Recap: Codesign

- **Codesign**: intrinsic development loop between algorithm design, training, and implementation
- Compression
  - Maintain high performance while removing redundant operations
- Quantization
  - Reduce precision from 32-bit floating point to 16-bit, 8-bit, ...
- Parallelization
  - Balance parallelization (how fast) with resources needed (how costly)
Recap: Quantization types

- Quantization: using reduced precision for parameters and operations
- Fixed-point precision
- Affine integer quantization

\[ \text{ap\_fixed<width,integer>} \]

0101.1011101010

integer
fractional

width

- \( \beta = -3 \)
- \( z \)
- \( \alpha = 4 \)

-128 0 17 127

\( \text{Expected AUC} = \text{AUC achieved by 32-bit floating point inference of the neural network} \)
Affine integer quantization

An affine mapping of integers to real numbers \( r = S(q - Z) \)

Floating-point range

Floating-point Scale

Integer

Floating-point

\( q_{\text{min}} \) \( Z \) \( q_{\text{max}} \)

\( r_{\text{min}} \) \( 0 \) \( r_{\text{max}} \)

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob et al., CVPR 2018]
## Post-training quantization vs. quantization-aware training

<table>
<thead>
<tr>
<th>PTQ</th>
<th>QAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usually fast</td>
<td>Slow</td>
</tr>
<tr>
<td>No re-training of the model</td>
<td>Model needs to be trained/finetuned</td>
</tr>
<tr>
<td>Plug and play of quantization schemes</td>
<td>Plug and play of quantization schemes (requires re-training)</td>
</tr>
<tr>
<td>Less control over final accuracy of the model</td>
<td>More control over final accuracy since $q$-params are learned during training.</td>
</tr>
</tbody>
</table>
Post-training quantization

- General strategy: avoid overflows in integer bit then scan the decimal bit until reaching optimal performance

```
0101.1011101010
```

```
0101.1011101010
```

\[ ap\text{\_}fixed<\text{width},\text{integer}> \]

FPGA AUC / Expected AUC

Full performance with 6 integer bits

Full performance with 8 fractional bits
Quantization-aware training: how does it work?

- Fake quantization: using 32-bit floating-point math under the hood
- Straight-through estimator: during backpropagation, ignore quantization operation (treat as identity)

\[
\begin{align*}
W_{fp32} &= (S_w, W_{i4})_{fp32} \\
a_{fp32} &= W_{fp32} h_{fp32} \\
a_{i4} &= \text{Int}\left(\frac{a_{fp32}}{S_a}\right)
\end{align*}
\]
Quantization-aware training

- Full performance with 6 bits instead of 14 bits
- Much smaller fraction of resources
- Area & power scale quadratically with bit width

Figure 2: Area (\(A\)) vs. Bitwidth

\[
\log_2(A) = k_2(n^2) + k_4(n^4) + \log_2(\text{constant})
\]

This quadratic fit explains the PE area with high accuracy.

Full performance with 6 bits instead of 14 bits results in a much smaller fraction of resources and scales quadratically with bit width.

Figure 1: Accuracy ratio vs. Bitwidth

\[
\frac{\text{Baseline Accuracy}}{\text{Quantized Accuracy}} = \frac{\text{Ratio Model Accuracy}}{\text{Baseline Accuracy}}
\]

This ratio shows the accuracy degradation due to quantization, with a maximum degradation at 15 bits.

Graph 1 and Graph 2 demonstrate the impact of quantization on hardware resources and communication complexity.
Pruning + quantization-aware training

- Quantization-aware pruning (QAP): iterative pruning can further reduce the hardware computational complexity of a quantized model.

- After QAP, the 6-bit, 80% pruned model achieves a factor of 50 reduction in BOPs compared to the 32-bit, unpruned model.

- Study using Brevitas

Bit operations (BOPs) definition:

arXiv:1804.10969

arXiv:2102.11289
Hessian-aware quantization (HAWQ) arXiv:2011.10680

- Hessian of loss can provide additional guidance about quantization!
- Flat loss landscape: Lower bit width
- Sharp loss landscape: Higher bit width
Recap: Pruning and quantization

- Pruning and quantization can be used post-training to compress models.
- They can also be used more effectively during training to achieve even higher levels of compression.

But so far we haven’t touch the model architecture?
- Are there compression schemes that do that?
- Yes, knowledge distillation!
Challenge: tiny models are hard to train

Tiny models underfit large datasets

Question: Can we help the training of tiny models with large models?
Distilling the Knowledge in a Neural Network

Geoffrey Hinton∗†
Google Inc.
Mountain View
dehoffhinton@google.com
Oriol Vinyals†
Google Inc.
Mountain View
vinyals@google.com
Jeff Dean
Google Inc.
Mountain View
jeff@google.com

Abstract

A very simple way to improve the performance of almost any machine learning algorithm is to train many different models on the same data and then to average their predictions [3]. Unfortunately, making predictions using a whole ensemble of models is cumbersome and may be too computationally expensive to allow deployment to a large number of users, especially if the individual models are large neural nets. Caruana and his collaborators [1] have shown that it is possible to compress the knowledge in an ensemble into a single model which is much easier to deploy and we develop this approach further using a different compression technique. We achieve some surprising results on MNIST and we show that we can significantly improve the acoustic model of a heavily used commercial system by distilling the knowledge in an ensemble of models into a single model. We also introduce a new type of ensemble composed of one or more full models and many specialist models which learn to distinguish fine-grained classes that the full models confuse. Unlike a mixture of experts, these specialist models can be trained rapidly and in parallel.
Illustration of KD

Knowledge Distillation: A Survey [Gou et al., IJCV 2020]

Teacher Network

Student Network

Input

Logits

Distillation Loss

Classification Loss

Pruning neurons

Pruning synapses

Before pruning

After pruning
Illustration of KD

Matching prediction probabilities between teacher and student

Teacher Network

Student Network

<table>
<thead>
<tr>
<th></th>
<th>Logits</th>
<th>Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cat</td>
<td>5</td>
<td>0.982</td>
</tr>
<tr>
<td>Dog</td>
<td>1</td>
<td>0.017</td>
</tr>
</tbody>
</table>

\[
\text{exp}(5) \quad \frac{\text{exp}(5)}{\text{exp}(5) + \text{exp}(1)}
\]

The student model is less confident

<table>
<thead>
<tr>
<th></th>
<th>Logits</th>
<th>Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cat</td>
<td>3</td>
<td>0.731</td>
</tr>
<tr>
<td>Dog</td>
<td>2</td>
<td>0.269</td>
</tr>
</tbody>
</table>

\[
\text{exp}(1) \quad \frac{\text{exp}(1)}{\text{exp}(5) + \text{exp}(1)}
\]
Illustration of KD

Matching prediction probabilities between teacher and student

<table>
<thead>
<tr>
<th></th>
<th>Logits</th>
<th>Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cat</td>
<td>5</td>
<td>0.982</td>
</tr>
<tr>
<td>Dog</td>
<td>1</td>
<td>0.017</td>
</tr>
</tbody>
</table>

Teacher Network

Student Network

Logits

Probabilities

Cat

Dog

3

2

0.731

0.269
Illustration of KD

Concept of temperature

A larger temperature smooths the output probability distribution.
Formal definition of KD

• Neural networks typically use a softmax function to generate the logits $z_i$ to class probabilities
  \[ p(z_i, T) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \]. Here, $i, j = 0, 1, 2, \ldots, C - 1$, where $C$ is the number of classes. $T$ is the temperature, which is normally set to 1.

• The goal of knowledge distillation is to align the class probability distributions from teacher and student networks.
When there is a possibility of confusion, we will refer to the student's training data as the weighted combination of two objectives, that respectively achieve 55%, 75%, and 95% test accuracy.

3.2 Metrics and Evaluation

Further discussion on the interplay of teacher ensemble size, teacher network capacity, and distillation the full teacher distribution by choosing a moderate value (e.g. in turn determines the allocation of student capacity. If the student is much smaller than the teacher, the added knowledge distillation term that encourages the student to match the teacher. It is the teacher is an scaled by a temperature hyperparameter $\tau$.

<table>
<thead>
<tr>
<th>Average Predictive KL</th>
<th>Average Top-1 Agreement</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_{\text{NLL}}(z_s, y)$ := $-\sum_{j=1}^{c} y_j \log \sigma_j(z_s)$</td>
<td>$L_{\text{KL}}(z_s, z_t)$ := $-\tau^2 \sum_{j=1}^{c} \sigma_j \left(\frac{z_t}{\tau}\right) \log \sigma_j \left(\frac{z_s}{\tau}\right)$</td>
</tr>
</tbody>
</table>

While improvements in generalization metrics are relatively easy to understand, interpreting fidelity also evaluate another student points, whereas metrics requires some care. For example, suppose we have three independent models: Eqn. $L_{\text{NLL}}(z_s, y)$ := $-\sum_{j=1}^{c} y_j \log \sigma_j(z_s)$, $L_{\text{KL}}(z_s, z_t)$ := $-\tau^2 \sum_{j=1}^{c} \sigma_j \left(\frac{z_t}{\tau}\right) \log \sigma_j \left(\frac{z_s}{\tau}\right)$.

3.1 Knowledge Distillation

To account for such confounding when evaluating the distillation of a student also consider deploying the matrix and tensor decomposition (Allen-Zhu et al., 2018) and parameter sharing (Arora et al., 2016), structural mat-
Next time

• Guest lecture!