EECS 182 Deep Neural Networks
Spring 2023 Anant Sahai

Review: ConvNets & GNNs

1. CNN Design decisions

Consider a CNN for classification consisting of the layers shown below. "..." indicates appropriate arguments that have been passed in.

```
nn.Conv2d(...),
nn.ReLU(),
nn. MaxPool2d(...),
nn. Conv2d(...),
nn. ReLU(),
nn. MaxPool2d(...),
nn. Flatten(),
nn. Linear(...),
nn. Softmax(...)
```

In the questions below, explain how you would modify this design to achieve particular objectives.

(a) You would like to use fewer parameters in the fully connected layer at the end of the network.

(b) You would like the model to be able to handle images of different sizes.

(c) You would like to increase the receptive field of each pixel in the feature map output by the second conv layer.

(d) You would like every conv layer to leave the height and width dimensions of its input unchanged.

(e) You would like to add 50 more layers to this network without suffering from vanishing gradients.

2. 3D Convolutions

We've seen CNNs in 1D and 2D settings. Now, let's consider a 3D setting: you are building a classifier to detect the activity shown in short video clips.

(a) One approach is to build a CNN which uses 3D convolutions. What is the size of the parameters in each convolutional layer? Write your answer in terms of F (number of filters), S (stride), C (number of input channels), H (input height), W (input width), T (input time length), K (kernel size).

(b) Another approach is to process frames individually, then use an RNN to aggregate information over time. How would you combine a convolutional encoder with an RNN?

3. Feature Dimensions of Convolutional Neural Network

In this problem, we compute output feature shape of convolutional layers and pooling layers, which are building blocks of CNN. Let's assume that input feature shape is $W \times H \times C$, where W is the width, H is the height and C is the number of channels of input feature.

(a) A convolutional layer has 4 architectural hyperparameters: the filter size(K), the padding size (P), the stride step size (S) and the number of filters (F). **How many weights and biases are in this convolutional layer? And what is the shape of output feature that this convolutional layer produces?**

(b) A max pooling layer has 2 architectural hyperparameters: the stride step size(S) and the "filter size" (K). **What is the output feature shape that this pooling layer produces?**

(c) Let's take a real example. We are going to describe a convolutional neural net using the following pieces:

- CONV3-F denotes a convolutional layer with F different filters, each of size $3 \times 3 \times C$, where C is the depth (i.e. number of channels) of the activations from the previous layer. Padding is 1, and stride is 1.
- POOL2 denotes a 2×2 max-pooling layer with stride 2 (pad 0)
- FLATTEN just turns whatever shape input tensor into a one-dimensional array with the same values in it.
- FC-K denotes a fully-connected layer with K output neurons.

Note: All CONV3-F and FC-K layers have biases as well as weights. **Do not forget the biases when counting parameters.**

Now, we are going to use this network to do inference on a single input. **Fill in the missing entries in this table of the size of the activations at each layer, and the number of parameters at each**

layer. You can/should write your answer as a computation (e.g. $128 \times 128 \times 3$**) in the style of the already filled-in entries of the table.**

Layer	Number of Parameters	Dimension of Activations
Input	0	$28 \times 28 \times 1$
CONV3-10		$28 \times 28 \times 10$
POOL2	0	$14 \times 14 \times 10$
CONV3-10	$3 \times 3 \times 10 \times 10 + 10$	
POOL2		
FLATTEN	0	490
FC-3		3

4. Graph Neural Networks

For an undirected graph with no labels on edges, the function that we compute at each layer of a Graph Neural Network must respect certain properties so that the same function (with weight-sharing) can be used at different nodes in the graph. Let's focus on a single particular "layer" ℓ. For a given node i in the graph, let $\mathbf{s}_i^{\ell-1}$ be the self-message (i.e. the state computed at the previous layer for this node) for this node from the preceeding layer, while the preceeding layer messages from the n_i neighbors of node i are denoted by $\mathbf{m}_{i,j}^{\ell-1}$ where j ranges from 1 to n_i. We will use w with subscripts and superscripts to denote learnable scalar weights. If there's no superscript, the weights are shared across layers. Assume that all dimensions work out.

(a) **Tell which of these are valid functions for this node's computation of the next self-message** $\mathbf{s}_i^{\ell}$**. For any choices that are not valid, briefly point out why.**

Note: we are *not* asking you to judge whether these are useful or will have well behaved gradients. Validity means that they respect the invariances and equivariances that we need to be able to deploy as a GNN on an undirected graph.

(i) $\mathbf{s}_i^{\ell} = w_1 \mathbf{s}_i^{\ell-1} + w_2 \frac{1}{n_i} \sum_{j=1}^{n_i} \mathbf{m}_{i,j}^{\ell-1}$

(ii) $\mathbf{s}_i^{\ell} = \max(w_1^{\ell} \mathbf{s}_i^{\ell-1}, w_2 \mathbf{m}_{i,1}^{\ell-1}, w_3 \mathbf{m}_{i,2}^{\ell-1}, \ldots, w_{n_i-1} \mathbf{m}_{i,n_i}^{\ell-1})$ where the $\max$ acts component-wise on the vectors.

(iii) $\mathbf{s}_i^{\ell} = \max(w_1^{\ell} \mathbf{s}_i^{\ell-1}, w_2 \mathbf{m}_{i,1}^{\ell-1}, w_2 \mathbf{m}_{i,2}^{\ell-1}, \ldots, w_2 \mathbf{m}_{i,n_i}^{\ell-1})$ where the $\max$ acts component-wise on the vectors.

(b) Suppose we decide to use the following update rule for the internal state of the nodes at layer ℓ.

$$\mathbf{s}_i^{\ell} = \mathbf{s}_i^{\ell-1} + W_1 \frac{\sum_{j=1}^{n_i} \tanh\left(W_2 \mathbf{m}_{i,j}^{\ell-1}\right)}{n_i} \tag{1}$$

where the $\tanh$ nonlinearity acts element-wise.

For a given node i in the graph, let $\mathbf{s}_i^{\ell-1}$ be the self-message for this node from the preceeding layer, while the preceeding layer messages from the n_i neighbors of node i are denoted by $\mathbf{m}_{i,j}^{\ell-1}$ where j ranges from 1 to n_i. We will use W with subscripts and superscripts to denote learnable weights in matrix form. If there's no superscript, the weights are shared across layers.

(i) **Which of the following design patterns does this update rule have?**

☐ Residual connection

☐ Batch normalization

(ii) **If the dimension of the state $\mathbf{s}$ is d-dimensional and W_2 has k rows, what are the dimensions of the matrix W_1?**

(iii) If we choose to use the state $\mathbf{s}_i^{\ell-1}$ itself as the message $\mathbf{m}^{\ell-1}$ going to all of node i's neighbors, **please write out the update rules corresponding to (1) giving $\mathbf{s}_i^{\ell}$ for the graph in Figure 1 for nodes $i = 2$ and $i = 3$ in terms of information from earlier layers.** Expand out all sums.

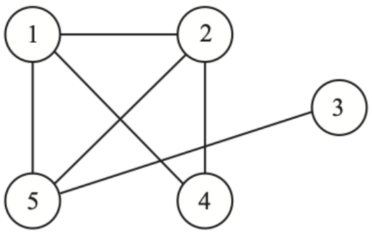

Figure 1: Simple Undirected Graph

Contributors:

- Suhong Moon.
- Fei-Fei Li.
- Jerome Quenum.
- Anant Sahai.

