Inference, Low-Cost Models, and Compression

CS6787 Lecture 10 — Spring 2026

Review: Inference

- Suppose that our training loss function looks like

$$f(w) = \frac{1}{N} \sum_{i=1}^{n} l(\hat{y}(w; x_i), y_i)$$

- Inference is the problem of computing the prediction

$$\hat{y}(w; x_i)$$

Why should we care about inference?

- **Train once, infer many times**
  - Many production machine learning systems just do inference
- Often want to run inference on **low-power edge devices**
  - Such as cell phones, security cameras
  - **Limited memory** on these devices to store models
- Need to get **responses to users quickly**
  - On the web, users won't wait more than a second

Inference on neural networks

- Just need to run the forward pass of the network.
  - A bunch of matrix multiplies and non-linear units.
- Unlike backpropagation for learning, here we do not need to keep the activations around for later processing.
- This makes inference a much simpler task than learning.
  - Although it can still be costly — it’s a lot of linear algebra to do.

What performance metrics do we care about when running inference?

E.g. for a deep neural network application

Metrics for Inference

- Important metric: **throughput**
  - **How many examples** can we classify in some amount of time
- Important metric: **latency**
  - **How long** does it take to get a prediction for a single example
- Important metric: **model size**
  - **How much memory** do we need to store/transmit the model for prediction
- Important metric: **energy use**
  - **How much energy** do we use to produce each prediction
- **What are examples where we might care about each metric?**

Metrics for Generative LLM Inference

- Important metric: **throughput**
  - **Tokens per second**: how many tokens can we generate in total in a length of time across many sequences
- Important metric: **latency**
  - **Tokens per second**: how long to generate a length-**N** sequence
- Important metric: **model size**
  - **How much memory** do we need to fit the model on device

Improving the performance of inference

Altering the batch size

- Just like with learning, we can **make predictions in batches**

- Increasing the batch size helps **improve parallelism**
  - Provides more work to parallelize and an additional dimension for parallelization
  - This improves **throughput**

- But increasing the batch size can make us do more work before we can return an answer for any individual example
  - Can negatively affect **latency**

Demo

Compression

- Find an **easier-to-compute network** with similar accuracy
  - Or find a network with **smaller model size**, depending on the goal
- **Many techniques** for doing this
- Usually involve some sort of **fine-tuning**
  - Apply a lossy compression operation, then retrain the model to improve accuracy

Low-precision arithmetic for inference

- Very simple technique: just use low-precision arithmetic in inference
- Can make any signals in the model low-precision
- Simple **heuristic for compression**: keep lowering the precision of signals until the accuracy decreases
  - Can often get down below 8 bit numbers with this method alone
- **Binarization/ternarization** is low-precision arithmetic in the extreme

Pruning

- **Remove parts of the model** that are usually zero
  - Or that don't seem to be contributing much to the model
- Effectively creates **a smaller model**
  - This makes it easy to retrain, since we're just training a smaller network
- There's always the question of whether training a smaller model in the first place would have been as good or better.
  - But usually pruning is observed to produce benefits.

Pruning Weights

- **Remove weights** that are zero or small
- **Induces sparsity in weight matrices**
  - How does this affect the metrics we care about?
- Many different methods to choose what to prune
  - Most common: **magnitude pruning**
  - SNIP (Lee et al), Grasp (Wang et al), SynFlow pruning (Tanaka et al): can prune at initialization

Pruning Activations/Neurons/Channels

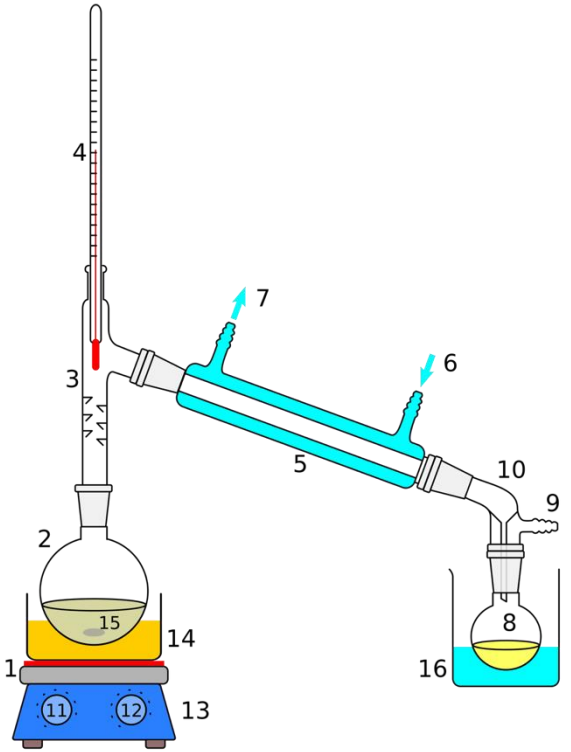
- **Remove whole activations** that are usually small
- **Weight matrices stay dense**
  - How does this affect the metrics we care about?
  - How does this compare to weight pruning?
- Many different methods to choose what to prune
  - Most common: still **magnitude pruning**

Fine Tuning

- After pruning/compressing a model, we usually **lose some accuracy**
- We can recover (some of) the lost accuracy **by running more epochs of our learning algorithm**
  - SGD/Adam
  - Usually with a small learning rate
- This is necessary to make most pruning methods useful
- But fine-tuning is usually the dominant factor that affects performance! So it's hard to compare compression methods.

Knowledge distillation

- Idea: take a large/complex model and **train a smaller network to match its output**
  - Often used for distilling **ensemble models** into a single network
  - E.g. Hinton et. al. “Distilling the Knowledge in a Neural Network.”



$$\text{loss} = \text{CrossEntropy}(y_i, h_w(x_i)) + \gamma \cdot \text{CrossEntropy}_T(\text{Teacher}(x_i), h_w(x_i))$$

- Can also improve the accuracy in some cases.

Efficient architectures

- Some neural network architectures are **designed to be efficient at inference time**
  - Examples: MobileNet, ShuffleNet, CirCNN
- These networks are often based on sparsely connected neurons
  - This limits the number of weights which makes models smaller and easier to run inference on
- To be efficient, we can just **train one of these networks in the first place** for our application.

Re-use of computation

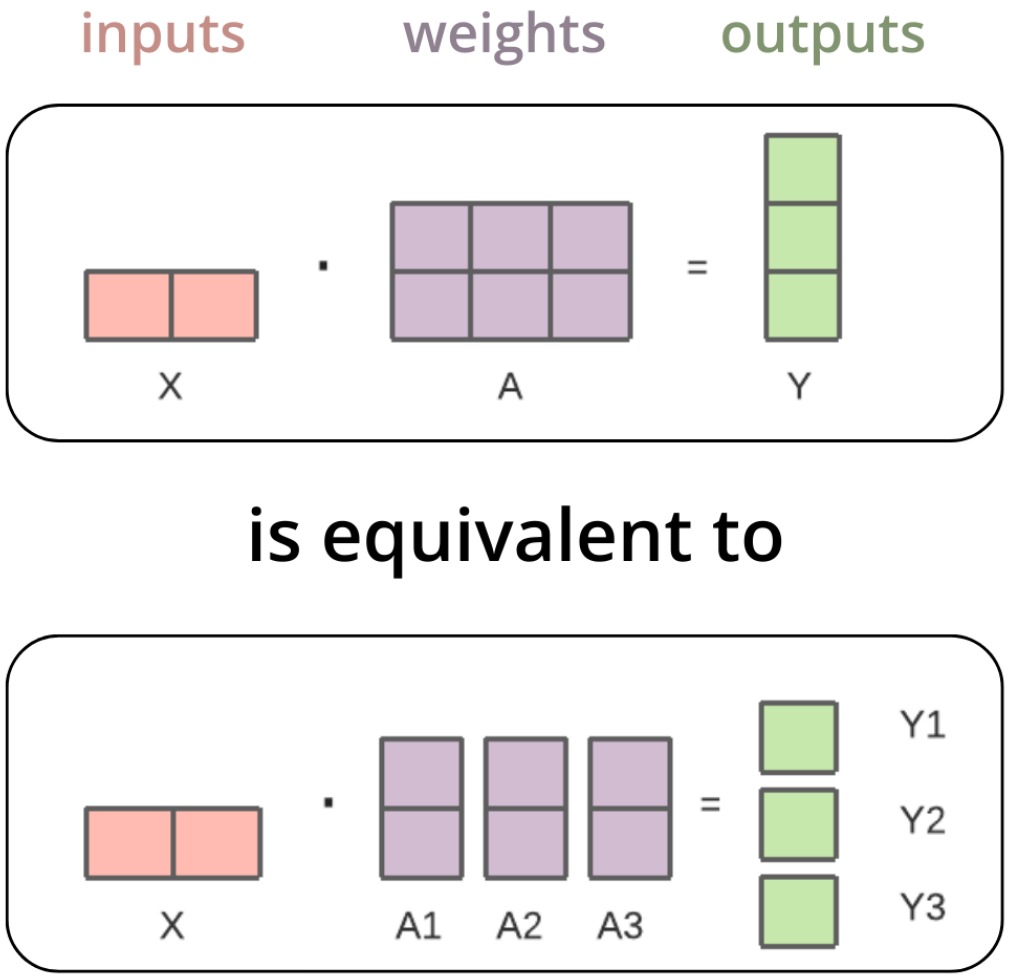
- For video and time-series data, there is a lot of **redundant information** from one frame to the next.
- We can try to **re-use some of the computation** from previous frames.
- This is less popular than some of the other approaches here, because it is not really general.
- But in LLMs it’s ubiquitous in the **key-value cache**

Model Parallelism

- Can **improve throughput** by separating the inference work among multiple parallel workers
  - Each worker is responsible for its own section of the network
  - Can help avoid loading weights from memory

- Often can **negatively impact latency**
  - Because activations need to be sent between machines

Tensor Parallelism

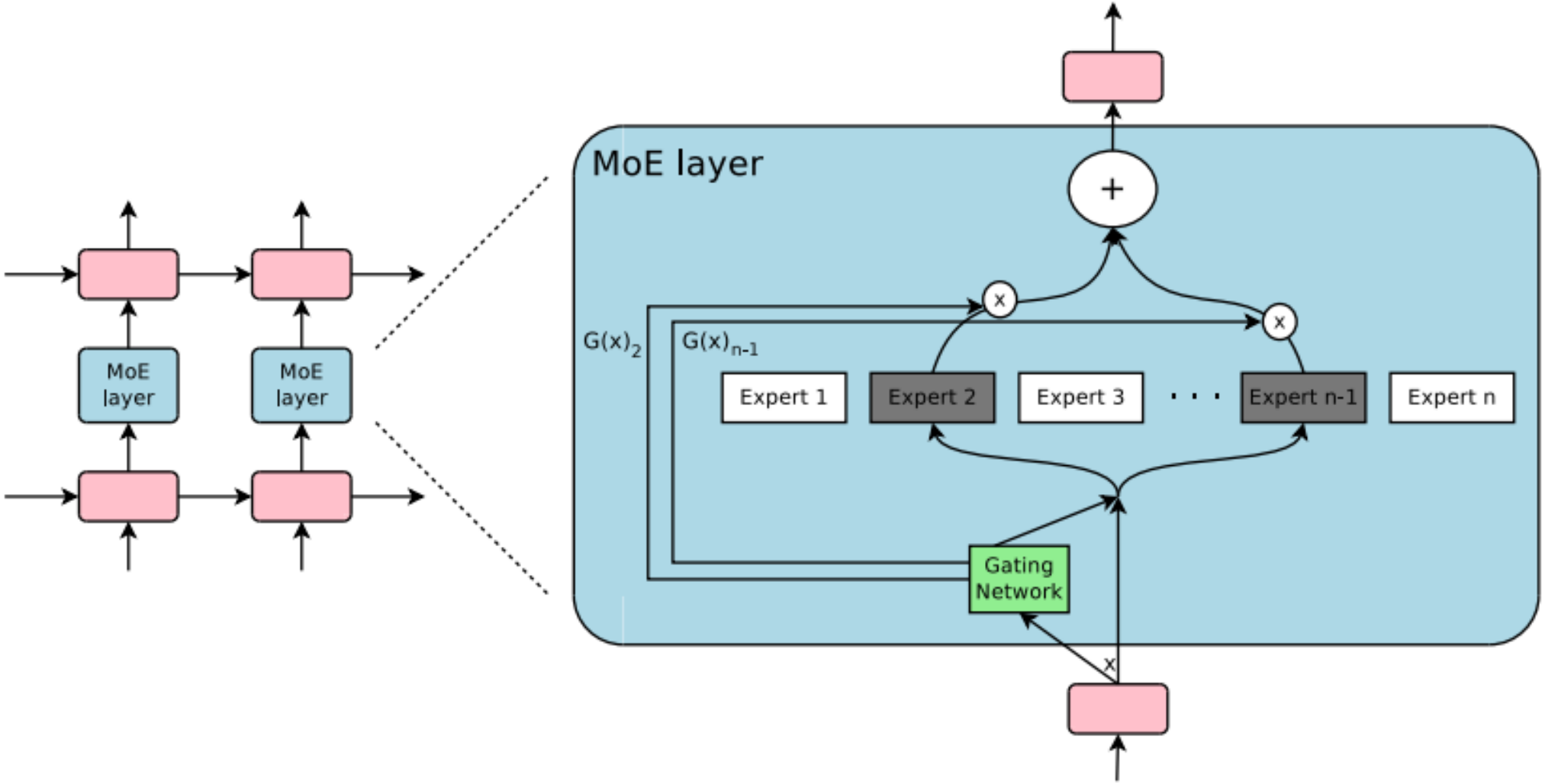


https://huggingface.co/docs/text-generation-inference/en/conceptual/tensor_parallelism

Tensor Parallelism on a Transformer

- In attention block, parallelize across attention heads
  - Output is summed over all heads
  - Requires an **all-reduce** after the block
- In MLP block, parallelize across intermediate size
  - Output is still summed across all GPUs
  - Requires an **all-reduce** after the block
- Can **decrease latency**, but limited by all-reduce time

Sparse Mixture of Experts (SMoE)



Speculative Decoding

- **“Guess” the output of a LLM** using a smaller model
  - Producing a block of **B** tokens
  - Smaller model called the “assistant” or the “draft” model
  - **Eagle-3** currently popular

- Run the LLM on the guessed sequence
  - Taking advantage of increased parallelism from batch size

- **Accept as many tokens as you can** from the guess while keeping distribution the same as the LLM

Just scale down

- Use a 7b model instead of a 70b model
- Benefits obvious

The last resort for speeding up DNN inference

- **Train another, faster type of model** that is not a deep neural network
  - For some real-time applications, you can't always use a DNN
- If you can get away with **a linear model**, that's almost always much faster.
- Also, **decision trees** tend to be quite fast for inference.

Where do we run inference?

Inference in the cloud

- Most inference today is run on **cloud platforms**
- The cloud supports **large amounts of compute**
  - And makes it easy to access it and make it reliable
- This is a good place to put AI that needs to think about data
- For interactive models, **latency** is critical

Inference on ML service providers

- **Many companies provide AI/ML services**
  - E.g. OpenAI
- These services can provide an ML inference stack that goes beyond just providing the hardware
  - E.g. they could charge by the token, rather than by the GPU-hour
- An interesting new business model

Inference on edge devices

- Inference can run on your **laptop or smartphone**
  - Here, the size of the model becomes more of an issue
  - Limited smartphone memory
- This is good for **user privacy and security**
  - But not as good for companies that want to keep their models private
- Also can be used to deploy **personalized models**

Inference on sensors

- Sometimes we want **inference right at the source**
  - On the sensor where data is collected
- Example: a surveillance camera taking video
  - Don’t want to stream the video to the cloud, especially if most of it is not interesting.
- **Energy use** is very important here.

Questions?