) + W_0^L$$

We could also see the final formula of f as a composition of functions.

$$f(x) = f^L \circ ... \circ f^1(x)$$

Why should we use deep neural network?

1. For some functions, we need exponentially many neurons to represent them. This is intractable and infeasible if we only use the 1-hidden layer NN representation, so deep neural networks come into play.
2. Deeper networks are usually easier to train than wider ones.
3. Feature reusability: Let's think about a logical classification problem where the output is true iff x_1 and x_2 are big and x_3 or x_4 are big. In a 1-hidden layer NN, we will learn a ϕ_1 when x_1, x_2, x_3 are big and another ϕ_2 when x_1, x_2, x_4 are big. With a deep NN, we can learn a ϕ_1 for x_1 and x_2 are big and a ϕ_2 for x_2 is big and a ϕ_3 for x_4 is big. Then, we can combine the ϕ's and we only needed to learn each part once, then reuse. In the 1-hidden layer, we were capturing the x_1 and x_2 being big twice, meaning we had to learn this twice.

6 More Types of Neural Networks

6.1 Convolutation Neural Networks (CNNs)

The intuition behind Convolutional Neural Networks (CNNs) stems from the understanding of how human visual perception works and the desire to mimic this process in artificial systems for tasks like image recognition, object detection, and segmentation.

CNNs are designed to learn hierarchical representations of visual features. This means that early layers in the network learn low-level features like edges and gradients, while deeper layers learn increasingly complex and abstract features. Each layer in a CNN captures higher-level features by combining and abstracting the features detected in the previous layer, leading to a hierarchical representation of the input data.

Pooling layers are interspersed between convolutional layers in CNNs to progressively reduce the spatial dimensions of the feature maps while retaining the most salient information. Pooling operations (e.g., max pooling) aggregate information from small local regions of the feature maps, helping to make the learned features more robust to variations in the input data.

6.2 Residual Net

The residue of the input you add back. Why does this work? We add something linear to something more complex. It works because it is easier to train. If you follow the shortcuts, you can model the identity.

6.3 Recurrent Neural Net

The intuition behind Residual Networks (ResNets) arises from the challenge of training very deep neural networks effectively. As neural networks grow deeper, they encounter difficulties in learning meaningful representations due to vanishing gradients and the risk of degradation, where deeper networks start to perform worse than shallower ones. ResNets address these challenges by introducing skip connections or shortcuts that enable the network to learn residual functions, which are the differences between the input and output of certain layers.

Instead of trying to learn the entire mapping from input to output, ResNets focus on learning residual functions, which represent the difference between the input and output of a layer. By learning residual functions, the network can focus on refining the learned features rather than completely relearning them, leading to more efficient training and better generalization.

We apply the function A over and over again. We keep updating the state. This is the backbone of most NLP's until very recently.

6.4 Autoencoders

Autoencoders consist of an encoder and a decoder, which work together to encode the input data into a lower-dimensional latent space and then decode it back to the original input space.

6.5 Attention

What attention does is taking your neural network and have it look at other inputs and do a weighted sum. take inner product and take softmax, then feed back into the model. This is a way of relating multiple inputs like the words in a sentence. This framework generalized is called Transformer. The model is looking at previous words to generate the next word.

Main Takeaways

First key takeaway is that there are lots of different architectures and you have the choice. At some point we need to decide how to use them. Second takeaway is that certain types of inputs work well with certain types of architectures. Convolution NNs are great for images because you can analyze the different parts of images. Recurrent networks are good for sequences. Finally, there is a question of how deep and how wide you make your neural network.

7 Concept Check

Suppose that classification problem is: anywhere in Fig 6, you can find the pattern in Fig 7?

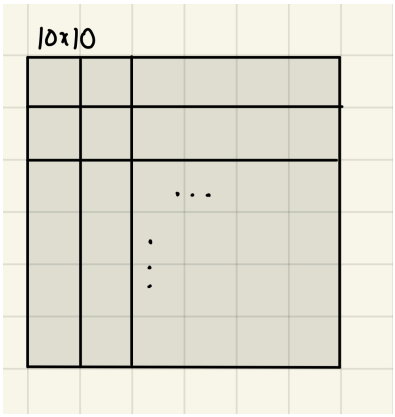

Figure 6: 10x10 image.

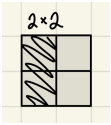

Figure 7: 2x2 pattern.

1) Can you solve this with a convolutional network?

2) What happens when you want to identify two types of patterns? Say we want to detect whether we have two patterns in the image, the first is Fig 7 and the second is Fig 8.

Figure 8: 2x2 pattern.

3) Only classify when the pattern (Fig 7) is on the edge. Now, can you solve this with a CNN?

Solutions are on the next page.

1) Can you solve this with a convolutional network?
Filter goes through every two by two subsection in the image and see if there is a high similarity between that subsection and the pattern. The filter is looking for something that looks like itself and if it does, it assigns a high value.

2) What happens when you want to identify two types of patterns?
No, unless we had two filters but with a single filter we can't.

3) Only classify when the pattern is on the edge. Now, can you solve this with a CNN?
Yes, this is possible. You can achieve this with the weights, by setting the weights so that they pick and choose parts of the image.