Section 09: Solutions

Solution:

Section Plan

- CNN slides
- Problem 1 (CNN advantage + disadvantage)
- RNN + LSTM slides
- RNN + LSM problem
- Attention + Transformer Slides
- Walk through Attention + Transformer problem

1. Convolutional Neural Networks

(a) Discuss the advantages of a convolutional layer compared to a fully connected one. **Solution:**

Convolutional layers are more flexible than fully connected ones since not all input neurons affect all output neurons. In addition, the number of weights per layer is smaller than that of linear layers, which would ease computation with high-dimensional data.

(b) Discuss the advantages of maxpooling in CNN. **Solution:**

Pooling layers are used to downsample feature maps, which make processing more efficient by reducing the number of parameters to learn.

2. Shapes in Convolutional Neural Networks

When designing a convolutional neural network, it's important to think about the shape of the data flowing through the network. In this problem you will get gain experience with thinking about the shapes in a neural network and a better intuition for why convolutional neural networks require so few parameters compared to fully connected layers.

Shape of a convolutional layer / maxpooling output: For a $n \times n$ input, $f \times f$ filter, padding p and stride s, the output size is $o \times o$ where:

$$o = \frac{n - f + 2p}{s} + 1$$

We will use Pytorch Conv2d to represent a 2D convolution, and Pytorch MaxPool2d to represent a 2D max pooling. Take a look at the documentation on Conv2d and MaxPool2d.

(a) Assume your input is a batch of N 64×64 RGB images. The input tensor your neural network receives will have shape $(N, 3, 64, 64)$. For each of the following operations, determine the new shape of the tensor as it flows through the network. Note that activations are omitted since they don't change the shape of the data as they act coordinate-wise.

1. `Conv2D(in_channels=3, out_channels=16, kernel_size=3, stride=1, padding=1)`

Solution:

$(N, 16, 64, 64)$

2. `MaxPool2d(kernel_size=2, stride=2, padding=0)`

Solution:

$(N, 16, 32, 32)$

3. `Conv2D(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=0)`

Solution:

$(N, 32, 30, 30)$

4. `MaxPool2d(kernel_size=2, stride=2, padding=1)`

Solution:

$(N, 32, 16, 16)$

5. `Conv2D(in_channels=32, out_channels=8, kernel_size=1, stride=1, padding=0)`

Solution:

$(N, 8, 16, 16)$

6. `Conv2D(in_channels=8, out_channels=4, kernel_size=5, stride=1, padding=0)`

Solution:

$(N, 4, 12, 12)$

7. `Flatten`

Solution:

$(N, 576)$

8. `Linear(in_features=576, out_features=10)`

Solution:

$(N, 10)$

(b) Again assume your input is a batch of N 64×64 RGB images. Now compute the number of parameters that each layers has. For the convolutional layers, also compute the number of parameters a fully connected layer

mapping from the flattened input channels to the flattened output channels would have. It is okay to leave the number of parameters as products and additions such as $64 \cdot 32 + 16$.

1. `Conv2D(in_channels=3, out_channels=16, kernel_size=3, stride=1, padding=1)`

   Solution:

   Conv: $3 \cdot 16 \cdot 3 \cdot 3 + 16 = 448$

   Fully connected: $3 \cdot 64 \cdot 64 \cdot 16 \cdot 64 \cdot 64 + 16 \cdot 64 \cdot 64 = 805371904$

2. `MaxPool2d(kernel_size=2, stride=2, padding=0)`

   Solution:

   0

3. `Conv2D(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=0)`

   Solution:

   Conv: $16 \cdot 32 \cdot 3 \cdot 3 + 32 = 4640$

   Fully connected: $16 \cdot 32 \cdot 32 \cdot 32 \cdot 30 \cdot 30 + 32 \cdot 30 \cdot 30 = 471888000$

4. `MaxPool2d(kernel_size=2, stride=2, padding=1)`

   Solution:

   0

5. `Conv2D(in_channels=32, out_channels=8, kernel_size=1, stride=1, padding=0)`

   Solution:

   Conv: $32 \cdot 8 + 8 = 264$

   Fully connected: $32 \cdot 16 \cdot 16 \cdot 8 \cdot 16 \cdot 16 + 8 \cdot 16 \cdot 16 = 16779264$

6. `Conv2D(in_channels=8, out_channels=4, kernel_size=5, stride=1, padding=0)`

   Solution:

   Conv: $8 \cdot 4 \cdot 5 \cdot 5 + 4 = 804$

   Fully connected: $8 \cdot 16 \cdot 16 \cdot 4 \cdot 12 \cdot 12 + 4 \cdot 12 \cdot 12 = 1180224$

7. `Flatten`

   Solution:

$$0$$

8. `Linear(in_features=576, out_features=10)`

Solution:

$$576 \cdot 10 + 10 = 5770$$

3. RNNs, LSTMs

Suppose we have a toy scalar RNN with the recurrence

$$h_t = \text{ReLU}(wh_{t-1} + ux_t)$$

where $h_t, x_t \in \mathbb{R}$, and $w, u \in \mathbb{R}$ are weights shared across all timesteps. Assume all pre-activation values are positive, meaning $\text{ReLU}' = 1$ throughout the network.

(a) Compute $\frac{\partial h_3}{\partial h_1}$ treating x_t as a constant, with $w = 0.3$ and with $w = 3$.

Solution:

Note that for each layer $h_t = \text{ReLU}(wh_{t-1}+ux_t)$, we have $\frac{\partial h_t}{\partial h_{t-1}} = \text{ReLU'}(wh_{t-1}+ux_t)\cdot w = 1\cdot w = w$. We can compute $\frac{\partial h_3}{\partial h_1}$ using chain rule:

$$\frac{\partial h_3}{\partial h_1} = \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} = w \cdot w = w^2 = (0.3)^2 = 0.09$$

$$\frac{\partial h_3}{\partial h_1} = \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} = w \cdot w = w^2 = (3)^2 = 9$$

(b) Compute $\frac{\partial h_{20}}{\partial h_1}$, with $w = 0.3$ and $w = 3$, and state how the magnitude of w impacts the model's ability to learn long-range dependencies.

Solution:

$$\frac{\partial h_{20}}{\partial h_1} = \frac{\partial h_{20}}{\partial h_{19}} \cdot \frac{\partial h_{19}}{\partial h_{18}} ... \frac{\partial h_2}{\partial h_1} = w^{19} = (0.3)^{19} \approx 0$$

$$\frac{\partial h_{20}}{\partial h_1} = \frac{\partial h_{20}}{\partial h_{19}} \cdot \frac{\partial h_{19}}{\partial h_{18}} ... \frac{\partial h_2}{\partial h_1} = w^{19} = (3)^{19} \approx 1 \times 10^9$$

The main takeaway here is that as we propagate back through more layers, we face the issue of vanishing gradients or exploding gradients. This is the inherent problem with RNNs, and why LSTMs are far more popular.

(c) To address the issue of vanishing gradients, the LSTM architecture uses a cell update

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t$$

where $f_t \in [0, 1]$ is the learned forget gate. Suppose the network learns $f_t = 0.99$ for all t. Compute $\frac{\partial c_{20}}{\partial c_1}$, treating $i_t \cdot \tilde{c}_t$ as a constant w.r.t. c_{t-1}.

Solution:

$$\frac{\partial c_t}{\partial c_{t-1}} = f_t \rightarrow \frac{\partial c_{20}}{\partial c_1} = \prod_{t=2}^{20} f_t = (0.99)^{19} \approx 0.83$$

Note: This architecture creates a "gradient highway". Because f_t is learned, the network can choose to maintain a gradient close to 1 (by learning $f_t \approx 1$), which prevents the exponential decay seen in the RNN example. For exploding gradients, we need to resort to gradient clipping (which can be done for RNNs as well).

4. Attention and Transformers

In this example, we will do a full forward pass through the self-attention mechanism, which is often used in Transformer architectures, such as ChatGPT. Our input sequence will be "The red panda is cute.", where our data matrix is

$$X = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Our weights will be given as

$$W_Q = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad W_K = \begin{bmatrix} \sqrt{3}\ln(80) & 0 & 0 \\ 0 & 0 & 0 \\ \sqrt{3}\ln(17) & 0 & 0 \end{bmatrix}, \quad W_V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

As you can see, $d_{model} = 3$ here.

(a) Calculate the Query, Key, and Value matrices Q, K, V.

Solution:

$$Q = XW_Q = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$K = XW_K = \begin{bmatrix} 0 & 0 & 0 \\ \sqrt{3}\ln(80) & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \sqrt{3}\ln(17) & 0 & 0 \end{bmatrix}$$

$$V = XW_V = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(b) Compute the pre-softmax attention scores, i.e., $\frac{QK^\top}{\sqrt{d_{model}}}$.

Solution:

$$QK^\top = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & \sqrt{3}\ln(80) & 0 & 0 & \sqrt{3}\ln(17) \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{3}\ln(80) & 0 & 0 & \sqrt{3}\ln(17) \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\frac{QK^\top}{\sqrt{d_{model}}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \ln(80) & 0 & 0 & \ln(17) \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(c) Compute the softmax to obtain our final attention scores.

Solution:

For all zero rows, the softmax is just a vector of all $\frac{1}{5}$s. This is because $\text{softmax}(\mathbf{0}_i) = \frac{e^0}{\sum_j e^0} = \frac{1}{5}$

For row 3, the denominator of the softmax will be $e^0 + e^{\ln(80)} + e^0 + e^0 + e^{\ln(17)} = 1 + 80 + 1 + 1 + 17 = 100$. Therefore, the softmax of this row will be

$$\begin{bmatrix} e^0/100 & e^{\ln(80)}/100 & e^0/100 & e^0/100 & e^{\ln(17)}/100 \end{bmatrix} = \begin{bmatrix} 0.01 & 0.8 & 0.01 & 0.01 & 0.17 \end{bmatrix}$$

Thus, our final attention scores are

$$\begin{bmatrix} 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.01 & 0.80 & 0.01 & 0.01 & 0.17 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \end{bmatrix}$$

(d) Finally, compute $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_{model}}}\right)V$.

Solution:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_{model}}}\right)V = \begin{bmatrix} 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.01 & 0.80 & 0.01 & 0.01 & 0.17 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 \\ 0.80 & 0.01 & 0.17 \\ 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 \end{bmatrix}$$

(e) What information does it seem like this attention head is transferring between word embeddings?

Solution:

The only word embedding with a meaningful update is ”panda”, which is a noun. When we look at the attention scores, it is attending to two words: ”red” and ”cute”. We can infer that this attention head is getting nouns to attend to the relevant adjectives, such that after this self-attention block, the embedding for ”panda” now also encodes that it is ”cute” and ”red”.