National University of Singapore
School of Computing
CS2109S: Introduction to AI and Machine Learning
Semester I, 2023/2024

Tutorial 9
Convolutional Neural Networks and Recurrent Neural Networks

These questions will be discussed during the tutorial session. Please be prepared to answer them.

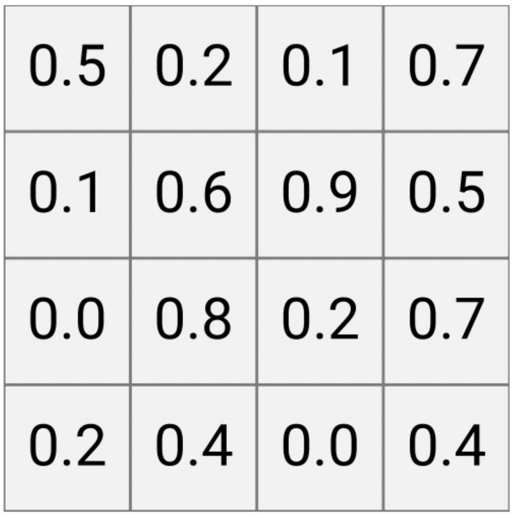
Summary of Key Concepts

In this tutorial, we will discuss and explore the following learning points from Lecture:

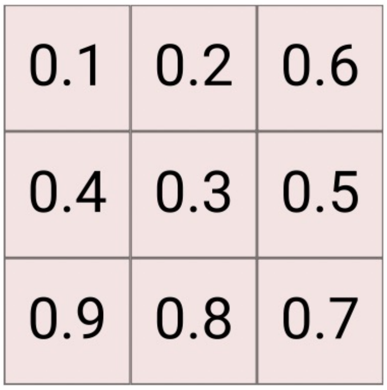
1. CNN
   (a) Convolution kernel
   (b) Padding and stride
   (c) Design choice
2. RNN Design choice
3. CNN vs RNN
4. Dying ReLU Problem

Problems

1. **ConvNets**
   (a) You are given an image $\mathbf{x} \in \mathbb{R}^{4\times4}$ and a filter $\mathbf{W} \in \mathbb{R}^{3\times3}$. **No** extra padding is added.



0.5	0.2	0.1	0.7
0.1	0.6	0.9	0.5
0.0	0.8	0.2	0.7
0.2	0.4	0.0	0.4



0.1	0.2	0.6
0.4	0.3	0.5
0.9	0.8	0.7

Figure 1: (left) The image $\mathbf{x}$. (right) The filter $\mathbf{W}$.

Get the output feature map from this convolution and write down its output dimensions if the kernel moved with a stride of 1×1. No Padding or Pooling operation required. Note that while mathematically, there is a need to flip the kernel matrix, in practice when using CNNs we do not flip the kernel. This saves on the cost of flipping while making little practical difference. For this question, we compute the output feature map with no flipping.

Solution: Output dimension: 2×2
Output feature map:
$$\begin{bmatrix} 1.60 & 2.59 \\ 1.51 & 1.91 \end{bmatrix}$$

(b) Now, let's say you have access to a very deep, large CNN model. We feed a single image to the network. Each image has $3(C)$ channels (RGB) with a height and width of 224×224 ($H = 224, W = 224$). Here our input is a 3-dimensional tensor ($H \times W \times C$). The first layer of the big CNN is a Convolutional Layer with 96 (C_1) kernels which has a height and width of 11, and each kernel has the same number of channels as the input channel. The stride is 4×4 and no padding is used. Calculate the output size $H_1 \times W_1 \times C_1$ after the first Convolutional Layer.

Solution:

For a single image and a single kernel, the output height $H_1 = \lfloor \frac{H-K+2P}{S} \rfloor + 1 = 54$. Similarly output height is $W_1 = 54$. Hence the output size is $H_1 \times W_1 \times C_1$. Here, H_1 and W_1 are the dimensions of the intermediate feature map, K is the kernel size, P is the padding, and S is the stride.

(c) In most of Deep Learning libraries such as PyTorch, images are fed together as a batch B. B can take values such as $8, 16, 32, 64$. Comment on the output shape if we feed the large CNN in part (b) with a batch of images. What are the advantages of using a batch of images rather than a single image?

Solution:

The output shape will be $B \times H_1 \times W_1 \times C_1$. Using a batch of images is computationally efficient and more stable in gradient descent convergence.

2. **RNN design choice**

(a) What type of RNN model does Image captioning use? What characteristics make this a standard model for Image captioning?

Solution:

One-to-many model. One-to-many sequence problems are sequence problems where the input data has one time-step, and the output contains a vector of multiple values or multiple time-steps. Thus, we have a single input and a sequence of outputs. Input: One image. Output: Multiple words as captions.

(b) What type of RNN model does Text classification use? What characteristics make this a standard model for Text classification?

Solution:

Many-to-one model. In many-to-one sequence problems, we have a sequence of data as input, and we have to predict a single output. Sentiment analysis or text classification is one such use case. Input: Many words. Output: Which class this text belongs to.

(c) What type of RNN model does Language translation use? What characteristics make this a standard model for Language translation?

Solution:

Many-to-many model. Many-to-Many sequence learning can be used for machine translation where the input sequence is in some language, and the output sequence is in some other language. Input: Many words / code of language A. Output: Many words / code of language B.

3. **CNN vs RNN** Describe how you can use the deep learning models learnt in class to tackle the following problems.

(a) Let's apply what you learnt to a specific example, performing sentiment analysis on Covid-19 posts on twitter. Explain what characteristics of RNN make it a standard model for sentiment analysis and which RNN model you want to use to tackle this problem.

Solution:

The Recurrent neural network method is the standard method for dealing with any sequential (e.g., time series) input. As the final state of the RNN encodes the representation of the entire sentence, we use this final state to yield a classification of the sentiment of the sentence. This corresponds to a many-to-one RNN model.

(b) Given the same context in part (a), would it be possible to perform sentiment analysis using CNN? Explain why or why not.

Solution:

The key thing about sentiment analysis is that we do not just need to capture what words appear, but we also need to capture the general context of the whole sentence. The CNN model is also capable of doing sentence classification because as we add more convolution layers and make the network deeper, we will be able to detect higher-level features and capture the general context of the whole sentence.

(c) Now let's talk about another application - image recognition. Suppose we now want to recognize whether the image contains Chihuahua or muffin, briefly explain why CNN is good for image recognition.

Solution:

CNN allows us to exploit spatial structure. Convolution allows us to capture not just one pixel, but a group of pixels at a time; different convolution filters can be used to extract different features; and methods like max pooling can be used to capture higher level features and reduce dimensionality. Last but not least, kernels are learned automatically through weight updates.

(d) In part (b), we've looked at ConvNets for sequences (sentiment analysis of text, for instance). Now, let's examine RNNs for image processing.

"Tokens" are what constitute a sequence; words are tokens in sentences, for instance. Usually, when it comes to words, we represent them as vectors. We also know RNNs take in sequences (temporally-related collection of tokens). If we had to treat an image as a sequence, what would you do to allow RNNs to classify images? In other words, how can you tokenize an image? Your final method of "tokenizing" an image should ensure an RNN can be fed these tokens as inputs.

Express your creativity in coming up with different ways to go about doing so.

Solution:

Possible ways are to treat each row/column as a token, or use overlapping / non-overlapping patches, or treat windows of 10 pixels as a token. Among these, the best (as in, practical) way is to use patches as they retain locality, allowing important features (like ears or eyes) to be kept together.

4. **Dying ReLU Problem** This problem occurs when majority of the activations are 0 (meaning the underlying pre-activations are mostly negative), resulting in the network dying midway. The gradients passed back are also 0 which leads to poor gradient descent and hence, poor learning. How does `Leaky ReLU` fix this? What happens if we set $a = 1$ in the Leaky ReLU? Refer to Figure 2.

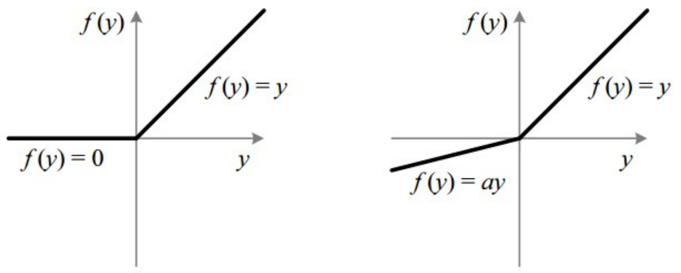


Figure 2: The Rectified Linear Unit (`ReLU`) (left) vs The Leaky Rectified Linear Unit (`Leaky ReLU`) with a as the slope when the values are negative. (right)

Solution:

If a ReLU activation is dead, it will always output 0 for any input. When a ReLU unit ends up this in state, it is unlikely to recover because the function gradient at 0 is 0. Leaky ReLU has a small positive gradient a for negative inputs. Having a small positive gradient for the negative inputs gives the network a chance to recover. However, this depends on the dataset being used and the value of a

being set prior to training. When $a = 1$, our activation function simply becomes a linear function and therefore our neural network layers simply degenerates back to single matrix multiplication as shown back in Tut 7.