Efficient Inference

Weizhe Yuan

10/23/2023

Overview

- **General Techniques**
  - Quantization
  - Pruning
  - Knowledge Distillation
- **Speculative Decoding** (specific to Transformer text generation)

Quantization

- **Idea**: Representing the weights and activations (output of the layer) with low-precision data types

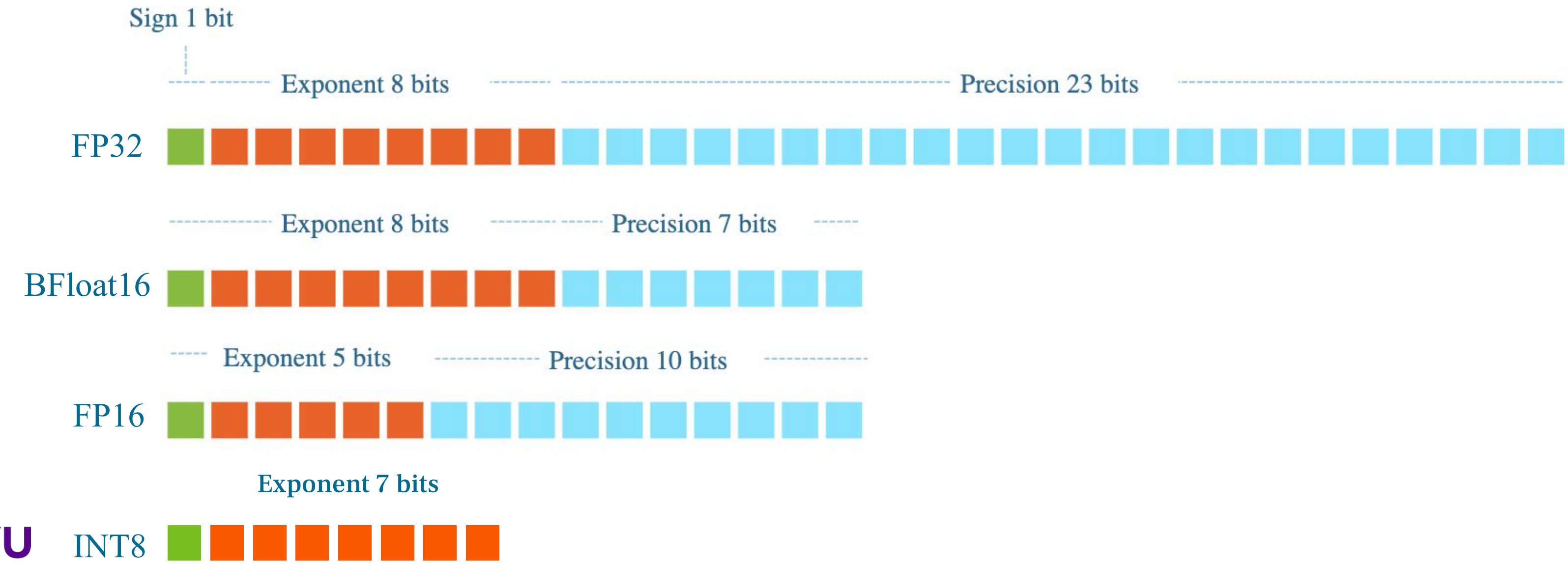


Image from: https://frankdenneman.nl/2022/07/26/training-vs-inference-numerical-precision/

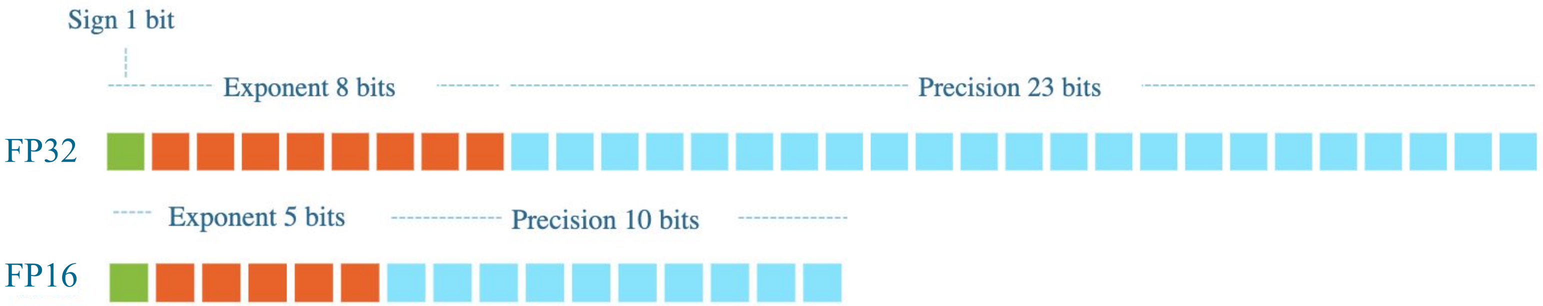
Quantization

- The two **most common** quantization cases are
  - float32 -> float16
  - float32 -> int8

NYU

Quantization

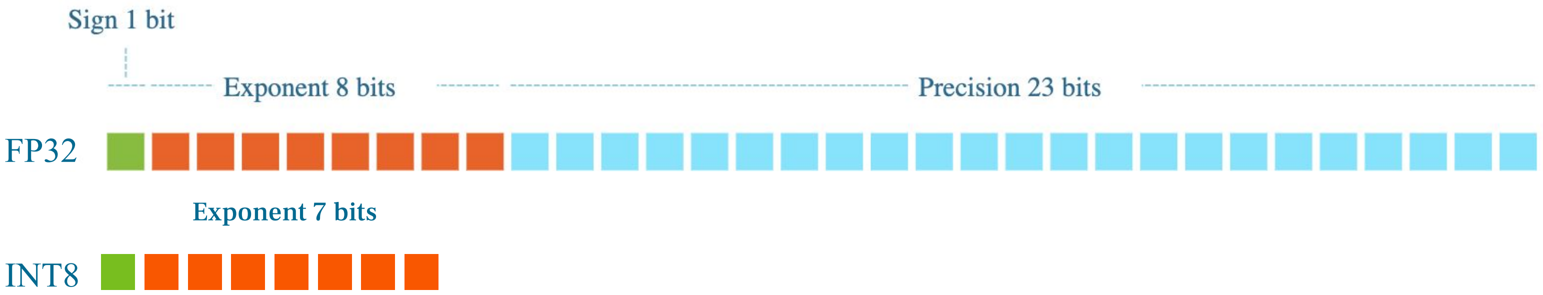
- **FP32 → FP16**
  - Performing quantization to go from float32 to float16 is quite straightforward since both data types follow the same representation scheme.
  - Need Clamping
  - Lose some precision



NYU

Quantization

- **FP32 → INT8**
  - More tricky
  - INT8 Range: [-127, +127]
  - Only 256 values can be represented in int8, while float32 can represent a very wide range of values.



Quantization

- **FP32 → INT8**
  - **Idea**: Find the best way to project our range [a, b] of float32 values to the int8 space.
  - Let's consider a float x in [a, b]

$$x_q = round(\frac{x}{S} + Z)$$

Quantized value of x

Scale

Zero Point

Quantization

- **FP32 → INT8**
  - **Idea**: Find the best way to project our range [a, b] of float32 values to the int8 space.
  - Let's consider a float x in [a, b]
  - Linear mapping: a is mapped to the smallest int (-127), b is mapped to the largest int (+127), so we can calculate **S** and **Z**
  - And float32 values outside of the [a, b] range are clipped to the closest representable value
  - **De-quantization**
    - $x = S * (x_q - Z)$

Quantization

- **Per-tensor Quantization**
  - Each tensor will have its own (S, Z) pair
- **Per-channel Quantization**
  - A pair of (S, Z) per element along one of the dimensions of a tensor.

Quantization

- **Quantized Matrix Multiplication**
  - Suppose we want to perform $Y = XW + b$, where $X \in \mathbb{R}^{m \times p}, W \in \mathbb{R}^{p \times n}, b \in \mathbb{R}^n$
    Resulting in $Y \in \mathbb{R}^{m \times n}$

$$Y_{i,j} = b_j + \sum_{k=1}^{p} X_{i,k} W_{k,j}$$

NYU

Quantization

- **Quantized Matrix Multiplication**
    - Here we apply the de-quantization equation.

De-quantization

$$
\begin{aligned}
&Y_{i,j} \\
&= b_j + \sum_{k=1}^{p} X_{i,k} W_{k,j} \\
&= \underline{s_b(b_{q,j} - z_b)} + \sum_{k=1}^{p} \underline{s_X(X_{q,i,k} - z_X)} \underline{s_W(W_{q,k,j} - z_W)} \\
&= s_b(b_{q,j} - z_b) + s_X s_W \sum_{k=1}^{p} (X_{q,i,k} - z_X)(W_{q,k,j} - z_W) \\
&= s_b(b_{q,j} - z_b) + s_X s_W \left[\left(\sum_{k=1}^{p} X_{q,i,k} W_{q,k,j} \right) - \left(z_W \sum_{k=1}^{p} X_{q,i,k} \right) - \left(z_X \sum_{k=1}^{p} W_{q,k,j} \right) + p z_X z_W \right] \\
&= s_Y (Y_{q,i,j} - z_Y)
\end{aligned}
$$

NYU

Quantization

- **Quantized Matrix Multiplication**
  - Therefore, we can get

$$
\begin{aligned}
Y_{q,i,j} &= z_Y + \frac{s_b}{s_Y}(b_{q,j} - z_b) \\
&+ \frac{s_X s_W}{s_Y}\left[\left(\sum_{k=1}^{p} \underline{X_{q,i,k} W_{q,k,j}}\right) - \left(z_W \sum_{k=1}^{p} X_{q,i,k}\right) - \left(z_X \sum_{k=1}^{p} W_{q,k,j}\right) + p z_X z_W\right]
\end{aligned}
$$

Integer Matrix Multiplication

NYU

Quantization

- **Quantized Matrix Multiplication**
  - If we have to do a sequence of floating point matrix multiplications

$$
\begin{aligned}
X_1 &= X_0 W_0 + b_0 \\
X_2 &= X_1 W_1 + b_1 \\
&\vdots \\
X_n &= X_n W_n + b_n
\end{aligned}
$$

Quantization

- **Quantized Matrix Multiplication**
  - If we have to do a sequence of floating point matrix multiplications
  - We could convert the math to the followings using quantized matrices.

$$X_{0,q} = f_q(X_0, s_{X_0}, z_{X_0})$$

Quantization

$$X_{1,q} = f_m(X_{0,q}, W_{0,q}, b_{0,q}, s_{X_0}, z_{X_0}, s_{W_0}, z_{W_0}, s_{b_0}, z_{b_0}, s_{X_1}, z_{X_1})$$
$$X_{2,q} = f_m(X_{1,q}, W_{1,q}, b_{1,q}, s_{X_1}, z_{X_1}, s_{W_1}, z_{W_1}, s_{b_1}, z_{b_1}, s_{X_2}, z_{X_2})$$
$$\vdots$$
$$X_{n,q} = f_m(X_{n-1,q}, W_{n-1,q}, b_{n-1,q}, s_{X_{n-1}}, z_{X_{n-1}}, s_{W_{n-1}}, z_{W_{n-1}}, s_{b_{n-1}}, z_{b_{n-1}}, s_{X_n}, z_{X_n})$$

Quantized matrix multiplication

$$X_n = f_d(X_{n,q}, s_{X_n}, z_{X_n})$$

De-quantization

Quantization

- **Final Question:** How is the range of [a, b] decided?
- **Recall**: Idea for quantization is representing the weights and activations (output of the layer) with low-precision data types

NYU

Quantization

- **Final Question:** How is the range of [a, b] decided?
  - Weights ⟶ **Easy**
  - Activations
    - Dynamic quantization
      - The range for each activation is computed *<u>on the fly at runtime</u>*.
    - Static quantization
      - The range is computed *<u>in advance</u>* by passing through representative data to estimate

NYU

Pruning

- **Idea**: Remove some neurons or connections in the network

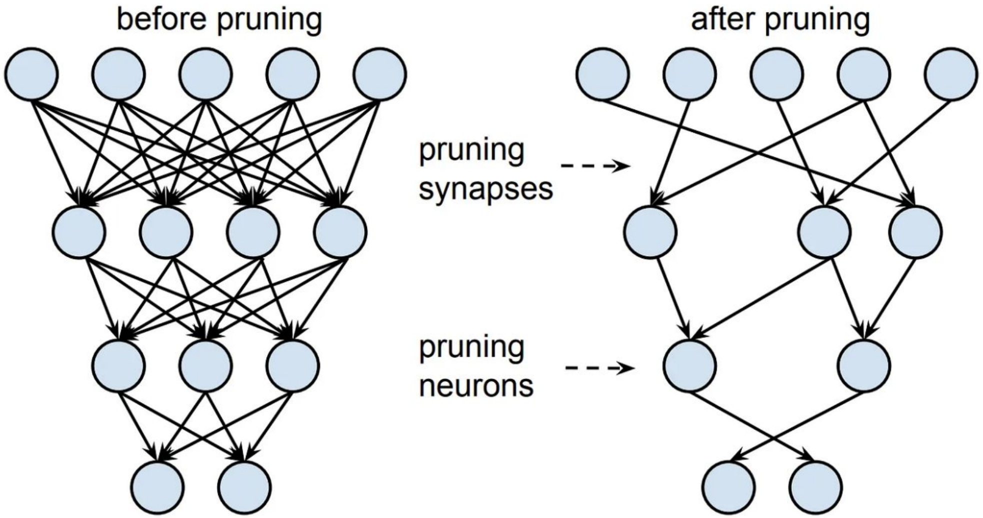


Image from https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html

Pruning

- **Weight Pruning**
- **Neuron Pruning**
- (Optional): Retrain the model to recover accuracy

Pruning

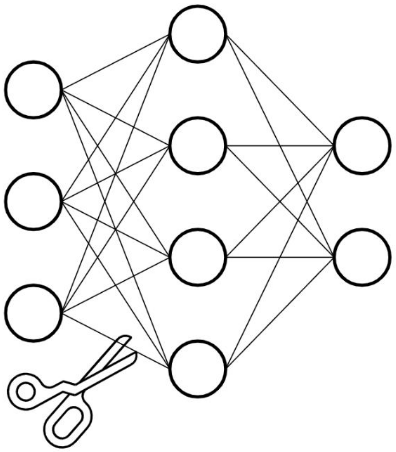
- **Weight Pruning**
  - In terms of the **Matrix Computation**
    - A lot of the values in the matrix get set to 0
      - Less Storage
      - Faster Compute

0	7	0	0	0	0	6
0	7	6	3	0	4	0
0	4	3	0	0	0	0
4	2	0	0	0	0	0
0	0	0	0	3	2	4

Image from https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html

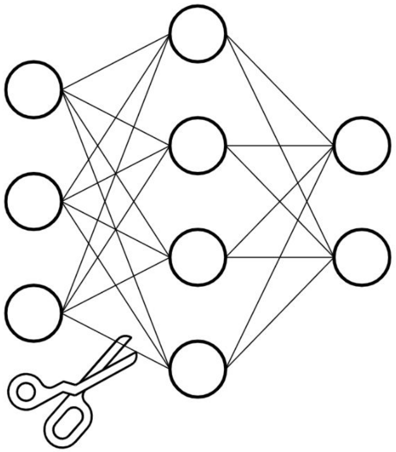
Pruning

- **Weight Pruning**
  - **Simplest method: Magnitude Pruning**
    - Pick pruning factor X
    - In each layer, set the lowest X% of weights (by absolute value) to zero



Pruning

- **Weight Pruning**
  - **Simplest method: Magnitude Pruning**
  - **Other methods**
    - Gradient-based Pruning: Prune weights that have consistently low gradients throughout training
    - Hessian-based Pruning
    - Etc…



NYU

Pruning

- **Weight Pruning**
  - **Unstructured Pruning**
    - Simply remove connections from a network without any further pattern
  - **Structured Pruning**
    - Enforce more structure on which weights are allowed to set to 0

Pruning

- **Weight Pruning**
  - **Unstructured Pruning**
  - **Structured Pruning**
    - Enforce more structure on which weights are allowed to set to 0
    - E.g., 2:4 Structured Sparsity Pattern

At least two 0

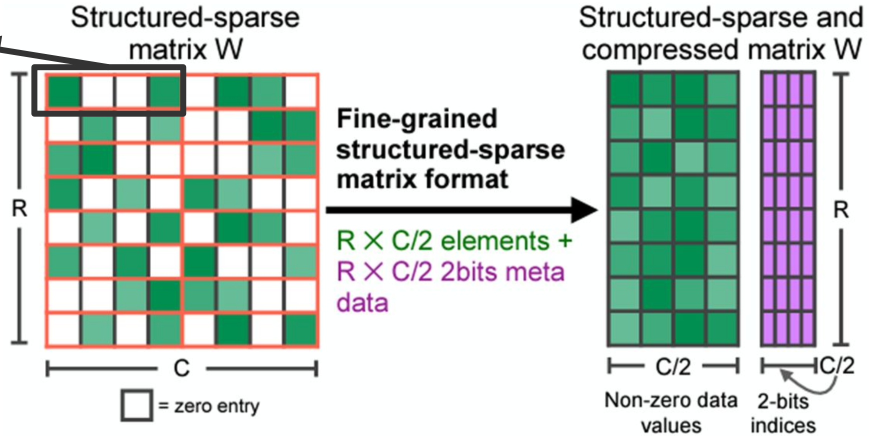


Image from: https://developer.nvidia.com/blog/structured-sparsity-in-the-nvidia-ampere-architecture-and-applications-in-search-engines/

Pruning

- **Weight Pruning**
  - **Unstructured Pruning**
  - **Structured Pruning**
    - Enforce more structure on which weights are allowed to set to 0
    - E.g., 2:4 Structured Sparsity Pattern
    - INVIDIA's tensor core GPUs are able to execute this type of structured sparsity with greater efficiency.

Image from: https://developer.nvidia.com/blog/structured-sparsity-in-the-nvidia-ampere-architecture-and-applications-in-search-engines/

Pruning

- **Weight Pruning**
  - **Hardware dependent**
    - Design pruning algorithms with the hardware in mind
    - Depend on what kind of sparsity runs fast on the hardware that you intend to deploy your neural network on

Pruning

- **Weight Pruning**
- **Neuron Pruning**
    - Run data through the network and observe the activations
        - Prune neurons that output near-zero values
        - Prune redundant neurons that have very similar weights or activations
        - Etc.

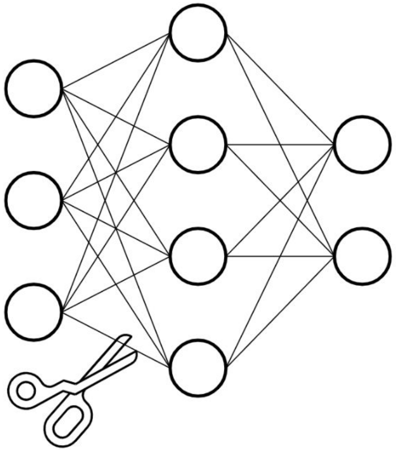


Image from https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html

NYU

Pruning

- **Weight Pruning**
- **Neuron Pruning**
  - Change the model architecture

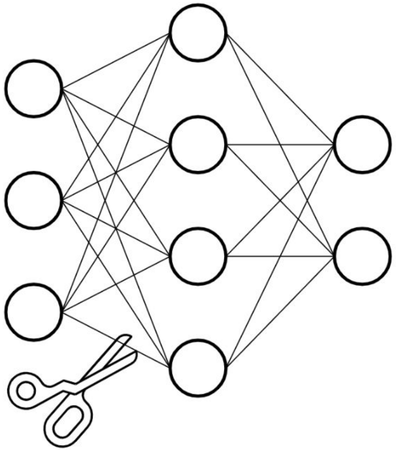


Image from https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html

NYU

Knowledge Distillation

- **Idea**: Transferring knowledge from a large model to a smaller one

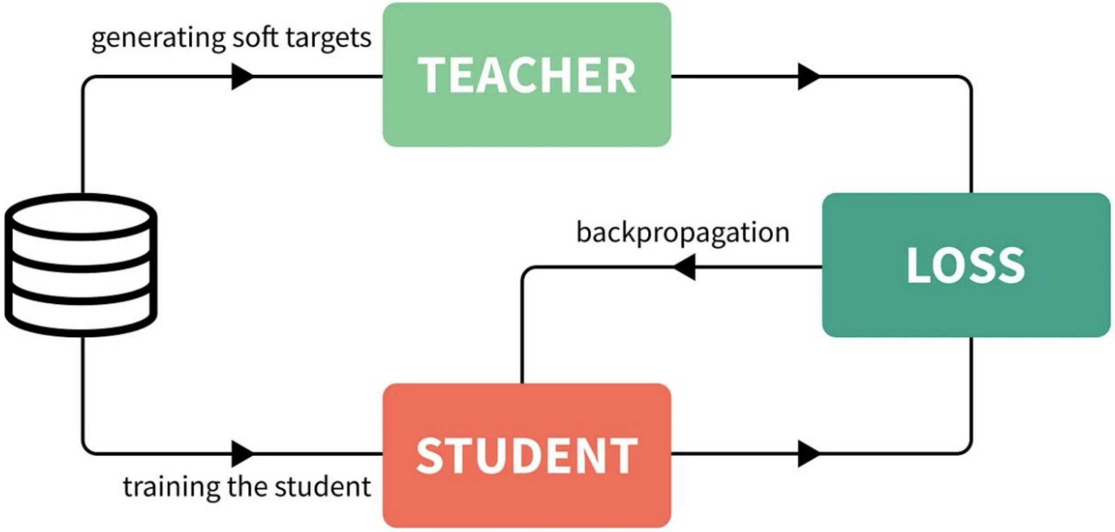


Image from: https://rmoklesur.medium.com/knowledge-distillation-in-deep-learning-keras-implementation-b61261c552db

Knowledge Distillation

- **General Recipe**
  - First, we use the training data to train a teacher network
  - Then, we start to train the student network to align its outputs to the outputs of the teacher network

Why don't we directly train the student network on the training data?

Knowledge Distillation

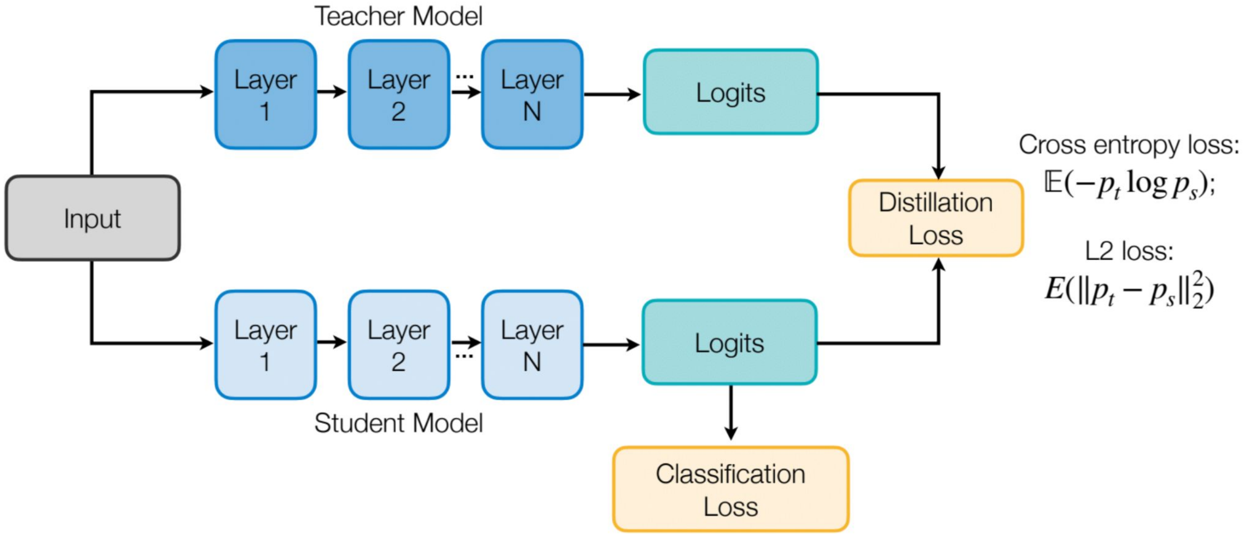
- **General Recipe**
  - First, we use the training data to train a teacher network
  - Then, we start to train the student network to align its outputs to the outputs of the teacher network
  - **Reasons**:
    - **Proven Fact**: small models are hard to train using the training data
      - Over-parameterization has become the de-facto, easier to train and generalize better

Knowledge Distillation

- **General Recipe**
  - First, we use the training data to train a teacher network
  - Then, we start to train the student network to align its outputs to the outputs of the teacher network
  - **Reasons**:
    - **Proven Fact**: small models are hard to train using the training data
    - Outputs from the teacher network contains much more information (distribution) than just a label

Knowledge Distillation

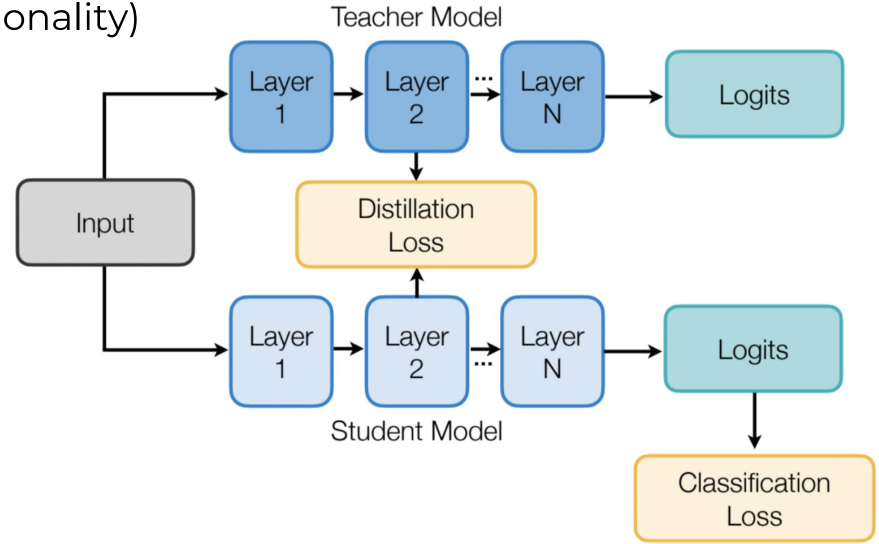
- **What to match?**
  - Simplest: **Output logits**



Images from:

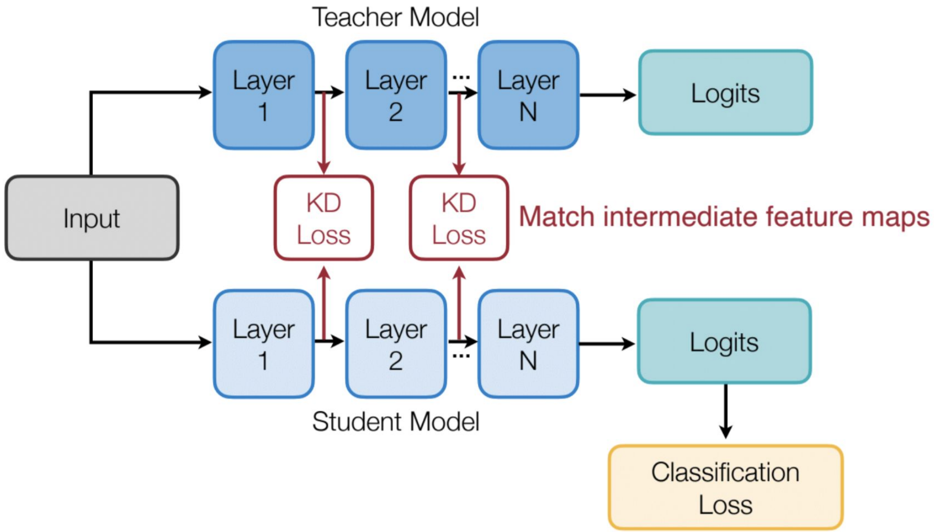
Knowledge Distillation

- **What to match?**
  - Simplest: **Output logits**
  - **Intermediate Weights** (Linear transformation is applied to match the dimensionality)



Knowledge Distillation

- **What to match?**
  - Simplest: **Output logits**
  - **Intermediate Weights**
  - **Intermediate Features** (outputs)



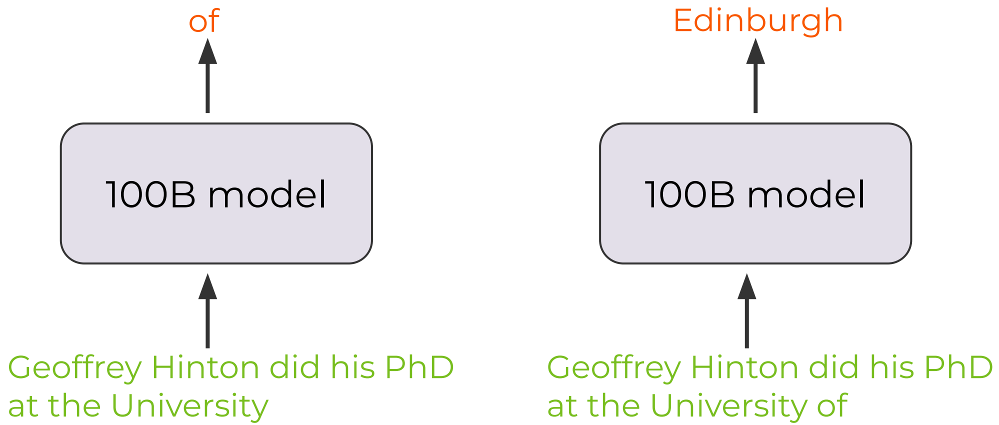
Knowledge Distillation

- **What to match?**
  - Simplest: **Output logits**
  - **Intermediate Weights**
  - **Intermediate Features** (outputs)
  - **Others**
    - Gradient
    - Sparsity Features
    - Etc…

NYU

Speculative Decoding

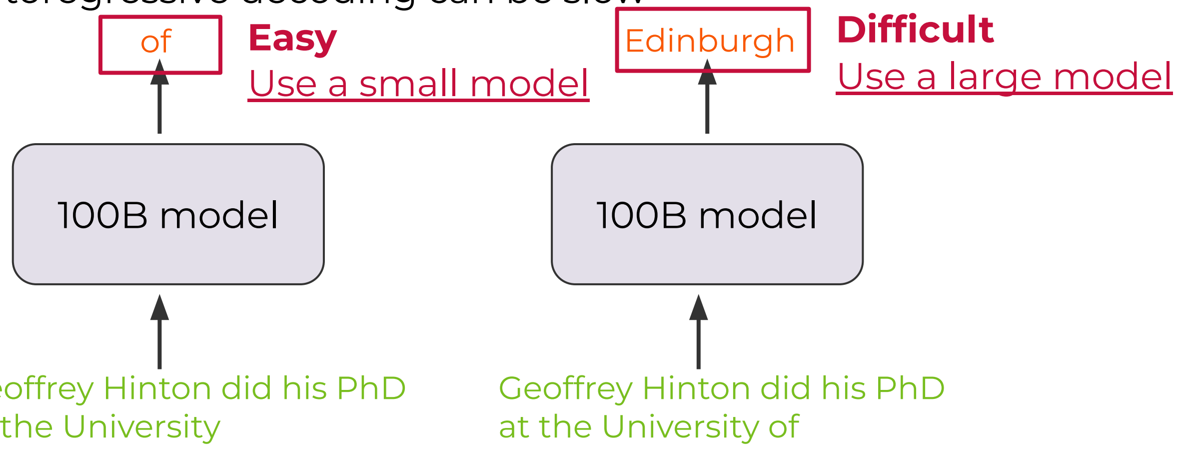
- **Specific to Transformer text generation model**
- **Background:**
  - Autoregressive decoding can be slow



Example from: https://www.youtube.com/watch?v=S-8yr_RibJ4

Speculative Decoding

- **Specific to Transformer text generation model**
- **Background:**
  - Autoregressive decoding can be slow



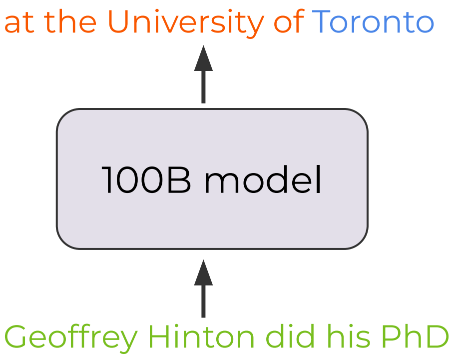
Example from: https://www.youtube.com/watch?v=S-8yr_RibJ4

Speculative Decoding

- **Specific to Transformer text generation model**
- **Background:**
  - Autoregressive decoding can be slow
- **Idea: Use two models**
  - The original large model
  - Another smaller draft model

Speculative Decoding

- **Key reason that this works** is related to the transformer model architecture
  - In one forward pass, it can generate probability distribution for multiple tokens in parallel



NYU

Speculative Decoding

- **Method**
  - **Notation**

M_p = draft model → Small

M_q = target model → Large

pf = prefix, K = 5 tokens

Speculative Decoding

- **Method**
    - **Step 1**: Run the draft model autoregressively **five times** to generate a sequence of 5 tokens

Probability distribution

$$p_1(x) = M_p(pf) \longrightarrow x_1$$

$$\boxed{p_2(x)} = M_p(pf, x_1) \longrightarrow \boxed{x_2}$$

...

$$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$$

Sampled token

Speculative Decoding

- **Method**
  - **Step 1**: Run the draft model autoregressively **five times** to generate a sequence of 5 tokens.
  - **Step 2**: Run the target model **once** to generate a probability distributions for five (+1) tokens in parallel.

$$q_1(x), q_2(x), q_3(x), q_4(x), q_5(x), q_6(x)$$

$$= M_q(pf, x_1, x_2, x_3, x_4, x_5)$$

Speculative Decoding

- **Method**
  - **Step 1**: Run the draft model autoregressively **five times** to generate a sequence of 5 tokens.
  - **Step 2**: Run the target model **once** to generate a probability distributions for five (+1) tokens in parallel.

Token	x1	x2	x3	x4	x5
	dogs	love	chasing	after	cars
p(x)	0.8	0.7	0.9	0.8	0.7
q(x)	0.9	0.8	0.8	0.3	0.8

Speculative Decoding

- **Method**
  - **Step 1**: Run the draft model autoregressively **five times** to generate a sequence of 5 tokens.
  - **Step 2**: Run the target model **once** to generate a probability distributions for five (+1) tokens in parallel.
  - **Step 3**: Decide which tokens to keep and return to Step 1

Speculative Decoding

- **Method**
  - **Step 3**: Decide which tokens to keep and return to Step 1

Token	x1	x2	x3	x4	x5
	dogs	love	chasing	after	cars
p(x)	0.8	0.7	0.9	0.8	0.7
q(x)	0.9	0.8	0.8	0.3	0.8

✅ ✅ ❌

Case 1: If $q(x) \geq p(x)$, then accept

Case 2: If $q(x) < p(x)$, then accept with probability $\frac{q(x)}{p(x)}$

Speculative Decoding

- **Method**
  - **Step 3**: Decide which tokens to keep and return to Step 1
  - Accepted token num: 0~5
    - Since we have run the large model once in each loop, we are at least as good as having only the large model
    - Worst case: we reject the first token and sample it from the large model's first token distribution
    - Best case: accept all the tokens

Speculative Decoding

- **Method**
  - **Theoretical Guarantee**
    - This method is equivalent to sampling from the original model q(x), so there is no loss of accuracy.
  - **Speedup**
    - Recommended value for K is 3-7
    - Typically 2-3x speedup

Questions?

NYU