Inference, Deployment, and Compression

CS4787/5777 Lecture 26 — Fall 2023
Review: Inference

• Suppose that our training loss function looks like

\[ f(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(h_w(x_i); y_i) \]

• Inference is the problem of computing the prediction

\[ h_w(x_i) \]
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
Metrics for Inference

• Important metric: **accuracy**
  • Inference accuracy can be close to test accuracy — if data from same distribution
• 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
• Important metric: **cost**
  • How much money will all this cost us
Tradeoffs

- When designing an ML system for inference, there are trade-offs among all these metrics!
  - Most “techniques” do not give free improvements, but have some trade-off where some metrics get better and others get worst

- There is no one-size-fits-all “best” way to do ML inference.

- We need to decide which metric we value the most
  - Then keep that in mind as we design the system
Improving the performance of inference

What tools do we have in our toolbox?
Choosing our hardware: CPU vs GPU

• For training, people generally use GPUs for their high **throughput**

• But for inference, the right choice is less clear

• For small networks, CPUs can have the edge on **latency**
  • And CPUs are generally cheaper...lower **cost**

• CPU-like architectures are often a good choice for low-power systems, since it’s easier to put a low-power CPU on a mobile device
  • Many mobile chips are now CPU/GPU hybrids, so line is blurred here
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

Latency vs Throughput
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.
Neural Network Compression

• Find an easier-to-compute network with similar accuracy
  • Or find a network with smaller model size, depending on the goal

• Most compression methods are lossy, meaning that the compressed network may sometimes predict differently

• Many techniques for doing this
  • We’ll see some in the following slides
Simple Technique: “Old-School” Compression

• Just apply a standard lossless compression technique to the weights of your neural network.
  • **Huffman coding** works here, for example.
  • Even something very general like **gzip** can be beneficial.

• This **lowers the stored model size without affecting accuracy**

• But this does mean we **need to decompress eventually**, so it comes at the cost of some compute & can affect start-up latency.
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 16 bit numbers with this method alone

• **Binarization/ternarization** is low-precision arithmetic in the extreme
Pruning

- **Remove activations** that are usually zero
  - Or that don’t seem to be contributing much to the model
  - Good heuristic: remove the smallest X% of weights

- 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.
Fine-Tuning

• Powerful idea: apply a lossy compression operation, then **retrain the model** to improve accuracy

• A general way of “getting back” accuracy lost due to lossy compression.
Knowledge distillation

- Idea: take a large/complex model and train a smaller network to match its output
  - E.g. Hinton et. al. “Distilling the Knowledge in a Neural Network.”

- Often used for distilling ensemble models into a single network
  - Ensemble models average predictions from multiple independently-trained models into a single better prediction
  - Ensembles often win Kaggle competitions

- 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, SqueezeNet

- 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.
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**, it’s almost always much faster.

• Also, **decision trees** tend to be quite fast for inference.

• ...but with how technology is developing, we’re seeing more and more support for fast DNN inference, so this will become less necessary.
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 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.