CS5542 Big Data Analytics and Apps

Problem Set-4 (PS-4)

Deadline: April 5 (Th), 2018

Submit a hard copy of your solutions to the instructor during the class

Name:

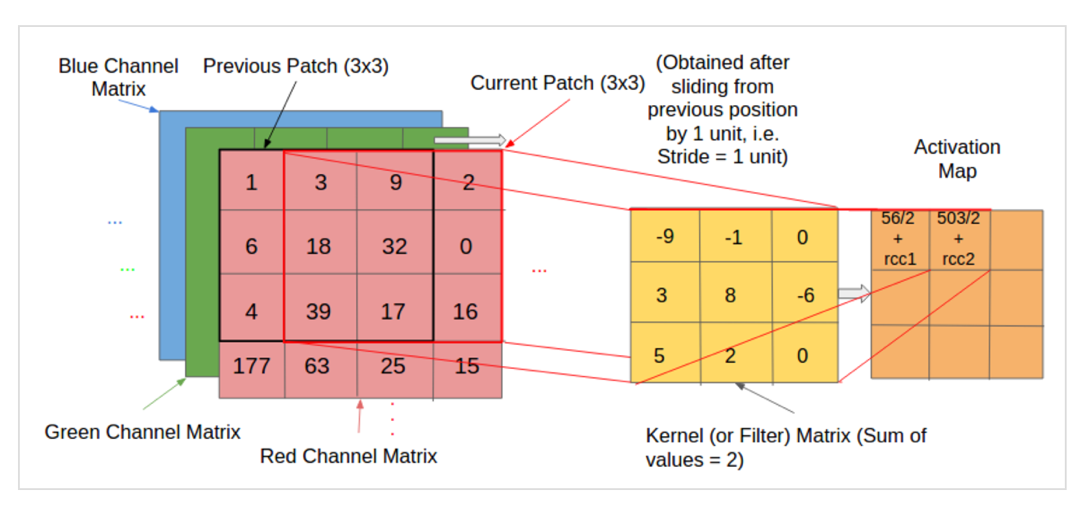
Class ID:

1. **Convolutional Neural Networks**

Answer the following questions by referring to the following site.

<http://xrds.acm.org/blog/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/>

1. Explain the overall process of the Convolutional Neural Network using the following example consisting of a 3-channel 4×4-pixels input image and a 3×3 kernel matrix.



The process is demonstrated using a toy example consisting of a 3-channel 4×4-pixels input image and a 3×3 kernel matrix. Note that:

pixels are numbered from 1 in the example;

the values in the activation map are normalized to ensure the same intensity range between the input volume and the output volume. Hence, for normalization, we divide the calculated value for the ‘red’ channel by 2 (the sum of values in the kernel matrix);

we assume the same kernel matrix for all the three channels, but it is possible to have a separate kernel matrix for each colour channel;

for a more detailed and intuitive explanation of the convolution operation, you can refer to the excellent blog-posts by Chris Olah and by Tim Dettmers.

1. Explain each step
   1. First, describe the Input/Output Volumes for CNN. Explain with an example of the cross-section of an input volume of size: 4 x 4 x 3.

CNNs are usually applied to image data. Every image is a matrix of pixel values. The range of values that can be encoded in each pixel depends upon its bit size. Most commonly, we have 8 bit or 1 Byte-sized pixels. Thus the possible range of values a single pixel can represent is [0, 255]. However, with coloured images, particularly RGB (Red, Green, Blue)-based images, the presence of separate colour channels (3 in the case of RGB images) introduces an additional ‘depth’ field to the data, making the input 3-dimensional. Hence, for a given RGB image of size, say 255×255 (Width x Height) pixels, we’ll have 3 matrices associated with each image, one for each of the colour channels. Thus the image in it’s entirety, constitutes a 3-dimensional structure called the Input Volume (255x255x3).

* 1. Second, describe the features & feature selection

Just as its literal meaning implies, a feature is a distinct and useful observation or pattern obtained from the input data that aids in performing the desired image analysis. The CNN learns the features from the input images. Typically, they emerge repeatedly from the data to gain prominence. As an example, when performing Face Detection, the fact that every human face has a pair of eyes will be treated as a feature by the system, that will be detected and learned by the distinct layers. In generic object classification, the edge contours of the objects serve as the features.

* 1. Third, describe the filters (convolution kernels)
     1. What are the activation maps?

The kernels are then convolved with the input volume to obtain so-called ‘activation maps’. Activation maps indicate ‘activated’ regions, i.e. regions where features specific to the kernel have been detected in the input.

* + 1. What does mean by the network learning?

The real values of the kernel matrix change with each learning iteration over the training set, indicating that the network is learning to identify which regions are of significance for extracting features from the data.

* 1. Fourth, describe the kernel operations.
     1. What does mean by convolving a Kernel?

The exact procedure for convolving a Kernel (say, of size 16 x 16) with the input volume (a 256 x 256 x 3 sized RGB image in our case) involves taking patches from the input image of size equal to that of the kernel (16 x 16), and convolving (or calculating the dot product) between the values in the patch and those in the kernel matrix.

* + 1. What is the patch?

The patch is the input to the kernel

* + 1. What is the stride?

The patch selection is then slided (towards the right, or downwards when the boundary of the matrix is reached) by a certain amount called the ‘stride’ value, and the process is repeated till the entire input image has been processed.

* + 1. Explain how to convolve between the values in the patch and those in the kernel matrix, e.g., a Kernel (size 16 x 16) with the input volume (a 256 x 256 x 3 sized RGB image in the case).
    2. How is the patch selection conducted?

The patch selection is then slided (towards the right, or downwards when the boundary of the matrix is reached) by a certain amount called the ‘stride’ value, and the process is repeated till the entire input image has been processed. The process is carried out for all colour channels.

* + 1. When is the process complete?

The process is repeated till the entire input image has been processed.

* + 1. When and why do we need the normalization?

Consider how a neural network learns its weights. C(NN)s learn by continually adding gradient error vectors (multiplied by a learning rate) computed from backpropagation to various weight matrices throughout the network as training examples are passed through.

The thing to notice here is the "multiplied by a learning rate".

If we didn't scale our input training vectors, the ranges of our distributions of feature values would likely be different for each feature, and thus the learning rate would cause corrections in each dimension that would differ (proportionally speaking) from one another. We might be over compensating a correction in one weight dimension while undercompensating in another.

This is non-ideal as we might find ourselves in a oscillating (unable to center onto a better maxima in cost(weights) space) state or in a slow moving (traveling too slow to get to a better maxima) state.

It is of course possible to have a per-weight learning rate, but it's yet more hyperparameters to introduce into an already complicated network that we'd also have to optimize to find. Generally learning rates are scalars.

Thus we try to normalize images before using them as input into NN (or any gradient based) algorithm.

* + 1. How is normalization going to work?

For normalization purposes, we divide the calculated value of the activation matrix by the sum of values in the kernel matrix.

## Describe about the main CNN hyperparameters.

## receptive field (R),

It is impractical to connect all neurons with all possible regions of the input volume. It would lead to too many weights to train, and produce too high a computational complexity. Thus, instead of connecting each neuron to all possible pixels, we specify a 2 dimensional region called the ‘receptive field[14]’ (say of size 5×5 units) extending to the entire depth of the input (5x5x3 for a 3 colour channel input), within which the encompassed pixels are fully connected to the neural network’s input layer. It’s over these small regions that the network layer cross-sections (each consisting of several neurons (called ‘depth columns’)) operate and produce the activation map.

## zero-padding (P),

Zero-padding refers to the process of symmetrically adding zeroes to the input matrix. It’s a commonly used modification that allows the size of the input to be adjusted to our requirement. It is mostly used in designing the CNN layers when the dimensions of the input volume need to be preserved in the output volume.

## the input volume dimensions (Width x Height x Depth, or W x H x D )

In CNNs, the properties pertaining to the structure of layers and neurons, such spatial arrangement and receptive field values, are called hyperparameters. Hyperparameters uniquely specify layers. The main CNN hyperparameters are receptive field (R), zero-padding (P), the input volume dimensions (Width x Height x Depth, or W x H x D ) and stride length (S).

## stride length (S).

The patch selection is then slided (towards the right, or downwards when the boundary of the matrix is reached) by a certain amount called the ‘stride’ value.

## the CNN Architecture

## describe about the **Convolutional Layer**

## what is the receptive field?

It is impractical to connect all neurons with all possible regions of the input volume. It would lead to too many weights to train, and produce too high a computational complexity. Thus, instead of connecting each neuron to all possible pixels, we specify a 2 dimensional region called the ‘receptive field[14]’ (say of size 5×5 units) extending to the entire depth of the input (5x5x3 for a 3 colour channel input), within which the encompassed pixels are fully connected to the neural network’s input layer. It’s over these small regions that the network layer cross-sections (each consisting of several neurons (called ‘depth columns’)) operate and produce the activation map.

* + 1. for an input image of dimensions 28x28x3, if the receptive field is 5 x 5, then What is the size of the region in the input volume where each neuron is connected in the Conv. layer (the region always comprises the entire depth of the input, i.e. all the channel matrices)? How many weighted inputs are there in each neuron?

Each neuron is connected to a certain region of the input volume called the receptive field (explained in the previous section). For example, for an input image of dimensions 28x28x3, if the receptive field is 5 x 5, then each neuron in the Conv. layer is connected to a region of 5x5x3 (the region always comprises the entire depth of the input, i.e. all the channel matrices) in the input volume. Hence each neuron will have 75 weighted inputs.

* + 1. Describe about the shared weights model. What is the depth column? What is the depth slice?

For optimized Conv. layer implementations, we may use a Shared Weights model that reduces the number of unique weights to train and consequently the matrix calculations to be performed per layer. In this model, each ‘depth slice’ or a single 2-dimensional layer of neurons in the Conv architecture all share the same weights. The caveat with parameter sharing is that it doesn’t work well with images that encompass a spatially centered structure (such as face images), and in applications where we want the distinct features of the image to be detected in spatially different locations of the layer.

* + 1. What do a feed-forward network and a Back-propagation algorithm with 3-dimensional arrangements of neurons, respectively?

We must keep in mind though that the network operates in the same way that a feed-forward network would: the weights in the Conv layers are trained and updated in each learning iteration using a Back-propagation algorithm extended to be applicable to 3-dimensional arrangements of neurons.

## Describe about the **ReLu (Rectified Linear Unit) Layer**.

## Describe how to compute the outputs of the CNN neurons using the derivative of the softplus function:

ReLu refers to the Rectifier Unit, the most commonly deployed activation function for the outputs of the CNN neurons. Mathematically, it’s described as:

[CodeCogsEqn (3)](http://xrds.acm.org/blog/wp-content/uploads/2016/06/CodeCogsEqn-3.png)

Unfortunately, the ReLu function is not differentiable at the origin, which makes it hard to use with backpropagation training. Instead, a smoothed version called the Softplus function is used in practice:

[CodeCogsEqn (4)](http://xrds.acm.org/blog/wp-content/uploads/2016/06/CodeCogsEqn-4.png)

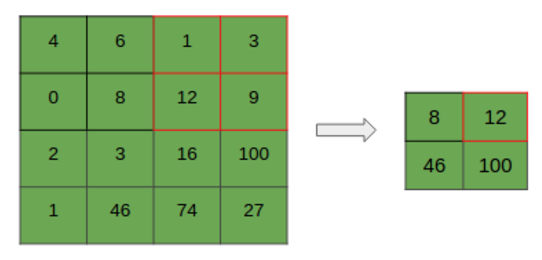
The derivative of the softplus function is the sigmoid function, as mentioned in a prior blog post.

[CodeCogsEqn (5)](http://xrds.acm.org/blog/wp-content/uploads/2016/06/CodeCogsEqn-5.png)

* 1. Describe about the **Pooling layer**
     1. explain about the Pooling Layer using the following example

Much like the convolution operation performed above, the pooling layer takes a sliding window or a certain region that is moved in stride across the input transforming the values into representative values. The transformation is either performed by taking the maximum value from the values observable in the window (called ‘max pooling’), or by taking the average of the values. Max pooling has been favoured over others due to its better performance characteristics.

The operation is performed for each depth slice. For example, if the input is a volume of size 4x4x3, and the sliding window is of size 2×2, then for each color channel, the values will be down-sampled to their representative maximum value if we perform the max pooling operation.

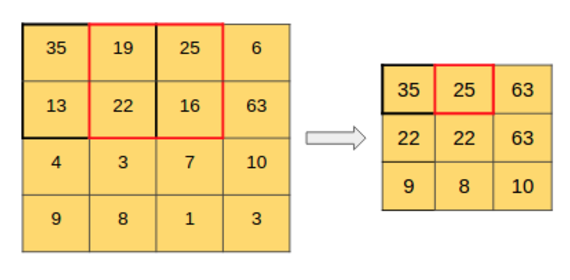


* + 1. What are the benefits of the down-sampling?

The operation performed by this layer is also called ‘down-sampling’, as the reduction of size leads to loss of information as well. However, such a loss is beneficial for the network for two reasons: the decrease in size leads to less computational overhead for the upcoming layers of the network; it work against over-fitting.

* + 1. Show how the operation can be applied for a stride value of [1,1], resulting in a 3×3 matrix

The operation is performed for each depth slice. For example, if the input is a volume of size 4x4x3, and the sliding window is of size 2×2, then for each color channel, the values will be down-sampled to their representative maximum value if we perform the max pooling operation.



## Describe about **the Fully Connected Layer**

The Fully Connected layer is configured exactly the way its name implies: it is fully connected with the output of the previous layer. Fully-connected layers are typically used in the last stages of the CNN to connect to the output layer and construct the desired number of outputs.

1. **Hyperparameters & Optimization**

Answer the following questions by referring to the following sites.

<https://databricks.com/blog/2016/01/25/deep-learning-with-apache-spark-and-tensorflow.html>

<http://stats.stackexchange.com/questions/164876/tradeoff-batch-size-vs-number-of-iterations-to-train-a-neural-network>

1. What's a hyperparameter?

A machine learning model is the definition of a mathematical formula with a number of parameters that need to be learned from the data. That is the crux of machine learning: fitting a model to the data. This is done through a process known as model training. In other words, by training a model with existing data, we are able to fit the model parameters.

However, there is another kind of parameters that cannot be directly learned from the regular training process. These parameters express “higher-level” properties of the model such as its complexity or how fast it should learn. They are called hyperparameters. Hyperparameters are usually fixed before the actual training process begins.

1. How does Hyperparameter tuning for optimization?

The TensorFlow library automates the creation of training algorithms for neural networks of various shapes and sizes. The actual process of building a neural network, however, is more complicated than just running some function on a dataset. There are typically a number of very important hyperparameters (configuration parameters in layman’s terms) to set, which affects how the model is trained. Picking the right parameters leads to high performance, while bad parameters can lead to prolonged training and bad performance. In practice, machine learning practitioners rerun the same model multiple times with different hyperparameters in order to find the best set. This is a classical technique called hyperparameter tuning.

1. What is the Number of neurons in each layer? How to do turning for this hyperparameter?

Number of neurons in each layer: Too few neurons will reduce the expression power of the network, but too many will substantially increase the running time and return noisy estimates.

1. What is the Learning rate? How to do turning for this hyperparameter?

If it is too high, the neural network will only focus on the last few samples seen and disregard all the experience accumulated before. If it is too low, it will take too long to reach a good state.

1. Discuss about the tradeoff curve for neural networks

The learning rate is critical: if it is too low, the neural network does not learn anything (high test error). If it is too high, the training process may oscillate randomly and even diverge in some configurations.

The number of neurons is not as important for getting a good performance, and networks with many neurons are much more sensitive to the learning rate. This is Occam’s Razor principle: simpler model tend to be “good enough” for most purposes. If you have the time and resource to go after the missing 1% test error, you must be willing to invest a lot of resources in training, and to find the proper hyperparameters that will make the difference.

1. Discuss about tradeoff batch size vs. number of iterations to train a neural network. Assuming that we train the neural network with the same amount of training examples, how to set the optimal batch size and number of iterations?

1. Batch Size

Batch size in mainly depended to your memory in GPU/RAM. Most time it is used power of two (64,128,256). I always try to choose 256, because it works better with SGD. But for bigger network I use 64.

2. Number of Iterations

Number if iterations set number of epoch of learning. Here I will use MNIST example to explain it to you:

Training: 60k, batch size: 64, maximum\_iterations= 10k. So, there will be 10k\*64 = 640k images of learning. This mean, that there will be 10.6 of epochs.(Number if epochs is hard to set, you should stop when net does not learn any more, or it is overfitting)

Val: 10k, batch size: 100, test\_iterations: 100, So, 100\*100: 10K, exacly all images from validation base.