Neural Networks and Language

CS 360 Machine Learning

Week 11, Day 1

April 9, 2024

Contents

1 Processing language with models

When processing language you could go letter by letter, word by word, phoneme by phoneme (parts of word) or other strategies. We'll go character by character. When considering the window length and the target length, suppose that the window length was 5 (i.e. 5 letters at a time) and the window included "to be". We'd consider what the following letter might be, and there are a number of options that might be natural to come after this. In order to encode the information in this window, we could translate character by character to the position in the alphabet, and we could end up with:

$$x = [20, 15, 27, 2, 5]$$

The target would be the predicted next letter tacked on to the shifted window:

$$y = [15, 27, 2, 5, 27]$$

Why is the target not just the next character? It's so that you keep the context and it makes it easier to train the resulting model. This also means that if we only have one character of context we can start to build up that context, since the target information contains some of that information as well. This can help us to jump start training with less data. While the resulting model will predict this entire target, we'll only care about the final character of the output, that unknown next character.

1.1 Determining the output

Recall from previous classes the notion of *softmax*. In this case, we'll have some scores:

$$scores = [s_1, s_2, .., s_v]$$

where v is the size of the vocabulary (or number of characters). We can then compute the softmax of these scores:

$$softmax(s_i) = \frac{e^{s_i}}{\sum_{j=1}^{v} e^{s_j}}$$

This gives us a probability distribution, i.e., a probability that each item in the vocabulary (each letter) is next in the sequence. One natural choice is to just choose the character with the highest probability, but if we do that then we'll keep ourselves from being able to generate some types of text. For example, with "to be" we might never manage to generate "to become" but only ever generate "to be ". So instead, we could sample proportionally to the resulting probabilities. The *temperature* can help us decide exactly how to use these probabilities; a temperature of 1 uses the probabilities directly, a high temperature will essentially sample uniformly from the available characters, and temperature values of less than 1 will weight the high probability predictions more heavily. This can help us determine the output.

1.2 Determining the input

When considering the input, note that one-hot encoding is not a good idea if I have a huge vocabulary size. The vectors would get longer and longer and consist of a lot of zeros in this case - one-hot encoding does not scale well! Instead, the input is generally encoded through a learned *embedding* where words are represented in a lower dimensional space. In these embedding spaces, represented via vectors per word, or sparse matrices for the full input, words are close to each other if they are related. For example, "woman" is closer to "queen" while "man" is closer to "king." (These embedding spaces are known to replicate language associations and also societal biases that are embedded in language. See for example, Man is to Computer Programmer as Woman is to Homemaker and Semantics derived automatically from language corpora contain human-like biases.)

2 Attention mechanisms

For all of these architectures the goals are the same, but the specific mechanisms are different. The idea of attention mechanisms, from the paper "Attention is all you need" has been very influential in the field. Attention is a way of quantifying how important every word in the sentence is (though we'll use characters) when considering every other word in the sentence. So attention is a property between pairs or sets of words (or characters).

Attention as a function takes three parameters:

- Q are the queries, such as a specific word that we're interested in. This might be the next word in the sentence, or not.
- K are the keys. These could be all the other words, or all the other words in a specific block of text.
- V is the value. This is flexible and could be different various roles. For self-attention this is a transformation of the input. In some cases it's a label, and in general we can think about this as the target value.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_{keys}}}\right)V$$

Where d_{keys} is the dimension of the keys which is usually the same as the values and the queries. What does this all mean?

Recall the idea of kernels. Suppose that we have inputs and outputs and we want to learn the relationship between them. This can be for a general task or a language-specific task. Let's call these inputs and outputs:

$$(\vec{x}_1, y_1), (\vec{x}_2, y_2)...(\vec{x}_t, y_t)$$

We can think of t as the time, and we'll think of the vectors $\vec{x}_i$ as existing in some embedding. The goal is to predict $\hat{y}$ for a new point $\vec{x}_0$ (a test example).

What's the simplest thing I could do? I could use k nearest neighbors for $k = 1$ and predict the same outcome as the closest point. Or we could average over all the data weighted by how similar the example point is to the test example. Suppose that we have some kernel function $K(\vec{u}, \vec{v})$ to measure similarity between $\vec{u}$ and $\vec{v}$. Again, we could make our prediction using the nearest neighbors idea to predict $\hat{y}$:

$$\hat{y} = \sum_{i=1}^{t} y_i \frac{K(\vec{x}_i, \vec{x}_0)}{\sum_{y=1}^{t} K(\vec{x}_j, \vec{x}_0)}$$

The numerator gives the similarity between the test and each training example, and the denominator normalizes. This idea is called *kernel smoothing*. This gives a probability distribution and uses it to weight the y values by that distribution. This basic function is attention! I can think about the kernel as the softmax function, and attention is producing a weighted average over the y values.

Consider an example, $K(\vec{u}, \vec{v}) = e^{(\vec{u} \cdot \vec{v})}$. Suppose that $\vec{u}$ and $\vec{v}$ are perpendicular, then $\vec{u} \cdot \vec{v} = 0$. Recall that $\vec{u} \cdot \vec{v} = ||\vec{u}||\ ||\vec{v}|| cos\theta$. So $exp(0) = e^0 = 1$. This is a measure of the similarity. Suppose that these vectors are not perpendicular, but rather at an acute angle. In this case, $\vec{u} \cdot \vec{v} > 0$; these are more similar. And if the angles are obtuse, then the dot product is less than 0. For kernel smoothing we really need a non-negative value, because they're part of a weighted average to generate probabilities, and the resulting probabilities need to be positive. So by taking these dot products as an exponent we can ensure the results are positive. This specific function choice will also heavily downweight the examples that aren't important and upweight the examples that *are* important to the prediction; which is good for the attention mechanism!

Consider the below transformation of the kernel function:

$$K(\vec{u}, \vec{v}) = exp((w_1\vec{u}) \cdot (w_2\vec{v}))$$

where w_1 and w_2 are weight matrices that can shift and modify the vectors into a different space. These are like the multiplication of the queries and the keys in the attention mechanism.

What does this look like in practice? Suppose that $\vec{x}_0$ is a word that has been embedded – this is our query, $\vec{x}_i$ is another word in that sentence – these are the keys, and $\vec{y}_i$ is the next word (the target after $\vec{x}_0$) – this is the value. When we consider applying this in a transformer context, we don't apply this to all the words in a sentence. For example, we don't include words that come after a word in the sentence.

Suppose we had three characters in our sentence. The attention map starts out looking like the below with a window size of 3. This means that the second character can only pay attention to the first character, and so on.

$$\begin{bmatrix} 0 & -\infty & -\infty \\ 0 & 0 & -\infty \\ 0 & 0 & 0 \end{bmatrix}$$

This is then transformed using softmax, e.g., multiplying by $\frac{e^0}{e^0+e^{-\infty}+e^{-\infty}}$ for the first row, giving the below attention map:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 0.5 & 0 \\ 0.33 & 0.33 & 0.33 \end{bmatrix}$$

So even before we have any data, this an average weight for each of the preceding characters, ensuring that characters can't peek ahead in a sentence.