CS 181 LECTURE 2/20

Scribe Notes

Contents

1 Intro

Take a look at the following website: https://restofworld.org/2023/ai-image-stereotypes/.

There are a lot of stereotypes in this AI generated images. For example, asking for an Indian person, you get the same man with the same outfit, not depicting the variety in what a person looks like from this culture. Similar to asking for New Dheli, you get a pictures of the same street. Today, we will talk about optimization, but looking at this, we see that what comes out of the machine is the reflection of the data.

When Google Images first came out and you search for CEO, in 2017, it was all white men. When you search in 2024, there is much more diversity. The representation of CEOs is still dominated by white man. If you represent the world as is, your search would continue to show white men only. But Google is making a conscious choice to show a more diverse representation. The data didn't change but the objective changed. And the objective is under our control.

So when it comes to AI image generation, we have a similar choice. We can change our objective to serve a larger purpose. There is a similar moment now, and we can remain hopeful because we saw the improvement in Google Images.

1.1 Where we are now

Last time, we laid out supervised learning in simple terms. We talked about a lot different deep neural architectures. Now, we will talk about how to do the optimization for these classes.

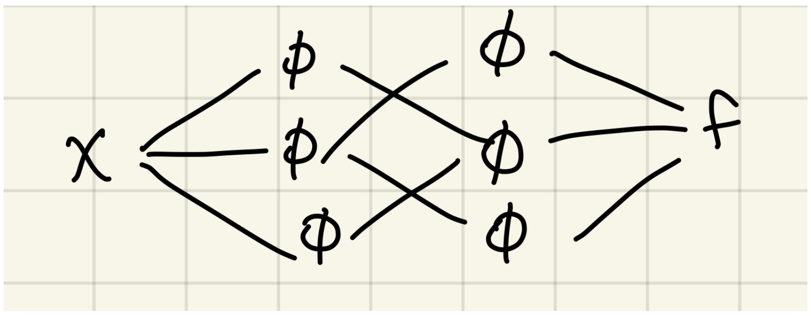
1.2 Terminology Review

We start off with **data**, some (x, y) pairs. The **model class** is the shape of the function, ex. linear, basis regression, etc. The model class has unknown **parameters**. We select the model class. Lastly, we have an **objective function**, ex. hinge loss. The objective function is also called the **loss**.

Then we do the **inference** step, where we find the parameters to minimize the loss function. This step is also called **training** the model.

2 Math: Backpropagation

This architecture is easy to work with and expressive (remember when we made the model very wide, it is a universal function).



Let us define our loss function as the sum of the loss from each (x_n, y_n) point.

$$\mathcal{L}(w) = \sum_n \mathcal{L}_n(w)$$

We can choose our specific loss function to be squared error, for example:

$$\mathcal{L}(w) = \sum_n (f_n - y_n)^2$$

where $f_n = f(x_n)$.

$$\frac{\partial \mathcal{L}_n}{\partial f_n} = (f_n - y_n)$$

We can take the output of the network and convert it to a loss. (We saw this in last lecture.)

$$\text{log loss } \mathcal{L}_n = y_n \log(1 + \exp(-f_n)) + (1 - y_n)\log(1 + \exp(f_n))$$

We can write down the derivative of the loss with respect to the output of the network.

$$\frac{\partial \mathcal{L}_n}{\partial f_n} = \frac{y_n \exp(-f_n)}{1 + exp(-f_n)}(-1) + \frac{(1 - y_n)\exp(f_n)}{1 + \exp(f_n)}$$

We do this derivative because our goal is to tune the parameters of the network. Reference the textbook section 4.4 to learn more.

2.1 Example

$\phi_j = \sigma(\sum_d w^1_{jd} x_d + w^1_{0,j})$

$f = \phi w^2 + w^2_0$

We have four sets of weights to optimize. So, we need to use gradient descent.

2.2 Review of the Chain Rule

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial v}\frac{\partial v}{\partial x}$$

Suppose u, v, x are vectors.

$\nabla_x y = \nabla_u y \mathcal{J}_y(u)\mathcal{J}_x(v)$

When we change dimension of v, we affect every dimension of u, which is represented by the $\mathcal{J}_y(u)$.

Changing the x vector changes the u vector, which is represented by the $\mathcal{J}_y(u)$.

2.3 Going Back to the Example

$f_n = \phi_n w^2 + w_0^2$

Suppose I want $\frac{\partial \mathcal{L}_n}{\partial w^2} = \frac{\partial \mathcal{L}_n}{\partial f_n}\nabla_w f_n$.

We already have $\frac{\partial \mathcal{L}_n}{\partial f_n}$, as we saw in an earlier section of this lecture. And, $\nabla_w f_n$ is ϕ_n.

For squared loss,

$$\frac{\partial \mathcal{L}_n}{\partial w^2} = (f_n - y_n)\phi_n$$

So, now let us take a look at this: $\phi_{nj} = \sigma(\sum_d w_{jd}^1 x_{nd} + w_{j,0}^1)$. The question is hwat is the derivative of the loss with respect to the w_{jd}^1. We can use the chain rule - w changes ϕ which then changes f which then changes the loss.

$$\frac{\partial \mathcal{L}_n}{\partial W^1} = \frac{\partial \mathcal{L}_n}{\partial f_n}\nabla_\phi f_n \mathcal{J}_W \phi_n$$

Dimensions of $\mathcal{J}_W \phi_n$ is (J x JD).

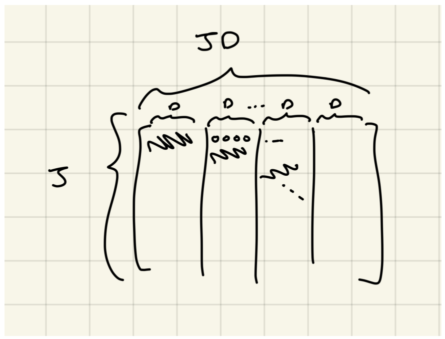
Dimensions of $\nabla_\phi f_n$ is (1 x J).

Dimensions of $\frac{\partial \mathcal{L}_n}{\partial f_n}$ is (1 x 1).

Let us do the last derivative of ϕ with respect to w.

$$\frac{\partial \phi_{nj}}{W_{dj}^1} = \sigma(\sum_d w_{dj}^1 x_{dn} + w_{j,0}^1) \cdot (1 - \sigma(\sum_d w_{dj}^1 x_{dn} + w^1 j, 0)) \cdot x_{dn}$$

If you are trying to compute the ginormous matrix of size (J x JD), there are going to be a bunch of zeroes.



The main point is that we can compute these derivatives. Everything may seem hairy and complicated but at the end of the day it is the chain rule.

2.4 Payoff of the Backprop Algorithm

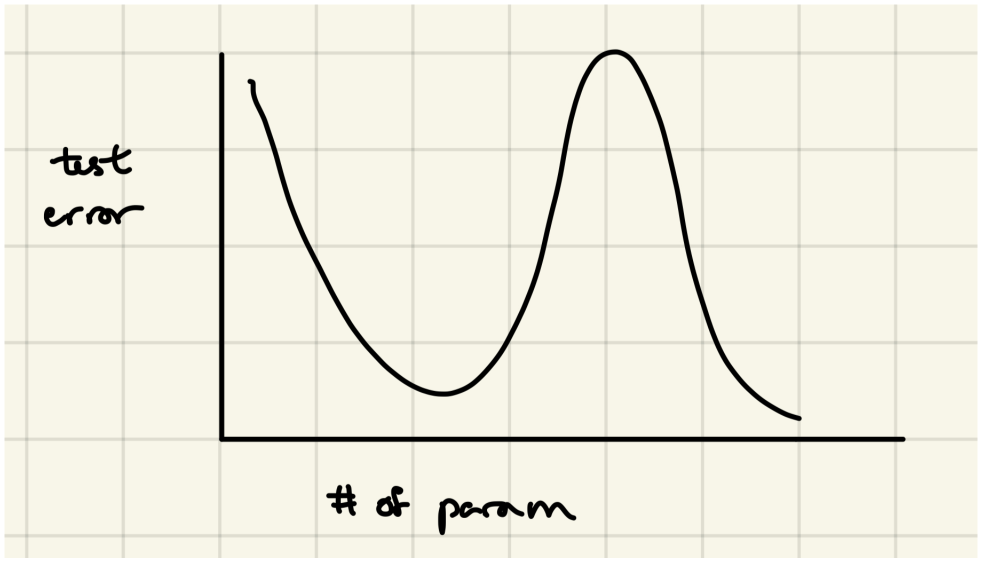
The key takeaway is that we need the derivatives for all $W^1, W^2, ..., W^L$.

$$\frac{\partial \mathcal{L}_n}{\partial W^l} = \nabla_{\phi^l} \mathcal{L}_n \mathcal{J}_{W^l}[\phi^l]$$

During the forward pass, input data is passed through the neural network layer by layer, from the input layer to the output layer. At each layer, the input is transformed by a set of parameters (weights and biases) and activation functions to produce an output. The output of one layer serves as the input to the next layer until the final output is generated. This process is called the forward process because it moves forward through the network, applying transformations to the input data to produce predictions or activations.

The backward pass, also known as backpropagation, occurs after the forward pass when the loss or error between the predicted output and the true labels is computed. The goal of the backward pass is to update the parameters of the neural network (such as weights and biases) to minimize the loss. It involves computing the gradients of the loss function with respect to the parameters of each layer in the network, starting from the output layer and moving backward through the network. These gradients represent the sensitivity of the loss function to changes in each parameter, indicating how much each parameter contributes to the error. The gradients are computed using techniques like the chain rule of calculus, which allows the gradients to be efficiently propagated backward through the layers of the network. Once the gradients are computed, optimization algorithms like gradient descent are used to adjust the parameters of the network in the direction that minimizes the loss.

3 Concept Check



1) As the number of parameters increases, can bias increase?

2) If the number of parameters is way less than the number of datapoints, can we fit the data perfectly?

If the number of parameters is equal to the number of datapoints, can we fit the data perfectly?

If the number of parameters is way greater than the number of datapoints, can we fit the data perfectly?

3) Why might we see this second descent in the loss?

1) As the number of parameters increases, can bias increase? No.

2) If the number of parameters is way less than the number of datapoints, can we fit the data perfectly? No.

If the number of parameters is equal to the number of datapoints, can we fit the data perfectly? Yes! One model that does this.

If the number of parameters is way greater than the number of datapoints, can we fit the data perfectly? Yes! Multiple models that do this.

3) Why might we see this second descent in the loss? Regularization happens implicitly. When you have lots of functions to choose from, you choose one that is more smooth.