Neural Nets Intro

Main Ideas

- Neural nets give us a way to learn complicated functions from data
- The basic neural network is a composition of many simple mathematical functions
- The net learns a complicated function by minimizing the loss function

Learning from Data

- Functions are often too complicated to write by hand
  - Ex: Function to generate art
  - Ex: Function to play Go
- Instead of writing function by hand, learn it from data
- Gather dataset of examples of function we want to learn

Basic Neural Network

- Composition of matrix multiplies and non-linearities
- Our input data is a vector
  - Most any data we have can be represented as a vector, for example flattening pixel values of images

Let's call our input vector v. Our neural network will transform this vector into some output vector. The network will consist of some matrices, lets say W_1, W_2. It will also consist of a nonlinear function, called a ReLU (Rectified Linear Unit). The ReLU takes a vector as input and outputs the positive part of the vector. In other words it operates on the vector elementwise, and for each entry v_i in the vector it outputs $max(0, v_i)$.

The basic neural network composes these functions like this:

$$W_2 \circ ReLU \circ W_1(v)$$

So we apply the matrix multiply defined by W_1, then the ReLU function, then the matrix multiply defined by W_2.

Since we have a composition of a few functions, the result may be a function that is more complicated than a single matrix multiply. We can also repeat this process of matrix multiplies and ReLU functions more than twice to potentially get an even more complicated function:

$$W_k \circ ReLU \circ W_{k-1} \circ ... \circ ReLU \circ W_1(v)$$

Question: what is the purpose of including the ReLU instead of just composing a bunch of matrix multiplies with no nonlinearity?

Training the Network

- Cannot just use random W_i and hope our network is close to our desired function, need some way to choose these matrices
- Use data to teach network
- Define a “loss function” which tells how badly net is doing
- Update net with gradient descent to minimize loss function

I have said that a neural network can potentially approximate a function we are interested in. However, we cannot simply choose matrices W_i at random and hope to be anywhere close to our function of interest. We need to choose these matrices very carefully. For convenience, let $\theta := \{W_1, ..., W_k\}$ denote the set of all our parameters.

First we need some way to measure how badly our network is doing. The exact way to measure this is problem dependent. For now, let $L(\theta)$ be a real-valued function that says how badly the network is doing. We will look at actual loss functions for language models later.

We will find a good set of parameters θ by using **Gradient Descent**. This is a method that is very general and can be applied to things other than neural networks. The method works as follows. First we initialize θ randomly. Then we update the parameters many times by following $\theta := \theta - \gamma \nabla L(\theta)$ where γ is a small number called the **learning rate**.

Pseudocode for this process is shown below:

```
theta = random_init()
gamma = 1e-3

while not converged:
    theta = theta - gamma * grad(L(theta))
```

Question: why does this minimize the loss (for reasonable learning rate)?

Question: what might happen if the learning rate is too small? What about too large?

Question: what would happen if we add the gradient rather than subtract?

Backpropagation

- Computing derivatives for each parameter naively could take a long time
- Many computations for the derivatives are shared for different parameters
- Backpropagation reuses these computations instead of computing them multiple times
- We will not go into more detail for this course, but feel free to look online if you are interested