# HW 5

Andrei A Simion

February 11, 2023

## 1 WaveNet Language Model

Please fill in the notebook related to this file. Note that the start has a small computation related to Layer Norm so you get a hands on experience with what it is doing.

We are going to build a Language Model which is based on the Wave Net structure (strictly speaking, that one is based on CNNs and has neat tricks, but this is similar and it has the core idea). Out goal will be as in HW 1 to predict the next character from a fixed context window of characters before this character. At a high level, we have a context of a fixed size. For example, we can use a context size of 256 characters to predict the next character. This is *context_size* in the notebook. The idea of this model is that you have a huge context but at each layer you half the context size. By the end, you are essentially predicting with a context of 256 but you only have about 6 layers, not 256.

The big idea of this model is as follows. Your input is originally of size $(N, 256)$. When you get the embeddings, this becomes $(N, 256, d\_model)$, where $d\_model$ is 384 (for example). Now, you reshape this into $(N, 128, 2 * d\_model)$ so that effectively you've halved your sentence length but the embedding is double what it is before, you basically concatenated adjacent words. For example, if you had 1 element in a batch and the embeddings were $(e_1, e_2, \ldots, e_256)$, now they are $((e_1, e_2), (e_3, e_4), \ldots, (e_{255}, e_{256}))$. You then pass this through a forward layer and get a new hidden layer of size 128; this layer is like $(h_1, h_2, \ldots, h_{128})$ where $h_i$ is gotten from $(e_{2i-1}, e_{2i})$ so each $h_i$ has information about two consecutive characters ($h$ might have a different dimension than $e$). The linear layer is good at going from 2 concatenated embeddings 1 embedding, since it applies to all pairs $(e_{2i-1}, e_{2i})$.

You do this again and again, so that the next time $(h_1, h_2, \ldots, h_{128})$ becomes $((h_1, h_2), (h_3, h_4), \ldots, (h_{127}, h_{128}))$ and the context that results is of size 64. Now, each hidden node has information about 4 (of the original) characters. Etc. After abut 7 layers, you have the full context of size 256. Think of this like a funnel, wider at the bottom and shorter at the top. This is unlike the Bengio model from Lecture 1: that model takes the entire vector of size $d\_model * 256$ and squashes it into one hidden layer. That is, strictly speaking, a higher perplexity model (worse).

# 2 Mathematical Problems

Below are some mathematical drills.

Problem 1 Take the neural network from Lecture 4. Assume the loss is $e = \frac{(t-y)^2}{2}$. This is

$$
\begin{aligned}
s &= W_1 x + b_1 \\
h &= \text{sigmoid}(s) \\
z &= W_2 h + b_2 \\
y &= \text{sigmoid}(z)
\end{aligned}
$$

What is $\frac{\partial e}{\partial W_1}$, $\frac{\partial e}{\partial W_2}$, $\frac{\partial e}{\partial b_1}$, $\frac{\partial e}{\partial b_2}$? If we modify one equation so that $z = W_2 h + b_2 + s$, what are the gradients now?

Problem 2 Drop Out at inference time. At training time, we randomly pick weights in the network and zero them out with probability $p$. So, each weight in the network, we can zero it out with a probability $p$ and keep it with a probability $1 - p$. Express each weight as a random variable times an appropriate independent Bernoulli random variable. What is the expected value of this random variable. This is what is used at training time.

Problem 3 Imagine that you have a MLP network where each Linear layer is followed by a Batch Norm layer. Do you need the bias term in each Linear layer? Prove that it is unnecessary. Thus, prove that if you fit a model where the layer is specified with $nn.Linear(\ldots, bias = False)$ no information is lost, the bias adds nothing and you can specify this. This is one effect of using Batch Norm.