Machine Learning Homework 4

Arya Jalali

January 23, 2023

1 Modern CNN

1.1

Taking the number of channels to be fixed in both settings, we'll have the following two sets of parameter count for each model

$$4 \times (C+1)C + (9C+1)C + (25C+1)C = 38C^2 + 6C$$

$$4 \times (C+1)C + 3 \times (9C+1)C = 31C^2 + 7C$$

The graphs of the above parameter counts intersect each other at points $(0,0)$ and $(0.143, 1.633)$; therefore, the number of parameters in our first model will always be bigger for $C \geq 1$.

1.2

Because our input and output are the same size, for each "pixel" the number of computations we have to do is directly proportional to the number of our parameters; so we'll just be calculating the number of parameters of different models, and comparing them.

$$(9C+1)C = 9C^2 + C$$

$$2 \times (4C+1)C = 8C^2 + 2c$$

These two graphs intersect each other at points $(0,0)$ and $(1,10)$; therefore, we can write the following inequality

$$\#_1 \geq \#_2$$

1.3

we'll compare the result of this part with the previous one by factorizing a 3×3 convolution into a 1×3 kernel, followed by a 3×1 kernel.

$$2 \times (3C+1)C = 6C^2 + 2C$$

Clearly the second method gives us a fewer number of parameters compared to the first method.

1.4

Without doing any computations we can see applying an inception module to a bigger feature map increases our computational cost, But doing the pooling layer first creates a representational bottleneck, because there is information loss which could be useful in detecting patterns, on the other hand, applying the inception module first might helps us find the useful features and map them to a more sparse space. In this new space applying the pool function might not cause that much of an information loss, and is overall the better case if we want to pass as much valuable information as possible through this segment of the model.

2 Autoencoders

2.1

Many reasons can be given, and there is not a "definite answer". Size of our latent space might be too small. Our learning rate might be too high. We might have faced problems such as the vanishing or exploding gradient, our initialization might be the cause.

2.2

No necessarily. The model might have overfitted on our training data, or maybe the test and training data haven't been properly shuffled. We should compare our training error with our test error, and if there is a large gap, we should simplify the model or use different techniques to prevent overfitting.

2.3

The model lacks robustness. We can increase this metric using techniques such as addition of noise to our data, and adding a regularization term to our loss function.
Adding noise gives us a new loss function shown below

$$\mathcal{L} = \mathbb{E}_{x\sim D, \epsilon\sim\mathcal{N}}[(f(x+\epsilon)-x)^2]$$

$\mathcal{N}$ can be any distribution with a zero mean, and D is the distribution of our data.
Note that the loss defined above is the usual L2 loss which is usually used for autoencoders, but we can replace the equation inside the brackets with any other function we like.

2.4

Each vector has about 20,000 elements, and we trained our model on only 800 samples. The reason our model is able to easily reconstruct data with such a few number of samples and neurons is that there are probably a lot of correlation between the features, and the dimension of our data might not actually be 20,000.

3 Generative adversarial networks(Optional)

3.1

Using the chain rule we have

$$\frac{\partial L}{\partial \theta_d} = \frac{\partial L}{\partial D}\frac{\partial D}{\partial \theta_d} = \sum_{i=1}^{n} \frac{1}{D(X^i;\theta_d)}\Delta_{\theta_d} D(X^i) - \frac{1}{1-D(G(Z^i))}\Delta_{\theta_d} D((G(Z^i)))$$

3.2

$$D(x) = g_d^{L_d}(z_d^{L_d}(x))$$

Note that $D(x) = g(z(x))$ so we can rewrite the Loss function like below

$$L(\theta_d, \theta_g) = \sum_{i=1}^{n} log(g(z(X^i))) + log(1-g(z(G(Z^i))))$$

Taking the derivative gives us

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial g}\frac{\partial g}{\partial z} = \sum_{i=1}^{n} \frac{g^{'}(z(X^i))}{g(z(X^i))} - \frac{g^{'}(z(G(Z^i)))}{1-g(z(G(Z^i)))}$$

g is the sigmoid function.

$$g^{'}(z(X^i)) = g(z(X^i)) \times (1-g(z(X^i))), \quad g^{'}(z(G(Z^i))) = g(z(G(Z^i))) \times (1-g^{(}z(G(Z^i))))$$

Substituting the derived equalities allows us to write the final result in a simpler form

$$\frac{\partial L}{\partial z} = \sum_{i=1}^{n} 1 - g(z(X^i)) - g(z(G(Z^i))) = n - \sum_{i=1}^{n} g(z(X^i)) + g(z(G(Z^i)))$$

3.3

Mode collapse is when the GAN produces a small variety of images with many duplicates (modes). This happens when the generator is unable to learn a rich feature representation because it learns to associate similar outputs to multiple different inputs.
We can use the Wasserstein Loss described below

$$\Delta_w \frac{1}{m} \sum_{i=1}^{m} [f(x^i) - f(G(z^i))]$$

In Wasserstein GANs the discriminator does not actually classify instances. For each instance it outputs a number. This number does not have to be less than one or greater than 0, so we can't use 0.5 as a threshold to decide whether an instance is real or fake. Discriminator training just tries to make the output bigger for real instances than for fake instances.
Wasserstein GANs are less vulnerable to getting stuck than minimax-based GANs, and avoid problems with vanishing gradients. The earth mover distance also has the advantage of being a true metric: a measure of distance in a space of probability distributions. Cross-entropy is not a metric in this sense.

4 Sequence-to-Sequence Models Comprehension

4.1

If we have to stick with RNN's, we can use the old techniques we used in other ANN's (gradient clipping, changing the activation function, ...). If we are given the freedom to tweak the model a little, we can go for LTSM's. These modules pass different type of memories straight from the to without letting them go through an activation function which lets the gradient to flow freely in backpropagation.

4.2

In bi-directional LTSM unlike standard LSTM, the input flows in both directions, and it's capable of utilizing information from both sides.
This new architecture gives every component of an input sequence information from both the past and present. For this reason, bi-directional LTSM can produce a more meaningful output, combining LSTM layers from both directions

4.3

The power of this model lies in the fact that it can map sequences of different lengths to each other. Inputs and outputs are not correlated and their lengths can differ.

4.4

In the decoding phase of a sequence to sequence model we usually take the most probable option as the input for the next time step. This however can accumulate error, because if the model gives a wrong prediction at the first step it messes up the entire decoding. Using the beam search algorithm with a fixed beam size k at each time step we keep the top k options, and we keep doing this until the very end. In the end we have a set of possible candidates from which we can select the best possible combination.

5 RNN Calculation

5.1

First we try to build a simple perceptron that can tell if two bits are equal or not

Our perceptron consists of a single hidden layer and a single output node. We propose the following parameters for our network

$$W^{[1]} = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, b^{[1]} = \begin{bmatrix} -0.5 \\ 1.1 \end{bmatrix}$$

The output of our first layer will be a $\mathcal{R}^2$ vector, where the first element is 1 iff both our bits were 1, and the second element is 1 iff both our bits were 0.

$$W^{[2]} = v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, b^{[2]} = C_0 = 0.1$$

The second layer is just a simple OR gate.
Now that we can correctly check the equality of two bits, we'll extend our model to a sequence of bits. For two sequence of bits to be equal at time t, all of their bits up until $t-1$ should be 1. Defining y^t the equality of two sequences until the t'th bit helps us define the following recursive relation

$$y^t = g(v^T h^t + 3.25 y^{t-1} - 3)$$

For the above equation to be 1, y^{t-1} has to be bigger than 0, and $v^T h^t$ which indicates the equality of the t'th bits has to be 1 as well. Since the highest value for $v^t h^t$ our defined expression makes sense.

$$W = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, b = \begin{bmatrix} -0.5 \\ 1.1 \end{bmatrix} v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, C_0 = 0.1, C = -3, r = 3.25$$

6 Word Embedding

6.1

Skip-gram is one of the unsupervised learning techniques used to find the most related words for a given word. It is used to predict the context word for a given target word.
Context window is the number of words to be predicted which can occur in the range of the given word.
In skip-gram we are trying to "guess" or calculate the probability of the surrounding words of an input word; this is accomplished by finding all the pairs our word appears in, and trying to guess the words around it. Because this is a classification task, we use the Cross Entropy loss for our model.

6.2

Taking the derivative of our loss function (Cross entropy) with respect to y gives us

$$\frac{\partial L}{y_i} = \frac{1}{M}(\frac{1-y_i}{1-\hat{y}_i} - \frac{y_i}{\hat{y}_i}) \rightarrow \frac{\partial L}{y_i} = S(y)$$

$$\frac{\partial L}{\partial y} = \frac{\partial Softmax(y)}{\partial y} \frac{\partial L}{\partial Softmax(y)} = S^{'}(y) S(y)$$

Since we don't need $\frac{\partial L}{\partial W}$ we skip it

$$\frac{\partial L}{\partial h} = W^{'}(\frac{\partial L}{\partial y})$$

$$W^T x = h$$

Since x is one-hot encoded, only one of it's entries is 1.

$$x_j = 1 \rightarrow w_j^T = h \rightarrow \begin{cases} j! = k \rightarrow \frac{\partial L}{\partial w_j} = 0 \\ O.w \rightarrow h = w_k^T \rightarrow \frac{\partial L}{\partial h} = \frac{\partial L}{\partial w_k} = W^{'}(\frac{\partial L}{\partial y}) = W^{'} S^{'}(y) S(y) \end{cases}$$