CMSC 25500: Introduction to Neural Networks

The University of Chicago, Autumn 2021

Midterm

There is no time limit on this midterm and you may refer to class notes, lectures, and external resources. However, you may not collaborate with others on the problems.

By submitting this midterm you are implicitly confirming that:

- You did not discuss the midterm content with anyone else.
- You will refrain from communicating with anyone about the midterm content until after the midterm submission deadline.

For any questions regarding clarifications on the midterm material, please start a private thread on Piazza or ask during office hours.

This midterm consists of 6 problems. In total, all problems are worth 100 points.

Problem 1. Convolutional Neural Networks *[20 points total]*

Consider a CNN with the following layer structure:

Layer	Tensor Dimensions	Parameter Count	FLOPs
Input	$32 \times 32 \times 3$	0	0
Conv F=16, 5×5			
ReLU			
Conv F=24, 3×3			
ReLU			
Pool			
Conv F=32, 3×3			
ReLU			
Conv F=48, 3×3			
ReLU			
Pool			
Conv F=64, 3×3			
ReLU			
Conv F=64, 1×1			
ReLU			
Pool			
Conv F=96, 3×3			
ReLU			
Pool			
Conv F=128, 3×3			
ReLU			
Pool			

where convolutional layers are specified in terms of number of filters (F) and filter spatial extent. Assume all Conv layers use stride 1, and an amount of zero-padding that allows for placement of the filter center over each spatial location in their input. Pooling layers are 2×2 max pooling with stride 2.

(a) Complete the above table with the dimension of the activation tensor produced by each layer, the number of parameters in each layer, and the number of floating point operations (total of additions, multiplications, comparisons, or other scalar operations) that occur in a forward pass through the layer. Assume we are operating on a single example, so batch size is omitted here. *[12 points]*

(b) What is the shallowest layer for which each spatial location in its output activation tensor has full receptive field with respect to the network input? *[3 points]*

(c) Suppose we wanted to replace each pooling layer in the above network with a corresponding convolutional layer. Suggest a suitable configuration (filter count, filter spatial extent, stride, padding) for each of these replacements. *[5 points]*

Problem 2. Activation Functions *[5 points total]*

Consider a deep fully-connected network that, except for the final output layer, consists of a repeated assemblage of the following layer types:

$$\cdots \longrightarrow \text{Linear} \longrightarrow \text{Batch Norm} \longrightarrow act(\cdot) \longrightarrow \cdots$$

where $act(\cdot)$ is an element-wise activation function.

Which of the following would be poor choices for the activation function? Select all that apply and explain each selection.

(1) $act(\mathbf{x}) = \min(0, \mathbf{x})$

(2) $act(\mathbf{x}) = \frac{1}{\sqrt{2}}\mathbf{x}$

(3) $act(\mathbf{x}) = \mathrm{ReLU}(\mathbf{x} - 10)$

(4) $act(\mathbf{x}) = \frac{\exp(\mathbf{x})}{\exp(\mathbf{x})+1}$

(5) $act(\mathbf{x}) = \mathrm{sign}(\mathbf{x})$

Problem 3. Growing Network Capacity *[10 points]*

Consider a deep fully-connected network consisting of a repeated stack of linear layers followed by ReLU activations:

$$\begin{aligned}
\mathbf{h}^{(1)} &= \mathrm{ReLU}\left(\mathbf{W}^{(in)}\mathbf{x}\right) \\
\mathbf{h}^{(2)} &= \mathrm{ReLU}\left(\mathbf{W}^{(1)}\mathbf{h}^{(1)}\right) \\
&\cdots \\
\mathbf{h}^{(n)} &= \mathrm{ReLU}\left(\mathbf{W}^{(n-1)}\mathbf{h}^{(n-1)}\right) \\
\mathbf{y} &= \mathbf{W}^{(out)}\mathbf{h}^{(n)}
\end{aligned}$$

where $\mathbf{x} \in \mathbb{R}^C$ is the network input, $\mathbf{y} \in \mathbb{R}^D$ is the output. For simplicity, assume that all intermediate hidden state vectors have the same size, $\mathbf{h}^{(j)} \in \mathbb{R}^K$.

This makes $\mathbf{W}^{(in)}$ a $K \times C$ matrix, each $\mathbf{W}^{(j)}$ a $K \times K$ matrix, and $\mathbf{W}^{(out)}$ a $D \times K$ matrix.

Suppose we have been training the network, and would like to increase its capacity by expanding the dimensionality of each hidden state, say from K to $2K$ dimensions. This will require a corresponding increase in the size of the layer parameter matrices $\mathbf{W}$. When expanding the network, we want to preserve its current functionality, while also allowing the resumption of training to take advantage of the additional parameters.

Denoting the parameters of the expanded model by $\tilde{\mathbf{W}}$, suppose we initialize:

$$\tilde{\mathbf{W}}^{(j)} = \begin{bmatrix} \mathbf{W}^{(j)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$

where $\mathbf{0}$ is the $K \times K$ matrix with all zero entries. We similarly initialize $\tilde{\mathbf{W}}^{(in)}$ and $\tilde{\mathbf{W}}^{(out)}$, though expand each only along a single dimension.

(a) What goes wrong with the strategy of initializing all newly added parameters to zero? *[5 points]*

(b) Design a correct initialization for the parameters of the expanded model. *[5 points]*

Problem 4. Sequence Normalization *[20 points total]*

Consider a neural network formed by stacking multiple Long Short-Term Memory networks (LSTMs) in depth.

Specifically, given an input sequence $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$ of vectors $\mathbf{x}_t \in \mathbb{R}^C$, a first level LSTM (call it A) is defined by:

$$(\mathbf{c}_t^A, \mathbf{h}_t^A) = LSTM\left(\mathbf{x}_t, \mathbf{c}_{t-1}^A, \mathbf{h}_{t-1}^A; \mathbf{W}^A\right)$$

where $\mathbf{c}$ and $\mathbf{h}$ are cell and hidden states, respectively, and $\mathbf{W}$ parameterizes all operations within the standard LSTM block.

The hidden states generated by this LSTM serve as input to a second LSTM (call it B), defined by:

$$(\mathbf{c}_t^B, \mathbf{h}_t^B) = LSTM\left(\mathbf{h}_t^A, \mathbf{c}_{t-1}^B, \mathbf{h}_{t-1}^B; \mathbf{W}^B\right)$$

For simplicity, assume all hidden states have the same dimensionality, $\mathbf{h}_t \in \mathbb{R}^K$.

We continue stacking LSTMs in this fashion, producing a network that is deep (propagates through each separately parameterized LSTM layer) at every time step. Equivalently, we can view the system as running the LSTMs in turn, mapping sequence $\mathbf{X}$ to sequence $\mathbf{H}^A = \{\mathbf{h}_1^A, \mathbf{h}_2^A, \ldots, \mathbf{h}_T^A\}$ to sequence $\mathbf{H}^B = \{\mathbf{h}_1^B, \mathbf{h}_2^B, \ldots, \mathbf{h}_T^B\}$.

We would like to add a form of dynamic normalization between the distinct LSTMs stacked in depth. Suppose we also want to avoid dependence on batch size, and thereby support training with batches consisting of just a single sequence $\mathbf{X}$, rather than a set of sequences $\{\mathbf{X}_1, \mathbf{X}_2, \ldots\}$.

A strategy we could employ is to consider the elements of a single sequence as comprising a batch, and apply Batch Norm along this sequence dimension. Let's call this strategy Sequence Normalization. Inserting a Sequence Norm layer $SN(\cdot)$ between LSTMs A and B, would modify our network to compute:

$$\begin{aligned}\mathbf{Y} &= \quad SN(\mathbf{H^A}) \\ &= SN(\{\mathbf{h}_1^A, \mathbf{h}_2^A, \ldots, \mathbf{h}_T^A\})\end{aligned}$$

with resulting sequence $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T\}$ serving as input to LSTM B:

$$(\mathbf{c}_t^B, \mathbf{h}_t^B) = LSTM\left(\mathbf{y}_t, \mathbf{c}_{t-1}^B, \mathbf{h}_{t-1}^B; \mathbf{W}^B\right)$$

The next page contains questions regarding this setup.

Problem 4. Sequence Normalization (continued)

(a) Specify the trainable parameters of the $SN(\cdot)$ layer and write equations for the forward computation pass through this layer. Remember, we are just applying Batch Norm along a different definition of batch. *[3 points]*

(b) The above formulation imposes the requirement that we must evaluate the first layer (LSTM A) at all time steps, before evaluating LSTM B and any subsequent layers at the first time step. We would like to break this dependence by changing the $SN(\cdot)$ layer to use running estimates of mean and variance. Specify the parameters, as well as both the forward and backward computation passes for this modified $SN(\cdot)$ layer. *[12 points]*

(c) Could sequence normalization layers be used in a Transfomer architecture? If so, describe how to modify the Transformer architecture. If not, why not? *[5 points]*

Problem 5. Memory-Limited Backpropagation through Time (BPTT) *[20 points total]*

Consider a simple RNN defined by:

$$\mathbf{h}_t = \tanh\left(\mathbf{W}_{(hh)}\mathbf{h}_{t-1} + \mathbf{W}_{(xh)}\mathbf{x}_t\right)$$

where $\mathbf{x}_t \in \mathbb{R}^C$, $\mathbf{h}_t \in \mathbb{R}^K$.

Assume that at every time step, we have access to an output prediction function $y_t = f(\mathbf{h}_t)$ and a corresponding loss L_t, which we can query to obtain an upstream gradient $\frac{\partial L_t}{\partial \mathbf{h}_t} \in \mathbb{R}^K$.

Suppose we need to train our network on long input sequences $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$, but have limited memory for storing hidden state activations. That is, we have $M < T$ available memory slots for storing K-dimensional vectors. While we could make an approximation by using truncated BPTT, we will instead compensate for limited memory by re-computation, thereby allowing us to backprop through the entire sequence.

(a) Given a memory budget that scales logarithmically with T, specifically $M \propto O(\log_2(T))$, write pseudocode for a recursive implementation of backpropagation that is as computationally efficient as possible, while traversing the entire sequence without approximation. *[15 points]*

(b) For a fixed, constant memory budget M that does not scale with T, how computationally expensive would exact BPTT be for large T? You need only give a rough estimate, in relation to the baseline case where $M = T$. *[5 points]*

Problem 6. Processing Images with RNNs *[25 points total]*

Here, we consider replacing a convolutional layer with a module than runs RNNs horizontally and vertically across the spatial dimensions of an input tensor.

Let $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$ be an input tensor with spatial extent $H \times W$ and C channels.

Suppose we have two RNNs:

$$\begin{aligned}\mathbf{h}_t^A &= \tanh\left(\mathbf{W}_{(hh)}^A \mathbf{h}_{t-1}^A + \mathbf{W}_{(xh)}^A \mathbf{x}_t\right)\\ \mathbf{h}_t^B &= \tanh\left(\mathbf{W}_{(hh)}^B \mathbf{h}_{t-1}^B + \mathbf{W}_{(yh)}^B \mathbf{y}_t\right)\end{aligned}$$

which operate on C-dimensional input and hidden state ($\mathbf{x}_t, \mathbf{y}_t, \mathbf{h}_t^A, \mathbf{h}_t^B \in \mathbb{R}^C$).

Our custom module runs RNN A across rows of the input tensor $\mathbf{X}$, regarding each row as a length W sequence of C-dimensional inputs, producing an output tensor $\mathbf{H}^A \in \mathbb{R}^{H \times W \times C}$. We then set $\mathbf{Y} = \mathbf{H}^A$ and run RNN B across the columns of $\mathbf{Y}$, regarding each as a length H sequence of C-dimensional input vectors. Resulting output $\mathbf{H}^B \in \mathbb{R}^{H \times W \times C}$ serves as the output of the entire module.

(a) Write psuedocode corresponding to the above description of the forward pass for this module. *[4 points]*

(b) Write psuedocode for the backward pass. Assume we are given an upstream gradient tensor $\mathbf{G} \in \mathbb{R}^{H \times W \times C}$ *[10 points]*

(c) For spatial location (i, j) in the output tensor $\mathbf{H}^B$, what is the size of its receptive field within the input tensor $\mathbf{X}$? *[3 points]*

(d) Estimate the total number of floating point operations in a forward pass of the proposed module. How does this compare to a convolutional layer with filters of 3×3 spatial extent and the same number of input and output channels? *[5 points]*

(e) What advantages and/or disadvantages might the proposed module have in comparison to a convolutional layer? *[3 points]*