IN5400 Exam

June 4, 2021

Exercise 1

1a

1. Residual connections improve gradient flow by letting earlier layers be "fewer steps" away from the loss function.

2. Residual connections let later layers combine higher level features from the "skipped layers" with lower level features from the data that was sent forward by the residual connection.

1b

```
class subnet(nn.Module):
    ...

    def forward(self, x):
        x = self.convblock17(x)
        x_res = self.convblock3(x)
        x = self.strangeblock1(x_res)
        x = self.convblock11(x)
        x = self.convblock9(x) + x_res
        x = self.strangeblock2(x)
        x = self.convblock81(x)

        return x
```

In the case that x_res has the wrong dimensions, a model specific fix can be applied.

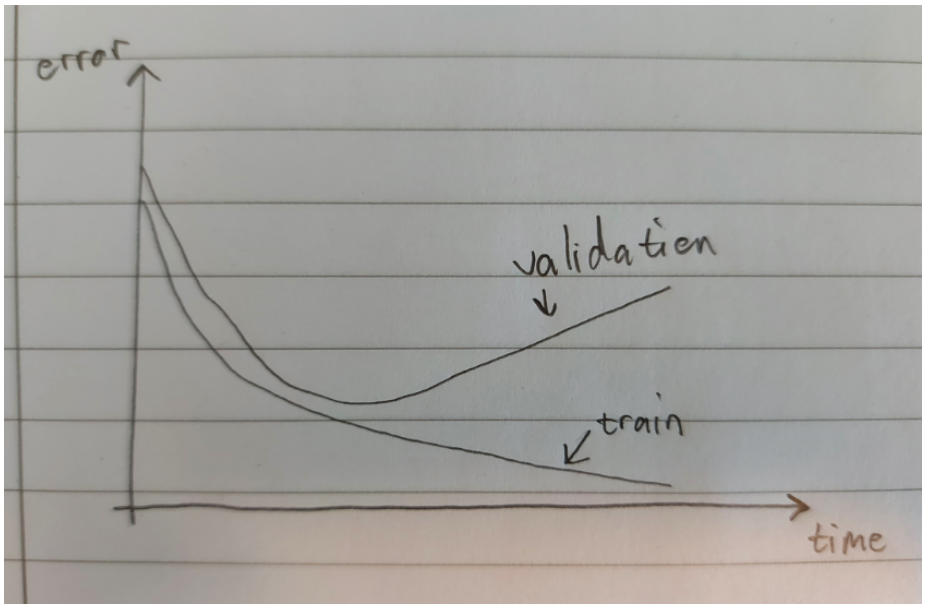
Exercise 2

2a

The network might fail to converge to any local minimum. A too high learning rate can be noticed by a quick drop in training error followed by sporadic change in training error with no apparend convergence.

2b

When training a network, we diagnose overfitting by splitting our training data into training and valitation data. We train the network on training data, and validate it using the validation error. If the training error is much smaller than the validation data, we have overfitting. We typically see overfitting happen later in training, as the model performs better on the training data, but worse on the validation data due to fitting spurious correlations in the training data. See figure .



2c

Option 1: You fit parameters to the test set by choosing the learning rate which performs best on it. This is really bad, as you have no idea how well your model acutally works on unseen data.

Option 2: You have a model which you can actually test, which is good.

Exercise 3

3a

1. You can augment your training data by cropping/rotating/etc. the images. As long as the augmented images are still within the domain of images for which you wish to train a model, you now have more training data, which will improve performance.

2. You can improve transfer learning by only training the final layers of your network. How many of the last layers to train varies. This might help as the pretrained weights of the first layers are already trained for extracting low level features of images. And retraining these on a new domain is often unnecessary.

3. If your dataset is imbalanced, if you have less instances of some classes, it might be beneficial to create more augmented "duplicates" of these images in your dataset. If for example you have fewer images of giraffes than you

would like, you could flip and crop the images you have to create more useful images.

Exercise 4

4a

Both fully-connected layers and 1D-convolution layers have 1D input and 1D output. However, in a 1D convolution, each element of the output is a vector product of the same 1D kernel weights and only part of the input data. In a fully connected layer, each element of the output is a vector product of different weight vectors and the entire input data.

4b

1. Adaptive instance normalization normalizes the input according to a different (style) input, while batch normalization scales according to the input across the batch of input data.

2. The scaling of batch normalization uses trainable parameters, while adaptive instance normalization does not.

Exercise 5

5a

From the week 5 lecture, we have: the output kernel size for a kernel of size ksize and stride s with padding of r is given as: floor((M+ 2rksize)/s+ 1). M is the input shape, r is the padding, ksize is the kernel size, s is the stride.

The formula can be interpreted as such: The length we need the kernel to cross, which is M + 2r due to the padding. Then the kernel size reduces the effective length, since we need room for it at the ends. And then the stride divides the length, due to the stride being the length of each step, and we are interested in the number of steps. And we round down since we can't go over the length.

This directly gives an output feature map of size (24, 32), there are no intermediary steps to show.

5b

Preamble:

3 elements in l_0 give 1 output in l_1.

5 elements in l_0 give 2 output in l_1. (due to stride and overlap)

$3 + 2n$ elements in l_0 give $n + 1$ output in l_1. (due to stride and overlap)

$5 + 3n$ elements in l_1 give $n + 1$ output in l_2. (due to stride and overlap)

$3 + n$ elements in l_2 give $n + 1$ output in l_3. (due to stride and overlap)

So:

1 output in l_3 needs 3 elements from l_2, which then need $5+3*2=11$ elemens from l_1 which then need $3+2*10=32$ elements from l_0.

1 output in l_2 needs 5 elements from l_1, which then need $3+2*4=11$ elemens from l_0.

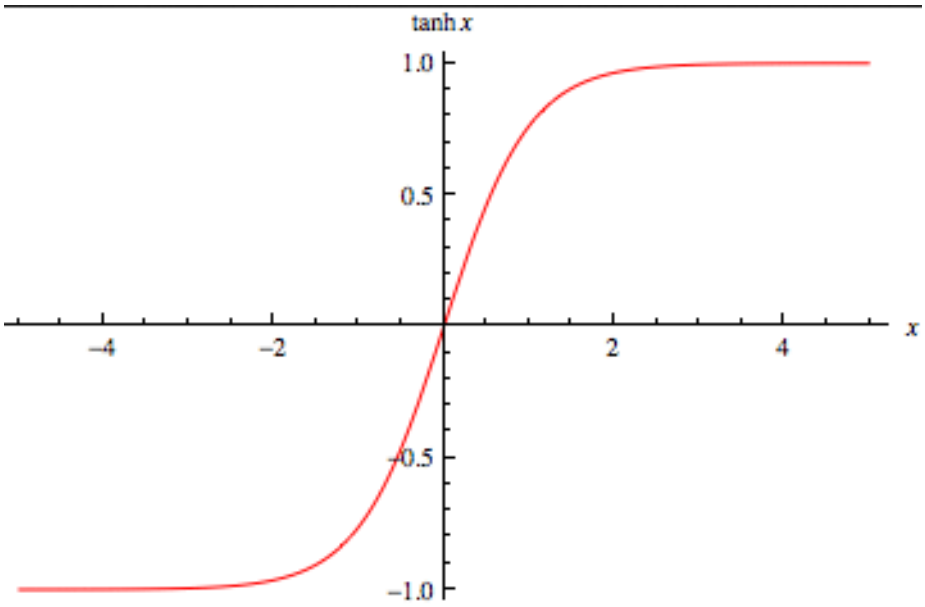
Exercise 6

6a

A surrogate attack has the advantage that it does not need direct access to the inner workings of a model which we want to attack, only its outputs for specific inputs to train a surrogate on.

2b

The gradient of the output of a tanh layer are very prone to being very small. As seen in figure , for large and small inputs, the slope of tanh is close to zero, which makes training very difficult and slow. The inputs to this layer are likely to not face this problem, and should thus often be used instead.



Exercise 7

7a

$$\begin{aligned}\frac{\partial e}{\partial x_2} &= \frac{\partial e}{\partial g}\frac{\partial g}{\partial x_2} \\ &= \frac{\partial e}{\partial g}(\frac{\partial g}{\partial j}\frac{\partial j}{\partial x_2} + \frac{\partial g}{\partial h}\frac{\partial h}{\partial x_2})\end{aligned} \tag{1}$$

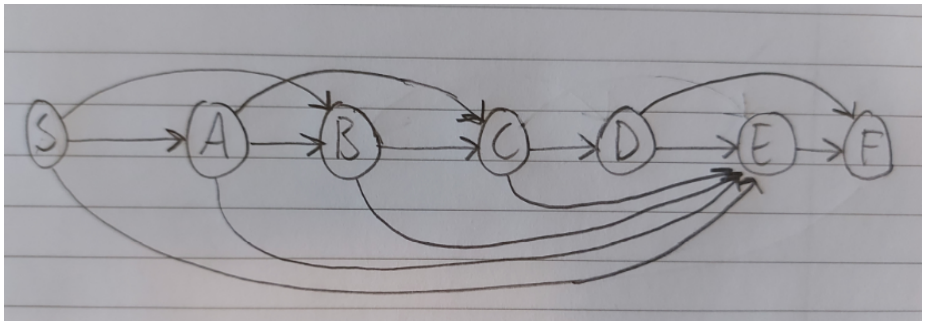
7b

$$\begin{aligned}\frac{\partial e}{\partial x_1} &= \frac{\partial e}{\partial c}\frac{\partial c}{\partial x_1} + \frac{\partial e}{\partial d}\frac{\partial d}{\partial x_1} + \frac{\partial e}{\partial f}\frac{\partial f}{\partial x_1} \\ &= \frac{\partial e}{\partial c}(\frac{\partial c}{\partial b}\frac{\partial b}{\partial x_1} + \frac{\partial c}{\partial a}\frac{\partial a}{\partial x_1}) + \frac{\partial e}{\partial d}(\frac{\partial d}{\partial a}\frac{\partial a}{\partial x_1}) + \frac{\partial e}{\partial f}(\frac{\partial f}{\partial d}\frac{\partial d}{\partial x_1} + \frac{\partial f}{\partial a}\frac{\partial a}{\partial x_1} + n'_f v_1) \\ &= \frac{\partial e}{\partial c}(\frac{\partial c}{\partial b}(\frac{\partial b}{\partial a}\frac{\partial a}{\partial x_1}) + \frac{\partial c}{\partial a}\frac{\partial a}{\partial x_1}) + \frac{\partial e}{\partial d}(\frac{\partial d}{\partial a}\frac{\partial a}{\partial x_1}) + \frac{\partial e}{\partial f}(\frac{\partial f}{\partial d}(\frac{\partial d}{\partial a}\frac{\partial a}{\partial x_1}) + \frac{\partial f}{\partial a}\frac{\partial a}{\partial x_1} + n'_f v_1)\end{aligned} \tag{2}$$

where the core rule gave the term $n'_f v_1$ where n'_f is the derivative of the activation of f (the outer function), and v_1 is the weight of x_1, which came from the derivative of the core.

Exercise 8

8a



8b

If the diagonal entries were 1, all neurons would use their output as part of their input. This would make it a recurrent neural network, though a highly unusual one.

Exercise 9

9a

1. Fractionally strided convolutions. Size and stride. Ex. a 3x3 fractionally strided convolution with stride 2 adds length 2 to each dimension for each element in that dimension.

2. Upsampling. You can use kernels for this with a chosen size and stride. Ex. a 2x2 area can be turned into a 4x4 area using a 3x3 kernel with stride 1 and padding 2 by letting the kernel have its center outside the 2x2 area.

9b

The discriminator model in a GAN is tasked with differentiating generated/fake images from real images. A discriminator can be defined as a binary classifier. It takes an image as input and outputs a number between 0 and 1 which is the probability that the image is fake.

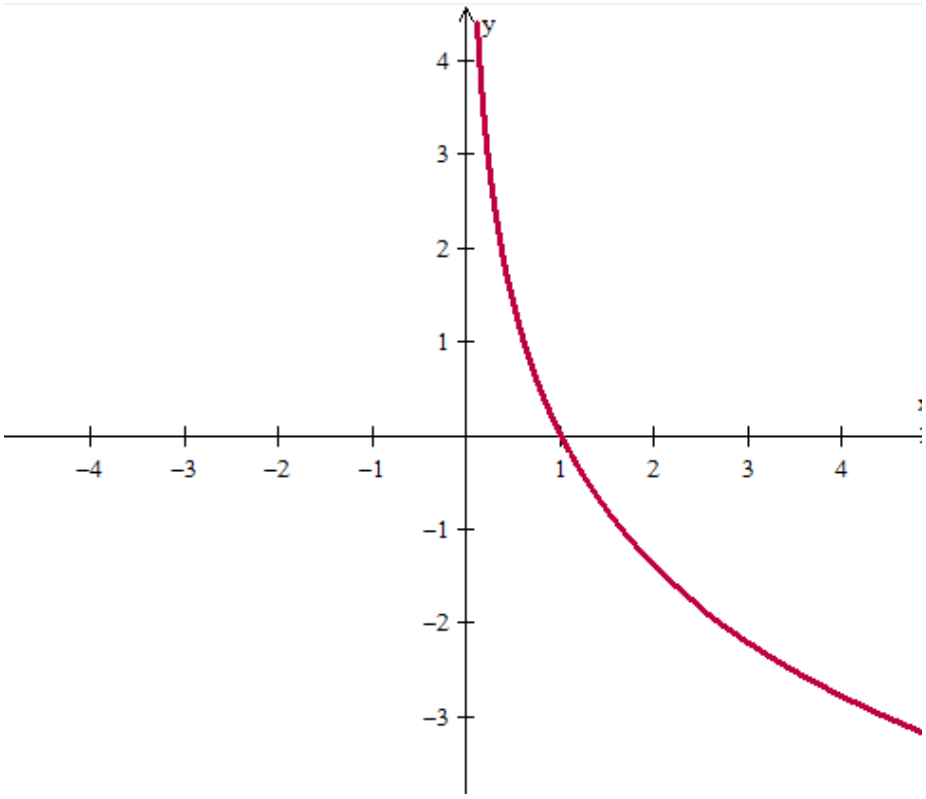
Another type of GAN uses a discriminator which is grown progressively in tandem with the generator, and not only trained. In this case the input size of the generator depends on the growth stage, at the start it can be 4x4 images and at the end it can be 1024x1024 images. More layers are added onto the model as the size increases. In any case, the output is still a probability between 0 and 1 of the input image being real.

9c

A possible choice of loss function for a generator is

$$L(G) = \frac{1}{b} \sum_{v_i \sim P_v}^{b} -\ln D(G(v_i)) \tag{3}$$

where v_i are the b different "seeds" used to generate images, G is the generator, L is the loss, D is the discriminator. The discriminator outputs what it thinks the probability is that the input image is real. We want this probability to be as large as possible, so we minimize its negative log over all generated images. If this loss is minimal (0), the Discriminator outputs 1 for all images. If it is close to minimal, the discriminator outputs close to 1 on average. This can be seen directly from the graph of $-ln(x)$, between 0 and 1.



Exercise 10

10a

The network output could be interpreted as follows

Index	Use	Values	Loss
1	Background	0-1	Cross Entropy
2	Bird	0-1	Cross Entropy
3	Balloon	0-1	Cross Entropy
4	Baboon	0-1	Cross Entropy
5	X pos	0-Xmax	MSE if not background
6	Y pos	0-Ymax	MSE if not background
7	Width	0-Xmax	MSE if not background
8	Height	0-Ymax	MSE if not background

Where X pos and Y pos are the X and Y coordinates of the center of the bounding box, and Width and Height are its width and height. All of these values should be scaled down to not dominate the loss function.

The loss for each element is indicated in the table. The class labels use a common cross entropy loss, just as in image classification, and the bounding box spatial values contribute their total mean squared error compared to the ground truth values, if there is an object.

10b

This is resolved by using different layers of the convolutional network for different size bounding boxes (early layers for small boxes where local features are important, and later layers for large boxes where contextual features are important), and then placing these possible boxes around the image at a limited number of locations and combining the scores of different classes for different possible bounding boxes to come up with a final answer.

Exercise 11

11a

Since block1, block2 and block3 are not trained, I will assume that the network weights have been taken from a pretrained network, and this code is an attempt at transfer learning.

In that case, since block4, block5 and block6 are being trained, but not the final fully connected layer, the network will perform badly as a result of the final fully connected layer not being trained to turn the high level features from the convolutions into a classification in the target domain.

11b

The first way I would fix this problem is to add the fully connected layer parameters to the list of trainable parameters.

I don't see a substantially different solution to this problem. A slight variant that could work is training only the last layer, though with such a small model it might be best to train the entire network, depending on the problem at hand.

Exercise 12

12a

I tried :)

We have

$$f \in \text{Lip}_1 \Leftrightarrow \forall x, y : \|f(x) - f(y)\| < \|x - y\| \tag{4}$$

For f_w we have

$$\begin{aligned} \|f(x) - f(y)\| &< \|x - y\| \\ \|0.3x_1 - 0.5x_2 - 0.3y_1 + 0.5y_2\| &< \|x_1 - y_1 + x_2 - y_2\| \\ \|(0.3x_1 - 0.3y_1) - (0.5x_2 - 0.5y_2)\| &< \|x_1 - y_1 + x_2 - y_2\| \end{aligned} \tag{5}$$

$$\nabla_{\omega_1} h(w) = \frac{1}{|A|} \sum_{x_i \in A} (x_{i,1} + \omega_2 x_{i,2}) - \frac{1}{|B|} \sum_{x_i \in B} (x_{i,1} + \omega_2 x_{i,2}) \tag{6}$$

$$\nabla_{\omega_1} h(w) = \frac{1}{|A|} \sum_{x_i \in A} (x_{i,2} + \omega_1 x_{i,1}) - \frac{1}{|B|} \sum_{x_i \in B} (x_{i,2} + \omega_1 x_{i,1}) \tag{7}$$