CprE 487/587

Homework 1: Machine Learning Introduction

1 Required Questions

1. Identify an application of deep learning you find interesting/motivating. Answer the following questions:
    a. Why do you find it interesting/motivating?

An application of deep learning that I find interesting is autonomous vehicles. I find this application interesting since it is very complicated, and there are a lot of factors for the computer to take into account, such as pedestrians, traffic, signs, etc.

    b. Identify possible datasets that are available (both public and private) relating to this application.

A public dataset that is available for autonomous vehicles is the Waymo dataset (formerly known as the Google Self-Driving Car Project. An example of a private dataset used for autonomous driving is Tesla's autonomous vehicle dataset.

    c. What is the current state-of-the-art model? What is its accuracy? Provide the model details if available (# layers, types/sizes of each layer, any distinguishing characteristics). Categorize the layer connectivity by sparsity, weights sharing, receptive field, and recurrent vs feed-forward.

A current state-of-the-art model used for autonomous vehicles is the BEVFusion model. This model has a mean average precision of around 71.7%, this model has three main components that they call the backbone for extracting features from input data, the neck for fusing features from different sensors, and the head for generating outputs. The exact number of layers depends on the implementation of BEVFusion, but BEVFusion usually uses a CNN with dozens of layers. BEVFusion has convolutional layers, which are applied to the backbone and fusion neck, pooling layers which are used for down-sampling features in the backbone, multi-sensor fusion layers used for merging LiDAR and camera features. Fully connected layers which are primarily in the head for classification and regression tasks, normalization layers to stabilize training. Sparsity in this model is dense, especially in post-feature extraction. Weight sharing is common in the backbone portion, the receptive field grows the network depth, and there are no recurrent layers, only feed-forward layers.

2. Given a simple neural network in Figure 1, calculate the final weights updated arrays for a single back-propagation update using a batch size of 1 (i.e., the provided input). The initial weight and bias values are shown below, where wLi and bLi denote the weights and biases between layers i and i + 1, respectively. For example, between the input and the hidden layer, we have a weight array of two by three and a bias array of three elements. An element at index i, j in the weight array denotes the value of the weight between the ith neuron of the previous layer and the jth neuron of the next layer. Similarly, an element at index of i in the bias array denotes the bias value added at the ith neuron of the output layer. All activation functions are assumed to be Sigmoid (see Section 3 for more information). Use a learning rate as 0.01, sample input as [0.1, 0.7], and expected (i.e., true) output class of input as [0, 1.0], for cross-entropy loss function. Please show your derivation of updated terms in your homework document. For the actual calculations, I suggest you don't do manual computations, but rather use a Jupyter notebook or equivalent – turn the code and a pdf in as your work.

$$w^{L_0} = \begin{bmatrix} 0.1 & 0.3 & 0.4 \\ 0.2 & 0.2 & 0.1 \end{bmatrix}$$

$$b^{L_0} = \begin{bmatrix} 0.31416 \\ -0.27182 \\ 0.186282 \end{bmatrix}$$

$$w^{L_1} = \begin{bmatrix} 0.3 & 0.1 \\ 0.5 & 0.6 \\ 0.2 & 0.7 \end{bmatrix}$$

$$b^{L_1} = \begin{bmatrix} 1.6180 \\ 0.1729 \end{bmatrix}$$

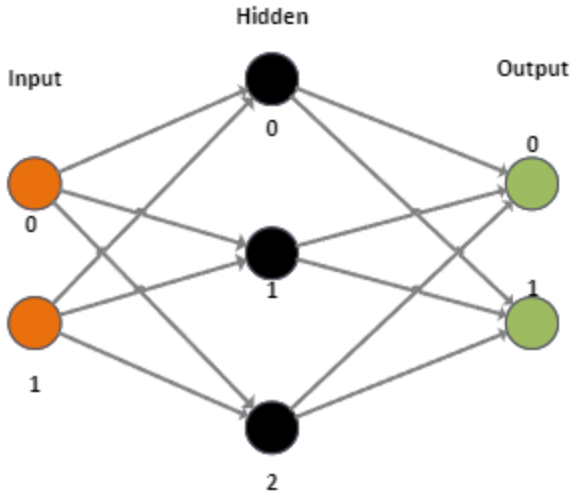


Figure 1: A Neural Network of Interest.

Feedforward Pass:

Input to hidden layer

Input: $z^{[0]} = w^{[0]} \cdot x + b^{[0]}$

Activation: $a^{[0]} = \sigma(z^{[0]})$

output to hidden layer

output layer input: $z^{[1]} = w^{[1]} \cdot a^{[0]} + b^{[1]}$

Activation: $a^{[1]} = \sigma(z^{[1]})$

$L = -[y \cdot \log(a^{[1]}) + (1-y) \cdot \log(1-a^{[1]})]$ ← loss function

$\delta^{[0]} = (w^{[1]} \cdot \delta^{[1]}) \cdot \sigma'(z^{[0]})$ ← Error

$\nabla w^{[0]} = \delta^{[0]} \cdot (x)^T$ > Gradient of loss

$\nabla b^{[0]} = \delta^{[0]}$

$w^{[1]}_{new} = w^{[1]} - \eta \cdot \nabla w^{[1]}$

$b^{[1]}_{new} = b^{[1]} - \eta \cdot \nabla b^{[1]}$

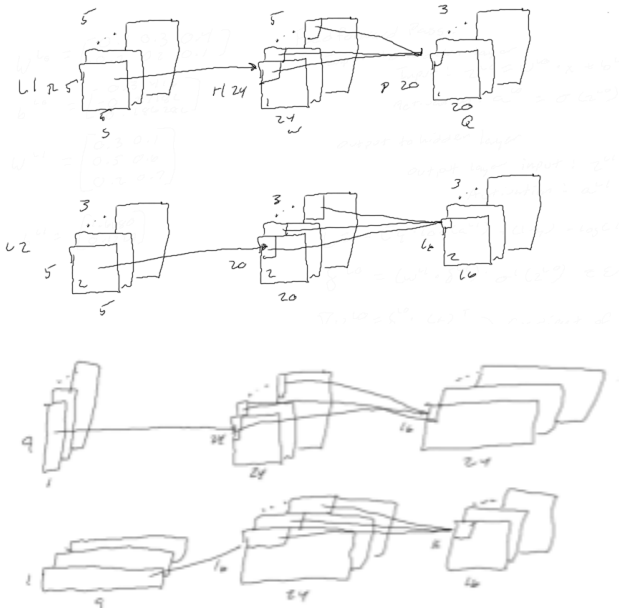
$w^{[0]}_{new} = w^{[0]} - \eta \cdot \nabla w^{[0]}$

$b^{[0]}_{new} = b^{[0]} - \eta \cdot \nabla b^{[0]}$

} Weight updates

```
Loss: 2.599206094006428
Updated wL0: [[0.09994304 0.2999309  0.40000494]
 [0.19960128 0.1995163  0.10003456]]
Updated bL0: [ 0.3135904  -0.272511    0.18633137]
Updated wL1: [[0.29449819 0.101749  ]
 [0.4957476  0.60135182]
 [0.1948608  0.70163373]]
Updated bL1: [1.6090394  0.17574853]
```

3. Consider the approach VGG-16 and GoogLeNet v3/v4 takes to construct larger filters from smaller filters (see Fig 2.9 from your textbook). Sketch out at least three different approaches you could take to construct a 9x9 spatial receptive field using multiple convolution layers (i.e., the baseline filter would have R = S = 9). Show the number of layers in each, the sizes of filters, and illustrate this in a similar manner to Figure 2.2(b) from the textbook. For this purpose assume there are no non-linear activation functions. Assuming I had a 24 x 24 input feature map, 5 input channels, and 3 output channels, what would be my total model size (in bytes assuming we only use 32-bit floats) for each of these approaches?

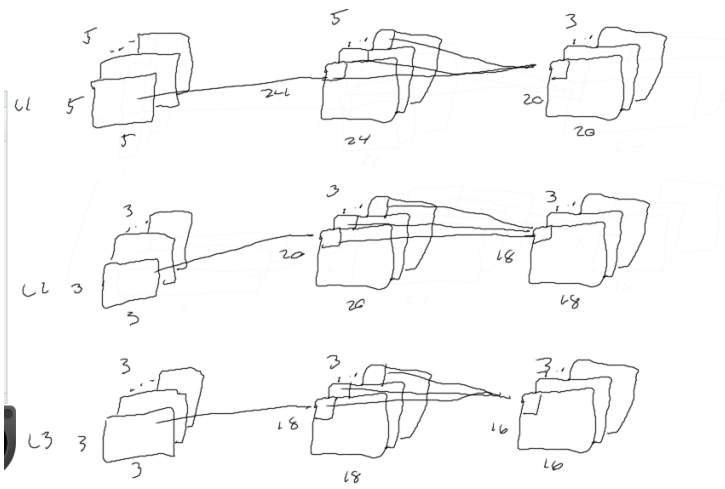


model size: (5×5×5) + (24×24×5) + (20×20×3) + (5×5×3) + (16×16×3) = 5048

5048 × 4 = 20,192 bytes

model size: (9×9×5) + (24×24×5) + (16×24×3) + (1×1×3) + (16×16×3) = 4,572

4572 × 4 = 18,288 bytes



4. Identify at least 5 different device types (e.g., AMD GPU, Intel CPU) which Tensorflow can support. For each of these devices, identify the sequence of lower steps a neural network must go through before it can be run. Which devices and their corresponding backends support training? Describe the advantages of using ahead-of-time (AOT) compilation versus just-in-time (JIT) compilation in Tensorflow. Under what situations would I use one versus the other?

   Tensorflow can support, NVIDIA GPUs, AMD GPUs, Intel CPUs, Google TPUs, and the Apple M1/M2 GPU. For NVIDIA's GPUs (CUDA backend), the sequence of steps is as follows, the neural network operations are converted to computational graphs, Tensorflow maps of high-level operations are sent to the CUDA kernels, the computational graph is then optimized for GPU execution, data and operations are then sent to the GPU memory and CUDA cores for processing, and finally the results are transferred back to the main memory. For AMD GPU's (ROCm backend, Tensorflow first uses the ROCm backend to map operations to AMD hardware, then neural network operations are translated to the Heterogeneous-Compute Interface for Portability (HIP) kernels, then optimizations are applied, and finally the optimized graph is executed on AMD GPU cores. For Intel CPUs (MKL-DNN backend), Tensorflow uses the Math Kernel Library to optimize neural networks, then MKL-DNN maps high-level operations to optimized CPU instructions, and Tensor operations run in parallel across multiple cores using vectorization, finally, results are calculated and stored in either the CPU cache or memory. For Google TPUs (XLA backend), Tensorflow converts operations to

XLA operations, then the computational graph is optimized and compiled to run on TPU hardware, the graph is then converted into TPU instructions that map directly to TPU cores. Finally, data flows between the host machine and TPU cores for processing. Apple's M1/M2 GPUs (Metal backend), Tensorflow is first translated to use Metal Performance Shaders, then the neural network is optimized for the M1/M2 GPU architecture, then Metal's API provides efficient access to the GPU for parallel processing, finally, data and computation operations are dispatched to Apple GPU cores.

AOT compilation has a faster execution time since the model is already compiled into an optimized format, has predictable performance, and is suitable for deployment on resource-constrained hardware. AOT is best used when models are deployed in production environments where inference speed is critical.

JIT compilation offers flexibility to adapt to dynamic models where the computational graph might change during execution, optimizations can be made based on actual data found during execution which can improve performance on some data patterns. JIT is best used when working with dynamic neural networks or when experimenting with different model structures, as well as in research where model structures frequently change.

5. Create your own assessment question. Given what you have learned through lecture, the textbook, supplemental materials, and your own investigation, come up with your own homework/quiz question and corresponding solution (will be shared with the class). Make sure that it is not simply a "recall" type question.

Question: In a CNN, an input image of size 32x32x3 is passed through a convolutional layer with 16 filters of size 5x5, and a stride size of 1 with no padding applied. Calculate the size of the output f-map after this convolutional layer. If the next layer uses the ReLU function followed by max pooling with a 2x2 filter and a stride of 2, what will be the size of the final f-map after the max pooling operation? Finally, explain the role of the number of filters in this process and how it impacts the f-map.

Solution:

Input Image Size : 32x32x3

Filter Size: 5x5

Stride: 1

No padding

Formula: Output size = $\frac{\text{Input Size} - \text{Filter Size}}{\text{stride}} + 1$

Output Height = $\frac{32-5}{1} + 1 = 28$

Output Width = $\frac{32-5}{1} + 1 = 28$

Since there are 16 filters, the depth of the output f-map is 16.

So the size of the output f-map is 28x28x16.

F-map after max pooling:

Height = $28/2 = 14$

Width = $28/2 = 14$

Size: $14 \times 14 \times 16$

The number of filters determines the depth of the output f-map. Each filter is responsible for detecting different features from the input image. Increasing the number of filters allows the network to learn more complex feature representations, but the computational cost is also

increased. So the number of filters directly impacts the feature extraction as well as the dimensionality of the f-maps.