**CS491/591 Deep Learning**

**Homework #6**

**1. What are benefits of using convolutional layers instead of fully connected ones for computer vision tasks?**

**Convolutional Layers**

We may achieve accuracy with fully connected layers for short epochs, but we aim to employ convolutional layers because they include pooling layers in addition to typical standard layers when longer epochs and huge datasets are required to be trained.

Because CNN architectures explicitly assume that the inputs are images, specific attributes can be encoded into the model architecture.

A basic CNN consists of a series of layers, each of which uses a differentiable function to transform one volume of activations into another. Convolutional Layer, Pooling Layer, and Fully Connected Layer are the three primary types of layers that make up the CNN architecture.

**Fully Connected Layers**

Every neuron in one layer is connected to every other neuron in the layer below it, forming a sequence of fully connected layers that make up a fully connected neural network.

Fully linked networks have the primary benefit of being "structure agnostic," meaning that no special assumptions about the input are required.

Fully linked networks can be used in a wide range of situations since they are structure-indifferent, but they typically perform worse than special-purpose networks designed to address a particular problem space's structure.

**2. Why is it important to place non-linearities between the layers of neural networks?**

It turns out that a neural network must select a non-linear activation function to compute interesting functions.

It indicates that the neural network can accurately forecast the class of a function that is divided by a nonlinear decision boundary or successfully approximation functions that do not follow linearity.

The model should be able to produce intricate mappings between the network's inputs and outputs with the help of a neural network with a non-linear activation function.

**Some non-linear activation functions**

**Sigmoid**

We employ the sigmoid function primarily because it occurs between (0 to 1). Since the probability of anything occurs only between the range of 0 and 1, sigmoid is the best option for models where we must anticipate the probability as an output.

A neural network may become stuck during training due to the logistic sigmoid function. As a result, the output layer is where it is most often used. (When a binary classifier is used)

Tanh

Tanh is a shifted variation of the sigmoid function, with a -1 to 1 range. The data is more centered because the mean of the activations that emerge from the hidden layer is closer to zero, which facilitates and accelerates learning for the following layer.

One drawback of both sigmoid and tanh is that the gradient (also known as the derivative or slope) of this function grows small and approaches zero if our weighted sum input(z) is either large or small. This may cause a steep descent to lag.

**3. What are the weight initialization techniques for neural networks?** **What is wrong with initializing all weights to the same number (e.g., zero)?**

A crucial design decision when creating deep learning neural network models is weight initialization.

Over the past ten years, more precise heuristics have been developed that make use of information like the type of activation function being utilized and the quantity of inputs to the node. Historically, weight initialization involved utilizing small random integers.

The stochastic gradient descent optimization approach can train neural network models more successfully because of these more specialized heuristics.

Weight initialization is as basic as dividing the integers into even and odd parts and assigning each one a weight of 0 or 1, respectively.

We create a series of data visualizations using random integers in such a way that each weight is represented on the graph.

An example of a range for a graphical representation is [- 1/ sqrt(n), 1/ sqrt(n)] or [- 1/ n, 1/ n].

Depending on the input we are processing, such as text, picture capture, or related sound patterns, we apply various activation functions in the layers of the neural network.

In a comparable situation, if we substitute zero for weights in place of the graphical representation produced by random values that fall on the graph, we will train datasets that are linearly symmetric and unable to produce accurate results.

**4. What are the main differences between a feedforward neural network and a recurrent neural network?**

Recurrent neural networks, as opposed to feedforward networks, have a single weight parameter that applies to all network layers. By modifying these weights via gradient descent and backpropagation, reinforcement learning can still be accomplished.

In contrast to recurrent neural networks, which continuously feed data from input to output, feedforward neural networks continuously feed data back into the input for additional processing and final output.

Recurrent neural networks provide a feedback loop that enables data to be reused as input before being sent once more for processing and output. Feedforward neural networks, however, merely forward data from the input to the output. Data in feedforward neural networks can only move in one direction. This forward traveling pattern prevents data from being saved from previous levels, hence there is no internal state or memory. RNN, on the other hand, cycles through the data using a loop, enabling it to maintain track of both new and old data.

**5. What are the inputs of the RNNs in each step?**

Recurrent neural networks (RNNs) are a type of neural network in which the results of one step are fed into the next step's computations. Traditional neural networks have inputs and outputs that are independent of one another, but there is a need to remember the previous words in situations where it is necessary to anticipate the next word in a sentence. As a result, RNN was developed, which utilized a Hidden Layer to resolve this problem. The Hidden state, which retains some information about a sequence, is the primary and most significant characteristic of RNNs.

RNNs have a "memory" that retains all data related to calculations. It executes the same action on all the inputs or hidden layers to produce the output, using the same settings for each input. In contrast to other neural networks, this minimizes the complexity of the parameter set.

**6. Is it possible to process long sequences using vanilla RNNs? If not, what is the reason and what could be the solution?**

To train the datasets, a plain RNN should be able to read the files in sequence and use the trained model to predict the output for the missing sequence given in the input. However, with the one-to-one Vanilla RNN, we can only conduct one-to-one operations without using sequences as input or output. We are limited to processing simple classifications and regression issues.

This is a blend of many-to-one and one-to-many architecture for the sequence-to-sequence models where you might want to perform things like machine translation. A variable-sized input, such as an English sentence, is received by the encoder, which performs encoding into a hidden state vector. The hidden state vector is then received by the decoder, which generates a variable-sized output. Utilizing this architecture is driven by its modularity. Encoders and decoders may easily be switched out to accommodate various language translations.

**7. Should we use the same weight matrixes in every time steps in the RNNs? Why?**

Recurrent neural networks (RNNs) are the most advanced algorithm for sequential data and are the foundation of Google voice search and Apple's Siri. Due to its internal memory, it is the first algorithm to recall its input, making it ideal for machine learning issues involving sequential data. It is one of the algorithms that helped deep learning accomplish some incredible successes over the past several years. It facilitates using examples of various lengths to apply the model to. If the RNN model employs different parameters for each step during training, it will not generalize to previously unknown sequences of various lengths while reading a sequence.

The sequences frequently follow the same set of rules throughout the sequence. As an illustration, in NLP

"On Monday it was snowing"

"It was snowing on Monday"

Despite the details being in various places in the sequence, these two phrases indicate the same thing. We do not need to re-learn the rules at each point in the phrase because parameter sharing reflects the fact that we are doing the same task at each stage.

**8. What is the difference between the backpropagation algorithm and backpropagation through time one?**

The Backpropagation Through Time (BPTT) technique applies the Backpropagation training method to recurrent neural networks trained on sequence data, such as time series. Each timestep, one input is presented to a recurrent neural network, which then predicts one output. The way BPTT operates conceptually is by unrolling each input timestep.

For recurrent neural networks, we effectively perform backpropagation across time, which entails computing gradients by first going backward through the entire sequence after computing losses through it.

But when we wish to train a long sequence, this becomes an issue. For instance, before computing a single straightforward gradient step, we must iterate through several layers after training a paragraph of text. Humans use a reduced backpropagation over time approximation. Run sections of the sequence back and forth rather than the entire thing.

Despite our input sequence being long or infinite, when we train our model, we will advance for a predetermined number of steps and only compute a loss over this subset of the data. Make a gradient step on the weights after backpropagating through this sub-sequence. We will pass this hidden state over to the subsequent batch of data since we still have it from the prior one. However, we shall just backpropagate via this second batch; the forward pass is unaffected.

**9. What is the vanishing gradient problem? How can we tackle this problem?**

The Basic RNN Block Has the Drawback That It Cannot Accurately Capture Long Range Sequences. When the gradient propagates but the derivatives are so small that it disappears, this is called the "vanishing gradient problem."

To reduce the problems, we will do the following:

To keep the gradients from exploding during model training, we shall set a cap on their number. The term for it is gradient clipping.

Setting the weights' starting values to identity matrices and zero for their biases will stop the weights from shrinking to zero. It is known as weight initialization in technical parlance.

**10.What are the three keys of the Long Short Term Memory operation?**

RNNs can recall inputs for a long time because of LSTMs. This is because LSTMs have a memory that stores information, much like a computer's memory. The LSTM can read, write, and delete data from its memory.

This memory can be thought of as a gated cell, where the cell determines the value, it assigns to the information whether to store or erase information (I.e., whether it opens the gates or not). Weights, which the algorithm also learns, are used to determine importance. This merely indicates that it gradually comes to understand what information is crucial and what is not.

There are three gates in a long short-term memory cell:

an input gate,

a forget gate,

and an output gate.

These gates decide whether to allow additional input (input gate), erase the data because it is unimportant (forget gate), or allow the information to affect the output at the current timestep (output gate).

The analog sigmoid that makes up an LSTM's gates have a range of zero to one. They can perform backpropagation since they are analog.

The issue of disappearing gradients is resolved by LSTM because it maintains gradients that are sufficiently steep, which keeps training time low and accuracy good.