EE 150 [Winter 2024]

Problem Set 3

Due:	03/03/25 at 11:59 p.m.
Submit:	on gradescope
Collaborators:	None
Teaching Assistant:	Rohun Agrawal

Collaboration is allowed, just list the names of the people you collaborated in the header. If you choose to use LaTeX to type your submission, we recommend filling in this template.

To prepare your code submission, run `zip_assignment.py` from the root of the project folder and upload that zip to the code part of the assignment on gradescope.

If you have any other questions, come to office hours or ask us on Piazza for a quick response.

Problem 1: Convolutional Neural Networks **[40 pts]**

(a) Write a convolution algorithm from scratch (10 pts)

i) Implement the `Conv2d` class in `conv.py`.

ii) Implement the `FasterConv2d` class in `conv.py`. Ensure all tests in `conv_tests` pass.

iii) Run `benchmark_conv2d.py` to compare your implementations to the PyTorch Implementation. For a batch of 16 28×28 images, how many times faster was your faster implementation compared to your initial implementation and the PyTorch implementation compared to your faster implementation?

iv) Give a reason for why the PyTorch implementation is faster.

(b) Implement and train a CNN to classify MNIST (5 pts)

i) Implement the `CNN` class in `cnn.py`.

ii) Complete the `TODO`s in `train_test.py` that are needed to train the CNN. Then, run `train_mnist.py` to achieve at least 96% test accuracy after 5 training epochs. After training, take a look at the `results/` folder to find results, plots, and the saved model. Attach your accuracy and loss plots here and report your test accuracy.

(c) Manually picking filters (5 pts)

Copy any necessary code from your `CNN` class to the `ManualCNN` class. Now, instead of letting the convolutional layer learn 4 filters, you will set the 4 filters manually based on what filters might result in features useful for distinguishing numbers. Initialize `self.conv.weight.data` manually and note that we set `self.conv.weight.requires_grad` to `False` so the filters remain fixed during training (only the fully-connected layer will train). Your goal is to achieve at least 94% test accuracy. Take a look at the `ArgumentParser` in `train_mnist.py` to determine how to train the `ManualCNN`. Attach your accuracy and loss plots here and report your test accuracy. Once you are satisfied with the performance of your filters, run `visualize_filters.py` to visualize your chosen filters' effects on MNIST digits. Note that it requires the path to your saved `ManualCNN` model. Attach the visualization here.

(d) Transfer Learning using AlexNet on PCAM image dataset (5 pts)

i) What is AlexNet and why is it such a big deal? (1 sentence)

ii) Make a copy of this notebook, complete it, and provide a link to your colab notebook that is shareable. Make one observation of the loss/accuracy plots at the end and give a reason for it.

(e) Why skip connections prevent vanishing gradients (10 pts)

Consider an L-layer neural network with each layer having a width of 1. Let $h^{(0)}$ be the input of the network and for $l = 1, \ldots, L$, define

$$h^{(l)} = \sigma(w^{(l)} h^{(l-1)} + b^{(l)})$$

where $h^{(L)} = \hat{y}$ is the output of the network.

i) Write an expression for $\frac{\partial \mathcal{L}}{\partial w^{(1)}}$ incorporating terms of the form $\frac{\partial h^{(l)}}{\partial h^{(l-1)}}$.

ii) Write an expression for $\frac{\partial h^{(l)}}{\partial h^{(l-1)}}$. Treat σ as an arbitrary activation function. Then, compute this expression for $h^{(l-1)} = 0.1, w^{(l)} = 0.1, b^{(l)} = 0$ and $\sigma(z) = \text{ReLU}(z)$.

iii) If $\left|\frac{\partial h^{(l)}}{\partial h^{(l-1)}}\right| < 1$, what happens to $\frac{\partial \mathcal{L}}{\partial w^{(1)}}$ for large L? Why does this harm training? What about if $\left|\frac{\partial h^{(l)}}{\partial h^{(l-1)}}\right| > 1$?

iv) Let's say we add skip connections to our neural network. This means that

$$h^{(l)} = \sigma(w^{(l)} h^{(l-1)} + b^{(l)}) + h^{(l-1)}$$

Write an expression for $\frac{\partial h^{(l)}}{\partial h^{(l-1)}}$ and explain how the skip connection alleviates the vanishing gradient problem.

(f) Visualize loss landscape of ResNet (5 pts)

ResNet is a CNN architecture that takes advantage of residual connections (skip connections) to train very deep networks reliably. It is the most widely used CNN architecture today. Visit this link to view some visualizations of the loss landscape for various ResNets with and without skip connections. "No short" means without skip connections.

i) Explain the difference between the loss landscapes of ResNet20 (No short) and ResNet56 (No short). (1 – 2 sentences)

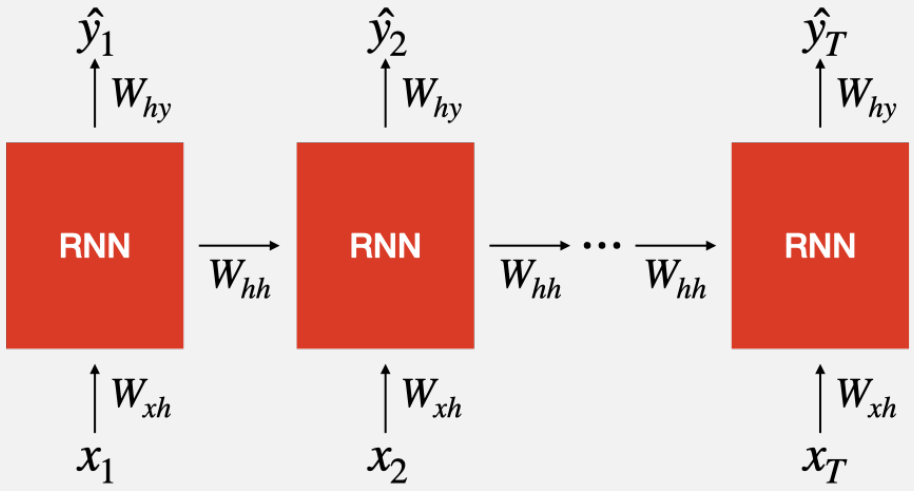
ii) Compare the loss landscapes of ResNet56 (No short) to ResNet56 (short). In terms of minimizing loss, why is the ResNet56 (short) loss landscape preferable? (1 – 2 sentences)

Solution

Student solution here.

Problem 2: RNNs and LSTMs **[40 pts]**

Recurrent Neural Networks (RNNs) were one of the first models aimed at handling sequential data. In this set we will focus on Simple RNNs, also called Elman Networks. Below is a figure of a Simple RNN.



The hidden state and output state computations are

$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

$$\hat{y}_t = W_{hy}h_t + b_y$$

for $t = 1, \ldots, T$. Note that the loss is

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}_t(y_t, \hat{y}_t)$$

where $\mathcal{L}_t$ is the same for all timesteps.

(a) Train RNN on IMDB Movie Reviews to classify sentiment (15 pts)

Text sentiment classification is the task of determining the emotional tone of a piece of text. Here, we will classify whether IMDB movie reviews are positive or negative from this dataset. For a sequence model like an RNN, the input is a word x_t and the output is the binary classification prediction $\hat{y}_t$. After passing in an entire piece of text, one word at a time, we have the sequence $\{\hat{y}_t\}_{t=1}^{T}$ where $p(\hat{y}_t) = p(\hat{y}_t|h_{t-1}) = p(\hat{y}_t|x_{<t}, h_{<t})$ where h_{t-1} acts as a summary of all the past information. Thus, we use $\hat{y}_T$ to be the sentiment classification for the entire piece of text.

i) Implement the `TODO`s in the `RNNLayer` class in `rnn.py`.

ii) Implement the `TODO`s in the `RNN` class in `rnn.py`.

iii) Text data can be very messy. Preprocessing a text dataset strategically before turning it into a Bag of Words (BOW) can significantly improve the performance of a text-based model. Complete the `TODO`s in the `TextClassificationDataset` class.

iv) Since the input to the RNN model has shape (B, T, D), the output has shape $(B, T, N_{classes})$. We only want $\hat{y}_T$. Fill in the `TODO` in `train_test.py` that accounts for this.

v) Train the RNN using `train_imbd.py` for 5 epochs, a minimum sequence length of 0 and a maximum sequence length of 200. Then train the RNN for 5 epochs, on a minimum sequence length of 200 and a maximum sequence length of 400 (check the `argparse` in `train_imbd.py` for changing sequence lengths). Tip: for faster debugging, set the `dataset_length` to be small so that you can hit errors faster, but use the full dataset once debugged. Attach your accuracy and loss plots here for both models and report test accuracies.

(b) Backpropagation through time equations (15 pts)

i) Let B be the size of a batch, H be the hidden dimension of the network, and D be the input/output embedding dimension. Write the shapes for x_t, $\hat{y}_t$, h_t, W_{xh}, W_{hh}, W_{hy}, b_h, and b_y.

ii) Given $\frac{\partial \mathcal{L}_t}{\partial \hat{y}_t}$, find expressions for $\frac{\partial \mathcal{L}}{\partial W_{hy}}$ and $\frac{\partial \mathcal{L}}{\partial b_y}$.

iii) Given a function $f(\alpha_1, \alpha_2, \ldots, \alpha_n)$ where $\alpha_i = g_i(\beta)$, give an expression for $\frac{\partial f}{\partial \beta}$. Hint: this is the multivariate chain rule.

iv) Find an expression for $\frac{\partial h_t}{\partial W_{hh}}$ considering the fact that $h_t = f(h_1, h_2, \ldots, h_{t-1}, W_{hh})$. Your answer should not contain any terms of the form $\frac{\partial h_i}{\partial h_j}$ with $i > j+1$ and should only contain partial derivatives (don't evaluate the derivatives). Hint: Use chain rule as much as possible. The Π notation for products might help.

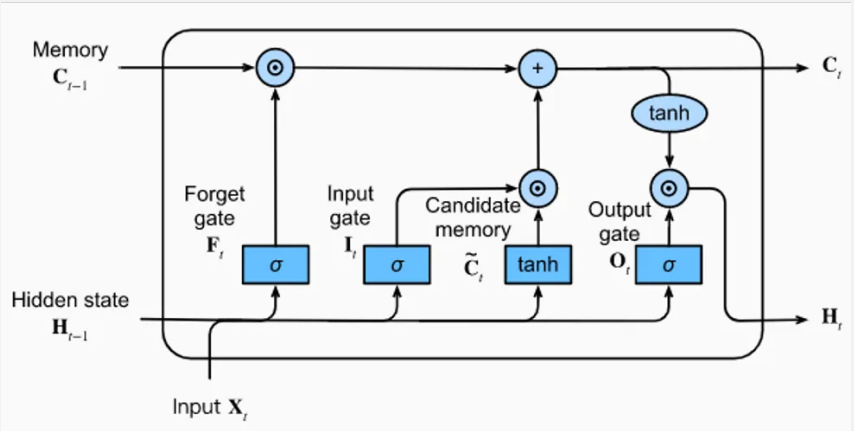
v) Find an expression for $\frac{\partial \mathcal{L}}{\partial W_{hh}}$ using your expression for $\frac{\partial h_t}{\partial W_{hh}}$. Again, your answer should contain only partial derivatives.

vi) Find an expression for $\frac{\partial \mathcal{L}}{\partial W_{xh}}$. Hint: Follow the same procedure for finding $\frac{\partial \mathcal{L}}{\partial W_{hh}}$.

vii) Identify which term in $\frac{\partial \mathcal{L}}{\partial W_{xh}}$ and $\frac{\partial \mathcal{L}}{\partial W_{hh}}$ leads to the vanishing/exploding gradient problem in RNNs and briefly explain why. (1 sentence)

(c) Train LSTM on IMDB Movie Reviews to classify sentiment (10 pts)

The Long-Short Term Memory (LSTM) model aims to alleviate issues with the vanishing gradient problem in RNNs by specifying different gates to control the passage of information between recurrent cells. The long term memory is C_{t-1} while the short-term memory is h_{t-1}.



i) The output of the forget gate is defined as

$$F_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$

where σ is Sigmoid.
The output of the input gate is defined as

$$I_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$

and is used to select from the candidate memory. The candidate memory is defined as

$$\tilde{C}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

Then, the updated long-term memory is

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$

where $\odot$ denotes element-wise multiplication. Explain in 1-2 sentences what the product $I_t \odot \tilde{C}_t$ represents. Explain in 1-2 sentences why it is added to $F_t \odot C_{t-1}$ to update C.

ii) The output of the output gate is defined as

$$O_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

The updated hidden state is

$$h_t = O_t \odot \tanh(C_t)$$

The output for the cell (not shown in the figure) is

$$\hat{y}_t = \text{softmax}(W_{xh}h_t + b_y)$$

Explain in 1-2 sentences why we update h by multiplying O_t by $\tanh(C_t)$ element-wise.

iii) Implement the `TODO`s in the `LSTMLayer` class in `lstm.py`. Implement the `TODO`s in the `LSTM` class in `lstm.py`. Train the LSTM using `train_imbd.py` (look at the `argparse` for how to switch the model type) for 5 epochs, a minimum sequence length of 0, and a maximum sequence length of 200. Then train the LSTM for 5 epochs, on a minimum sequence length of 200, and a maximum sequence length of 400. Attach your accuracy and loss plots here for both models and report test accuracies.

Solution

Student solution here.

Problem 3: Transformers **[20 pts]**

(a) Attention (10 pts)

i) Write down the scaled dot-product self-attention function Attention(x) in terms of x, W_q, W_k, W_v, and d_k. What are the shapes of x, W_q, W_k, W_v in terms of sequence length T, embedding dimension D, key dimension d_k, and value dimension d_v? For simplicity, we are not considering the batch dimension because in implementation this function is applied to every sequence in the batch the same way. You can use the function softmax(z) without defining it. What is the shape of the output of Attention(x)?

ii) The gradient of the softmax function is very small for inputs of large magnitude. Assume that $Q = xW_q$ and $K = xW_k$ are random variables sampled from a multivariate standard normal and $T = 1$. What is the distribution of the input to the softmax function if we don't scale the input by $1/\sqrt{d_k}$? Why does scaling by $1/\sqrt{d_k}$ alleviate the aforementioned issue? Hint: Write out the dot product and apply the Central Limit Theorem.

iii) Write down how to compute multi-head self attention in terms of x, W_o, the sequences of matrices $\{W_k^i\}_{i=1}^h$, $\{W_q^i\}_{i=1}^h$, $\{W_v^i\}_{i=1}^h$, d_v, and d_k, where h is the number of heads. Again, you can use softmax(z) without defining it. What are the shapes of W_k^i, W_q^i, W_v^i, and W_o? Assume that the outputs have dimension D.

iv) Implement the `SelfAttention` class in `attention.py`. Implement the `MultiHeadAttention` class as well. Ensure all the `attention_tests` pass.

(b) Train Transformer on IMDB Reviews to classify sentiment (5 pts)

Implement the `TODO`s in `transformer.py`. Train the Transformer using `train_imbd.py` for 5 epochs, a minimum sequence length of 0 and a maximum sequence length of 200. Then train the Transformer for 5 epochs, on a minimum sequence length of 200 and a maximum sequence length of 400. Attach your accuracy and loss plots here for both models and report test accuracies.

(c) Compare Sequence Models (2.5 pts)

Fill in the following table with test accuracies of all the models. Note that architecture hyperparameters were already adjusted for you so that each model has the same number of parameters, making this a fair comparison. What do you notice? (1 – 2 sentences)

Model	Short Sequences	Long Sequences
RNN		
LSTM		
Transformer		

(d) Try your model (2.5 pts)

Metrics can tell one story, but trying a model first hand can tell another. Run `test_imdb.py` with the path to the Transformer you trained on shorter sequences. You'll be able to write your own reviews and see what your trained Transformer predicts its sentiment as. Give an example of a very positive (95%+ confidence) review, a very negative review, and a review with a confidence below 90%.

Solution

Student solution here.