Handout for Hw1 (CMPT 412/762)

In this assignment, you will implement a convolutional neural network (CNN). You will be building a numeric character recognition system trained on the MNIST dataset[1].

We begin with a brief description of the architecture and its functions. For more details, you can refer to online resources such as http://cs231n.stanford.edu. Note that this assignment requires much less coding than the other assignments. In the future, we will not provide detailed instructions; students are expected to search online and/or reverse-engineer template code.

A typical convolutional neural network has five different types of layers.

- Fully Connected Layer / Inner Product Layer (IP)
- Convolutional Layer
- Pooling Layer
- Activation Layer
- Loss Layer

1 Fully Connected Layer / Inner Product Layer (IP)

The fully connected or inner product layer is the simplest layer that makes up neural networks. Each neuron of the layer is connected to all the neurons of the previous layer (refer to Fig. 1). Mathematically it is modeled by a matrix multiplication and the addition of a bias term. For a given input x the output of the fully connected layer is given by the following equation,

$$f(x) = Wx + b$$

W, b are the weights and biases of the layer respectively. W is a two-dimensional matrix of $m \times n$ size where n is the dimensionality of the previous layer and m is the number of neurons in this layer. b is a vector with size $m \times 1$.

2 Convolutional Layer

This is the fundamental building block of CNNs. As we saw in the lecture, convolution is performed using a $k \times k$ filter and a $W \times H$ image. The output of the convolution operation is a feature map. This feature map can bear different meanings according to the filters being used - for example, using a *Gaussian filter* will lead to a blurred version of the image. Using the *Sobel filters* in the x and y direction gives us the corresponding edge maps as outputs.

When we perform convolution, we decide the exact type of filter we want to use and accordingly decide the filter weights. With CNNs, we try to learn these filter weights and biases from the training data. We attempt to learn a set of filters for each convolutional layer. In general, there are two main motivations for using convolution layers instead of fully- connected (FC) layers (as used in neural networks).

[1] https://en.wikipedia.org/wiki/MNISTdatabase

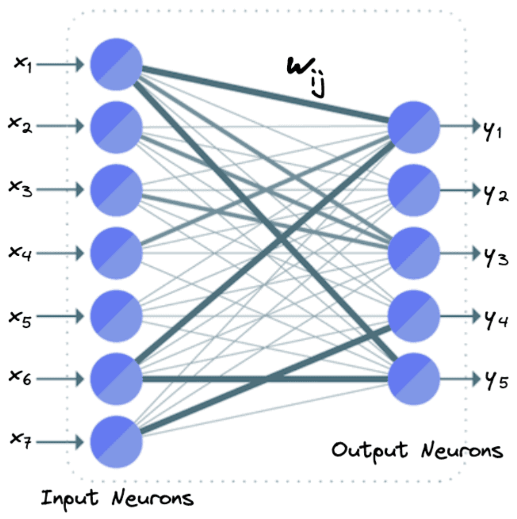


Figure 1: Fully connected / Inner Product layer

- **Reduction in parameters** – In FC layers, every neuron in a layer is connected to every neuron in the previous layer. This leads to a large number of parameters to be estimated - which leads to over-fitting. CNNs change that by sharing weights (the same filter is translated over the entire image).
- **Exploits spatial structure** – Images have an inherent 2D spatial structure, which is lost when we unroll the image into a vector and feed it to a plain neural network. Convolution by its very nature is a 2D operation that operates on pixels that are spatially close.

  The general convolution operation can be represented by the following equation:

  $$f(X, W, b) = X \circledast W + b$$

  where W is a filter of size $k \times k \times C$, X is an input volume of size $N \times N \times C$ and b is 1×1 element.

Terminology N_i represents the width and height of the input feature map, C_i represents the number of channels in the input feature map, k represents the width and height of the filter, s represents the stride of the convolution, p represents the number of padding pixels for the input feature map and C_o represents the number of convolution filters to be learned.

The height and width of the output feature map can be computed as:

$$N_o = \frac{N_i - k + 2p}{s} + 1$$

where subscript i represents input layer and o represents output layer. In summary, the input to the convolutional layer is a volume with dimensions $N_i \times N_i \times C_i$ and the output is a volume of size $N_o \times N_o \times C_o$ (refer to Fig. 2).

3 Pooling layer

A pooling layer is generally used after a convolutional layer to reduce the size of the feature maps. The pooling layer operates on each feature map separately and replaces a local region of the feature

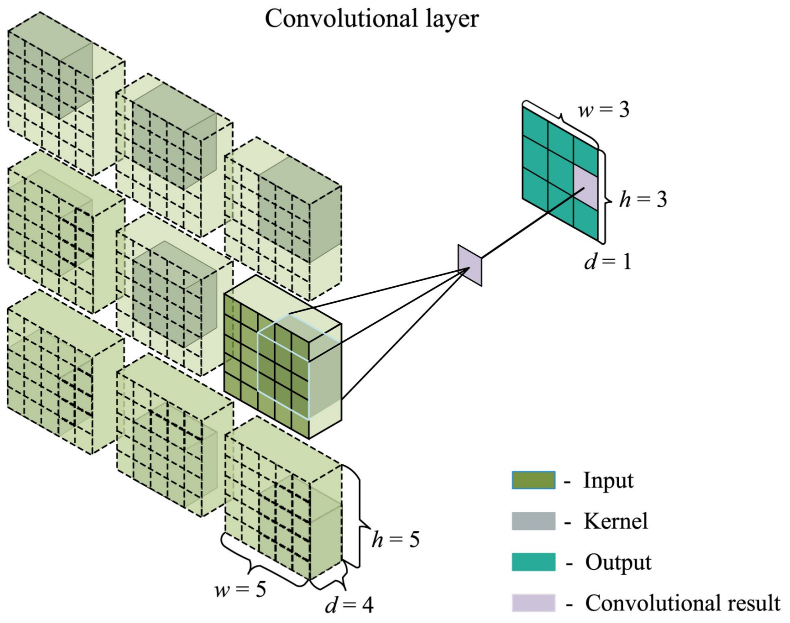


Figure 2: Convolutional layer

map with some aggregating statistics like maximum or average. In addition to reducing the size of the feature maps, it also makes the network invariant to small translations. This means that the output of the layer doesn't change when the object moves a little.

In this assignment, we will use only a max-pooling layer as shown in Fig. 3. This operation is performed in the same fashion as a convolution, but instead of applying a filter, we find the max value in each kernel. Let k represent the kernel size, s represent the stride and p represent the padding. Then the output of a pooling function f applied to a padded feature map X is given by:

$$f(X, i, j) = \max_{x \in [i-k/2, i+k/2],\, y \in [j-k/2, j+k/2]} X[x, y]$$

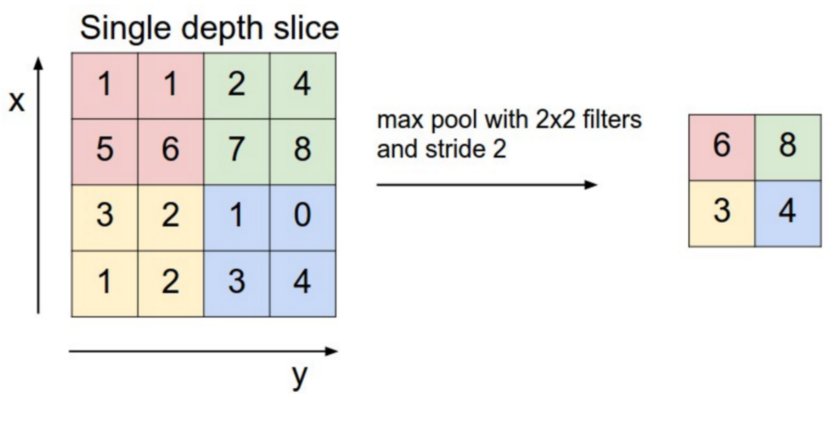


Figure 3: 2D Max Pooling layer

4 Activation layer

Activation layers introduce the non-linearity in the network and give the power to learn complex functions. The activation layers don't have any parameters and directly apply some function on the input feature map. The most commonly used non-linear functions are Rectified Linear Unit (ReLU), sigmoid, and tanh.

ReLU activation: In this assignment, we primarily use the ReLU activation function. ReLU activation functions are commonly used for mapping input feature values to the range $[0, \infty)$. ReLU function is defined as follows,

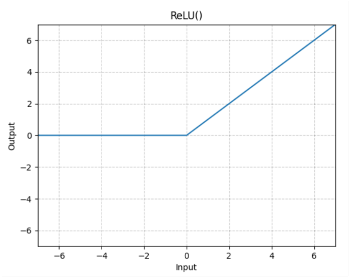
$$ReLU(x) = max(x, 0)$$

Sigmoid activation: For the bonus part, we train a network with the sigmoid activation function. Sigmoid(σ) activation functions are commonly used for mapping input feature values to the range $(0, 1)$. Sigmoid function is defined as follows,

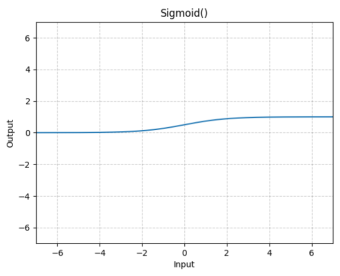
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Tanh activation: For the bonus part, we also train a network with the tanh activate function. Tanh activation functions are commonly used for mapping input feature values to the range $(-1, 1)$. Tanh function is defined as follows,

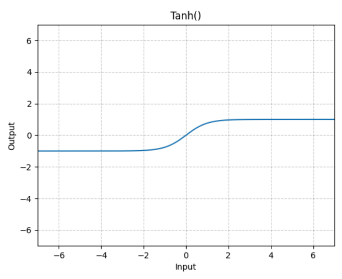
$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



(a) 1D ReLU activation



(b) 1D sigmoid activation



(c) 1D tanh activation

Figure 4: Different activation functions

5 Loss layer

The loss layer has a fully connected layer with the same number of neurons as the number of classes. And then to convert the output to a probability score, a softmax function is used. This operation is given by,

$$p = softmax(Wx + b)$$

where, W is of size $C \times n$ where C is the dimensionality of the previous layer and n is the number of classes in the dataset.

This layer also computes a loss function which is to be minimized in the training process. The most common loss functions used in practice are cross entropy and negative log-likelihood. In this assignment, we will just minimize the negative log probability of the given label.

6 LeNet architecture

In this assignment, we will use a simple architecture based on a very popular network called the LeNet[2]. The network takes as input a grayscale image of shape $1 \times 28 \times 28$. The network consists of the following layers:

- Convolution - $k = 5, s = 1, p = 0, C = 20$
- ReLU
- Max Pooling - $k = 2, s = 2, p = 0$
- Convolution - $k = 5, s = 1, p = 0, C = 50$
- ReLU
- Max Pooling - $k = 2, s = 2, p = 0$
- Fully Connected layer - 500 neurons
- ReLU
- Loss layer - 10 classes

Note that all types of deep networks use non-linear activation functions for their hidden layers. If we use a linear activation function, then the hidden layers has no effect on the final results, which would become the linear (affine) functions of the input values, which can be represented by a simple 2-layer neural network without hidden layers.

There are a lot of standard Convolutional Neural Network architectures used in the literature, for instance, AlexNet, VGG-16, or GoogLeNet. They are different in the number of parameters and their configurations.

[2] http://ieeexplore.ieee.org/abstract/document/726791/