Lecture 15: Transformers Completed, and Techniques to Accelerate DNN Training.

CS4787/5777 — Principles of Large-Scale ML Systems

Position. How do transformers handle positions? Three main approaches:

- Absolute positional encoding/embedding
- Relative positional encoding (e.g. Rotary Position Embedding RoPE)
- Causal attention masks

Batch normalization. Deep neural networks can have problems during training because of higher-order effects that exist because of the composition of many layers. Small changes made to earlier layers can have a major impact on what happens later in the network. This makes training difficult, because we often want to set the step size to be very small to damp out these higher effects, but this also makes training slow. One way to address this is *batch normalization*, which reparameterizes a deep neural network to reduce these effects.

Suppose that we are concerned with a single (scalar) activation in the network, and we are running mini-batch SGD with batch size B. Let $u \in \mathbb{R}^B$ denote the vector of activations of the original network. Batch normalization replaces this u with

$$\mathsf{BN}(u) = \frac{u - \mu}{\sigma}.$$

where at training time

$$\mu = \frac{1}{B} \sum_{b=1}^{B} u_b \quad \text{and} \quad \sigma = \sqrt{\frac{1}{B} \sum_{b=1}^{B} (u_b - \mu)^2}.$$

Explicitly, we're subtracting out the mean of the minibatch, then dividing by the variance. Importantly, we **still backprop through this** as if it were part of our network. At test time, since there is no minibatch, we instead use values of μ and σ computed from the distribution of that activation over the entire training set.

Commonly, to maintain the expressive power of the network, to do batchnorm we add the additional parameters γ and β and replace the activations with

$$\mathsf{BN}(u) = \gamma \cdot \frac{u - \mu}{\sigma} + \beta.$$

We then backprop through the entire network, using all the old parameters and, additionally, the new parameters γ and β for each activation we are using batch normalization for.

Batch normalization can greatly improve the speed at which we can train a neural network, and it has become standard in many applications.

Layer normalization. For many model architectures, we have additional "dimensions" of the tensor we can use for normalization, not just the batch.

When would we use layer normalization compared to batch normalization?

Autoregressive models: the language-modeling head. Maps to a prediction over tokens in the language model for the *next* token in the sequence.

Putting it all together. The Llama-2 Model: Diagram and Demo.

Let's draw a diagram of the whole transformer.

Review: Overfitting and Underfitting and Neural Networks.

- *Underfitting* informally means that the training error is high.
- *Overfitting* informally means that the difference between the test error and the training error is high.
- *Capacity* of a model informally refers to the ability of the model to fit a wide range of possible functions.
  - Models with high capacity tend to overfit.
  - Models with low capacity tend to underfit.
- The *representational capacity* of a parameterized class of models informally refers to the extent to which for a wide range of possible functions, some model in the class approximates that function well.
  - Deep neural networks have very high representational capacity.
  - In fact, they're universal approximators.
- The *effective capacity* of a parameterized class of models *given a specific learning algorithm with a specific amount of data* refers to the extent to which for a wide range of possible functions, the model in the class produced by the learning algorithm can approximate that function well.
- For convex optimization probblems (what we've studied so far) all the algorithms we've studied converge to the global optimum, so effective capacity will be equal to representational capacity.
- On the other hand, changing the optimization algorithm for a deep learning task can alter the effective capacity. Even changing the hyperparameters of an algorithm can have this effect.

Take away point: In addition to thinking about how changes to a model or algorithm affect optimization parameters like n, d, and κ, we also need to reason about how these methods interact with the capacity of the model, especially when we're training a model with a non-convex loss.