Lecture 10:
Raising the level of abstraction for ML

Parallel Computing
Stanford CS348K, Spring 2021
Note

- Most of this class involved in-class discussion of the Ludwig and Overton papers

- I am posting these slides as some were used during parts of the discussion
Services provided by ML “frameworks”

- **Functionality:**
  - Implementations of wide range of useful operators
    - Conv, dilated conv, relu, softmax, pooling, separable conv, etc.
  - Implementations of various optimizers:
    - Basic SGD, with momentum, Adagrad, etc.
  - Ability to compose operators into large graphs to create models
  - Carry out back-propagation

- **Performance:**
  - High performance implementation of operators (layer types)
  - Scheduling onto multiple GPUs, parallel CPUs (and sometimes multiple machines)
  - Automatic sparsification and pruning

- **Meta-optimization:**
  - Hyper-parameter search
  - More recently: neural architecture search
TensorFlow/MX.Net data-flow graphs

- Key abstraction: a program is a DAG of (large granularity) operations that consume and product N-D tensors
Modular network designs

Inception v4

**Stem**

- Input (299x299x3) 299x299x3
- 4 x Inception-A

**Reduction-A**

- Output: 35x35x384
- Output: 17x17x1024

**Reduction-B**

- Output: 8x8x1536
- Filter concat

**Inception-C**

- 3 x Inception-C

**A block**

- Avg Pooling (96)
- 1x1 Conv (96)
- 3x3 Conv (96)

**B block**

- Avg Pooling
- 1x1 Conv (64)
- 3x3 Conv (96)

**Softmax**

- Output: 1000
- Dropout (keep 0.8)
- Avarage Pooling

Different variants of this blocks (with various number of filters) are used in Figure 9, and 15 in each of the new Inception(-v4, -ResNet-v1, -ResNet-v2) variants presented in this paper. The \(k\), \(l\), \(m\), \(n\) numbers represent filter bank sizes which can be looked up in Table 1.
Inception stem

Historically, we have been relatively conservative about changing the architectural choices and restricted our experiments to varying isolated network components while keeping the rest of the network stable. Not simplifying earlier choices resulted in networks that looked more complicated than they needed to be. In our newer experiments, for Inception-v4 we decided to shed this unnecessary baggage and made uniform choices for the Inception blocks for each grid size. Please refer to Figure 9 for the large scale structure of the Inception-v4 network and Figures 3, 4, 5, 6, 7 and 8 for the detailed structure of its components.

All the convolutions not marked with "V" in the figures are same-padded meaning that their output grid matches the size of their input. Convolutions marked with "V" are valid padded, meaning that input patch of each unit is fully contained in the previous layer and the grid size of the output activation map is reduced accordingly.

3.2. Residual Inception Blocks

For the residual versions of the Inception networks, we use cheaper Inception blocks than the original Inception. Each Inception block is followed by filter-expansion layer (1x1 convolution without activation) which is used for scaling up the dimensionality of the filter bank before the addition to match the depth of the input. This is needed to compensate for the dimensionality reduction induced by the Inception block.

We tried several versions of the residual version of Inception. Only two of them are detailed here. The first one "Inception-ResNet-v1" roughly the computational cost of Inception-v3, while "Inception-ResNet-v2" matches the raw cost of the newly introduced Inception-v4 network. See Figure 15 for the large scale structure of both variants. (However, the step time of Inception-v4 proved to be significantly slower in practice, probably due to the larger number of layers.)

Another small technical difference between our residual and non-residual Inception variants is that in the case of Inception-ResNet, we used batch-normalization only on top of the traditional layers, but not on top of the summations. It is reasonable to expect that a thorough use of batch-normalization should be advantageous, but we wanted to keep each model replica trainable on a single GPU. It turned out that the memory footprint of layers with large activation size was consuming disproportionate amount of GPU-memory. By omitting the batch-normalization on top of those layers, we were able to increase the overall number of Inception blocks substantially. We hope that with better utilization of computing resources, making this trade-off will become unnecessary.
We have observed the degradation problem - the plain net has higher validation error than the shallower 18-layer plain nets. The 34-layer plain net is in Fig. 9. We evaluate both top-1 and top-5 error rates.

We consider two options: (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra options, when the shortcuts go across feature maps of two convolutional forms as in \[ \cdots \]

do not use dropout \[ \cdots \]

use a weight decay of 0.0001 and a momentum of 0.9. We use SGD with a mini-batch size of 256. The learning rate starts from 0.1 and is divided by 10 when the error plateaus, normalized (BN) \[ \cdots \] right after each convolution and for scale augmentation \[ \cdots \]. We initialize the weights of 224, 21, 14, 35, 384 \[ \cdots \] is used. We adopt batch normalization (BN) \[ \cdots \]

Figure 10. The schema for 35 × 35 grid (Inception-ResNet-A) module of Inception-ResNet-v1 network.
How to improve system support for ML?

Hardware/software for... faster inference? faster training?

Compilers for fusing layers, performing code optimizations?

List of papers at MLSys 2020 Conference
But as a user wanting to create a model, where does most of my time really go?
ML model development is an iterative process.

The figure illustrates the iterative process of ML model development, with boxes representing key actions performed during model development and arrows showing the flow of data and actions. The process starts with defining the task, selecting data, generating supervision, training the model, and validating the model. 

- Define Task: Identifying the problem to be solved.
- Define Inputs: Gathering the necessary data.
- Data Selection: Choosing the most relevant data.
- Generate Supervision: Creating labels or annotations.
- Train Model: Creating a model based on the training data.
- Validate Model: Testing the model on unseen data.
- Define Task (Sec 6.1): Refining the initial problem description.
- Define Inputs (Sec 6.1): Adding or removing data sources.
- Data Selection (Sec 5): Adjusting the data used.
- Generate Supervision (Sec 4): Modifying the labels or annotations.
- Train Model (Sec 6.1, 6.2): Tuning the model architecture or training hyperparameters.
- Validate Model (Sec 5): Evaluating the model's performance.
- Identify Important Data (New Spec, Different Pre-Trained Inputs): Identifying new data that could improve the model.
- Refine Task (Different Training Points, Rare Examples, New Failure Modes): Adjusting the problem description.
- Increase Supervision (New Supervision Sources): Adding new labeled data.
- New Spec, Different Pre-Trained Inputs (Different Training Points, Rare Examples, New Failure Modes): Revising the problem specification or using different pre-trained models.
Example: does TensorFlow help with data curation?

“We cannot stress strongly enough the importance of good training data for this segmentation task: choosing a wide enough variety of poses, discarding poor training images, cleaning up inaccurate [ground truth] polygon masks, etc. With each improvement we made over a 9-month period in our training data, we observed the quality of our defocused portraits to improve commensurately.”

Synthetic Depth-of-Field with a Single-Camera Mobile Phone

NEAL WADHWA, RAHUL GARG, DAVID E. JACOBS, BRYAN E. FELDMAN, NORI KANAZAWA, ROBERT CARROLL, YAIR MOVSHOVITZ-ATTIAS, JONATHAN T. BARRON, YAEL PRITCH, and MARC LEVOY, Google Research

(a) Input image with detected face  (b) Person segmentation mask  (c) Mask + disparity from DP  (d) Our output synthetic shallow depth-of-field image
Thought experiment: I ask you to train a car or person detector for a specific intersection.
- A good system provides valuable services to the user.
- So in the Ludwig/Overton papers, who is the “user” (what is their goal, what is their skillset?) and what are the painful, hard, or tedious things that the systems are designed to do for the user?
Let’s specifically contrast the abstractions of Ludwig with that of a lower-level ML system like TensorFlow. TensorFlow/MX.Net/PyTorch largely abstract ML model definition as a DAG of N-Tensor operations. How is Ludwig different?

Then let’s compare those abstractions to Overton.
Comparison to Google’s AutoML?