Section 09: Convolutional Neural Networks, Recurrent Neural Networks, Attention & Transformers

1. Convolutional Neural Networks

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

(b) Discuss the advantages of maxpooling in CNN.

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)`

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

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

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

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

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

7. `Flatten`

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

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

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

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

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

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

7. `Flatten`

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

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 $w = 3$.

(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.

(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}.

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.

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

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

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

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