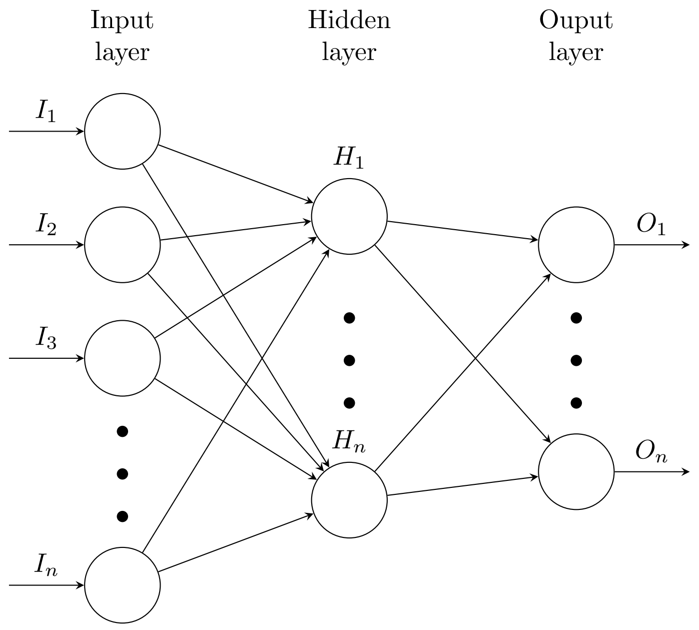
Convolutional Neural Networks

jawwad.shamsi and Rizwan Qureshi

March 2021

1 CNN Contd



Last week, we studied CNN. This week we will see a few examples of CNN and some fundamental concepts about DNN

1.1 Stochastic gradient descent

A variant of gradient descent algorithm in which the concept of mini batch is utilized. Large amount of resources are needed for large batch sizes. SGD algorithm employs mini batches for better utilization of resources.

1.2 Mini Batch

Instead of training on the whole dataset, we train in minibatches, where a mini batch is a smaller group of data samples. Each mini batch has a local minima. A global minima is computed from all the local minimas. This is specifically true for non-concave functions. Figure 1 illustrates the difference between a concave function and a non-concave function.

1.3 Normalization

It refers to changing (or scaling) the numerical values of the data set to put them along the same scale.

This is needed if the values in the input data are along different scales. For instance, age and salary. If we do not normalize data then salary is likely to be dominated.

standardization (or normalization) = z=x-m/S for each data point x, we subtract it from median and divide it by standard deviation.

The standardize data now has a mean of 0 and standard deviation of 1. Why do we do this? Some data points may be very high other may be very low.

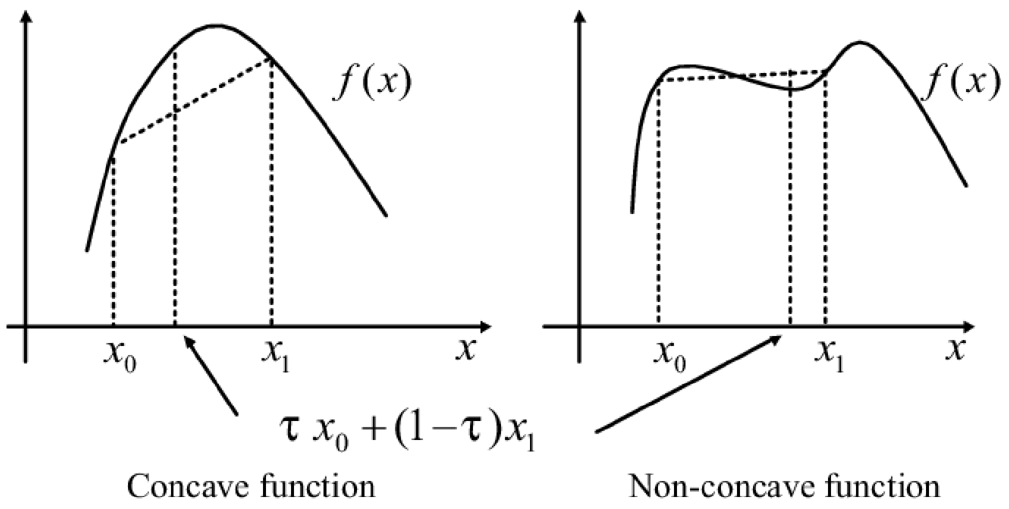


Figure 1: Concave vs Non-concave Function

Large variation in inputs may cascade to initial layers and cause imbalance gradient and exploding gradient problem Normalized data means that data is trained at high speed as well and will also cater the above problem

1.4 Vanishing/exploding gradient problem

slopes or derivatives can either be very big or very small. Recall that cost J(w) (the difference between the estimated output and the actual output) is reduced through back propagation. In that each parameter (W) is adjusted in proportion to it's gradient with respect to cost. For instance, if the gradient is large and +ve then it implies that the parameter has high correlation with the cost and decreasing it will decrease the cost. As we move from the final hidden layer to the first hidden layer, gradient is flattened and it vanishes. The farther the layer is from the output, the more slowly it tends to learn. If we keep adding more hidden layers, the farther layers, will eventually stop learning. This is specifically true for sigmoid (and similar) active functions which has a output space between 0 and 1. Figure 2 shows sigmoid and it's derivative. When the input to the sigmoid activation function is large (or small), it's output is close to one (or zero), and the derivative is close to zero. For neural networks with only a few hidden layers, this is acceptable. However, for large neural networks, the gradient could be too small. If we recall, that through the chain rule, the gradient is multiplied. For initial layers, this could be a bigger problem. When n hidden layers use an activation, n small derivatives are multiplied together. With small gradient, the value becomes too low for initial layers. This implies that the weights and biases of the initial layers will not be updated for effective range during the training phase. This can lead to overall inefficiency.

1.5 Batch Normalization

Often refers as BN as well. A technique to cater vanishing gradient problem.

Internal Covariate shift The distribution of hidden layers may gradually move around. Distribution of the inputs to layers somewhere down in the network may change after each mini-batch when the weights are refreshed.

weights are updated over each epoch. what if one of the weights becomes drastically larger over other weights. this will cause o/p from the corresponding neuron to be large and will cause instability and will cascade through the neural network

For BN, we use mean and SD over the batch instead of the complete data set.

BN is applied to layers which we chose to apply to. The purpose is to normalize the o/p from activation fn. Following are the steps:

1. mean of batch and SD z=(x-m)/s.
2. multiply the o/p to an arbitrary parameter g. z*g
3. add arbitrary parameter b to the resulting product (z*g) +b. sets new sd and mean for the data.

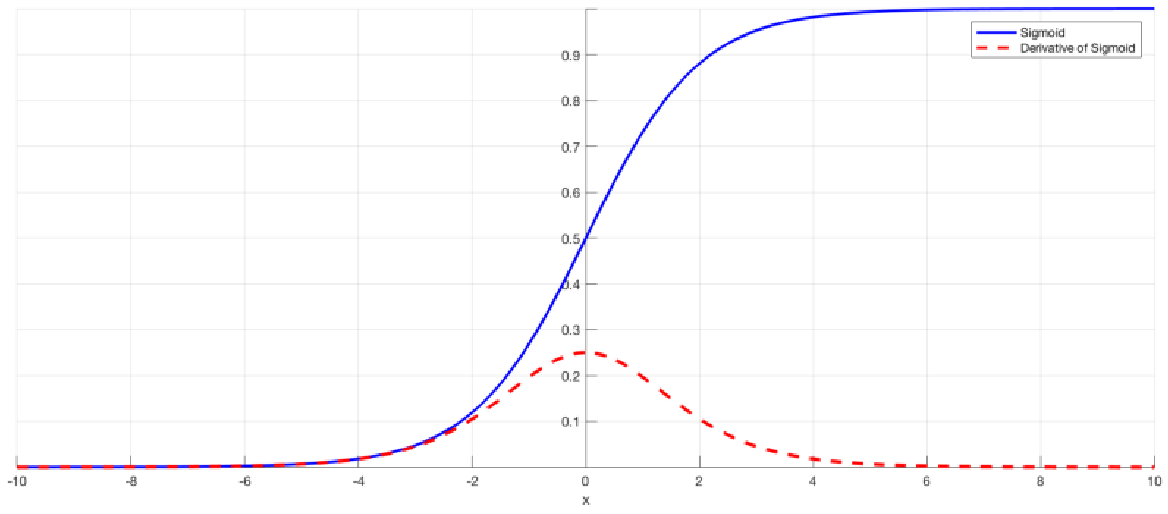


Figure 2: Sigmoid and It's derivative

g is analogous to sd and b is analogous to mean four parameters m,s,g, and b will change and will ensure that data is normalized.

BN is focused on taking an activation output from the preceding layer, subtracting the batch mean, and dividing by the batch standard deviation. Thus if there are extreme values in the preceding layer, they will not cause exploding or vanishing gradient problem in the next layer.

Difference between normalization and batch normalization

1. The former is applied on the whole data, while the latter is applied on the batch data.
2. Normalization is for the data at the input. BN is within the network.

1.6 Overfitting

An overfit model has negligible error during the training phase, but has comparatively higher error during the testing phase.

An underfit model has high error rate during the training phase. Figure **??**fig:overfit explains the difference. Overfitting occurs when we try to fit all possible data points.

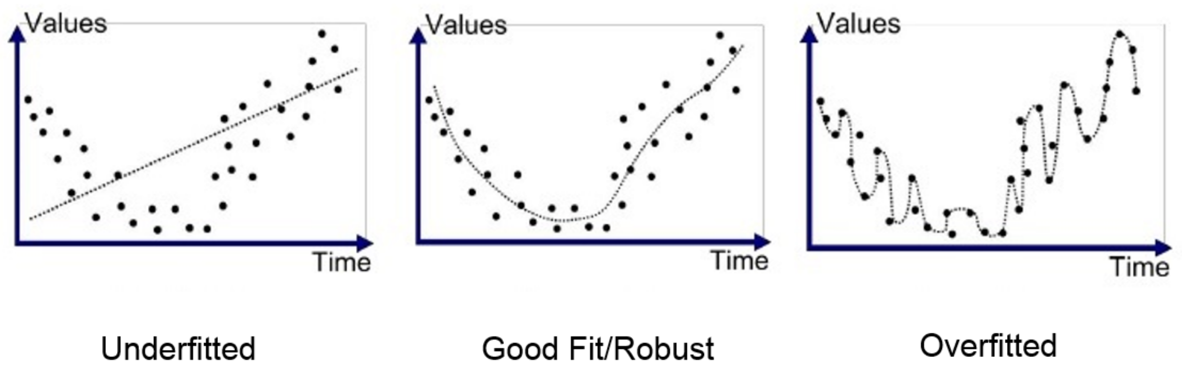


Figure 3: Overfitting vs. Underfitting

Following are the techniques to overcome the overfitting problem.

1.6.1 L1 and L2 Regularization

Penalize models for including parameters by including these parameters in the cost function. The larger the size of the parameter, the more will be the cost.

Consequently, parameters are not retained unless they contribute to the difference between estimated and actual values of y.

Difference between L1 and L2 is that L1 adds actual value of the parameter, whereas L2 adds the square value.

1.6.2 DropOut

It is another technique to solve the overfitting problem. Dropout prevents any neuron for becoming excessively influential. As compared to BN, however, dropout does not constrain parameter.

It assumes that a proportion of neurons in each layer does not exist. A round of iterations are followed. In each iteration, we randomly drop a certain percentage of neurons. This dropping is totally random, i.e, for a given layer, each iteration does not have any linkage with the dropout of previous iteration.

Once the training is completed, an additional step is needed in the validation process. Since during training, only a subset of neurons have been used, a proportion weight age is used as a multiplication factor during validation. That is, if for a given layer,

2 LeNet

Purpose: Hand Digit classification and recognition Commercial Use: read millions of cheques in banks Developed in 1998. Input: MNIST database size 28x28 (sir please confirm, the input size is 32x32. distortions were introduced images were scanned

2.1 Architecture

Figure 4 illustrates the architecture of Lenet, while figure 5 illustrates layered-wise approach. The input image is a 32x32 gray scale.

The LeNet architecture has two blocks, a convolutional block and a dense-layer block. The convolutional block has two convoltional layers, each of them consists of the following:

1. A convolutional layer with multiple filters of size 5x5
2. Sigmoid activation function, and
3. A pooling operation

Note that for CNN, RELU activation function works better. However, at that time (1998), this was unknown. LeNet uses average pooling function.

2.1.1 convolutional block

The first convolution layer used 6 filters with size of 5*5 with a stride of one and no padding. After that average pooling is applied with a size of 2*2. The second convolutional layer has 16 filters, each of size 5*5 with a stride of one and no padding. Again, average pooling is applied with a size 2*2.

2.1.2 Dense block

In LeNet, the output of the second convolutional layer is flattened (after applying activation and pooling). The output is connected to a fully connected layer of 120 neurons, which is further connected to another layer of 84 neurons. The final layer has 10-classes, one for each digit. Softmax function is used as an activation function.

$$\sigma(z) = \frac{e^{z_i}}{\sum_{J=1}^{k} e^{z_j}} \tag{1}$$

As we go deeper, size of feature map shrinks (Hint: pooling) but the no. of feature maps increase.

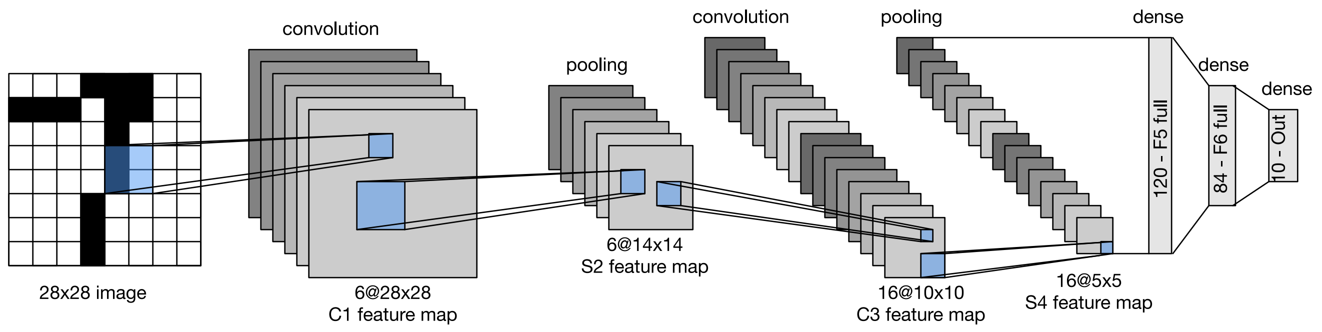


Figure 4: Lenet Architecture proposed by Yann Lecunn in 1998 is the pioneer convolutional neural network for image recognition. It was trained on gray scale images and on MNIST handwritten digit dataset.

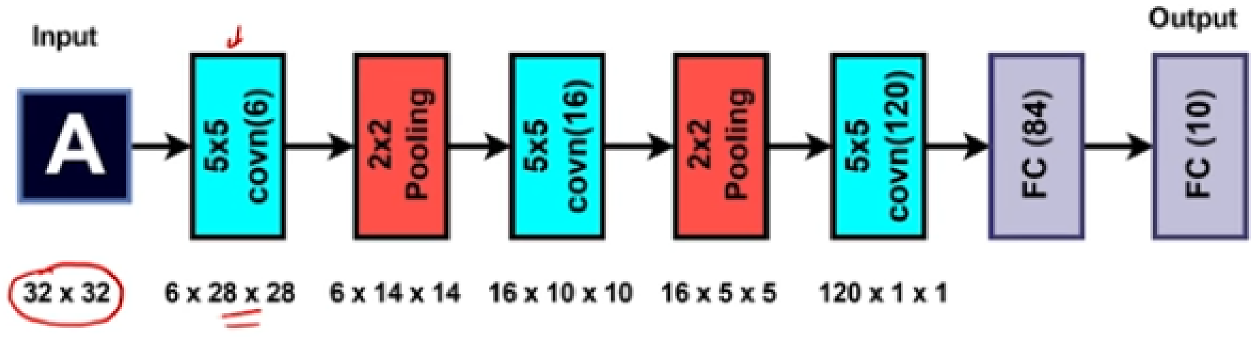
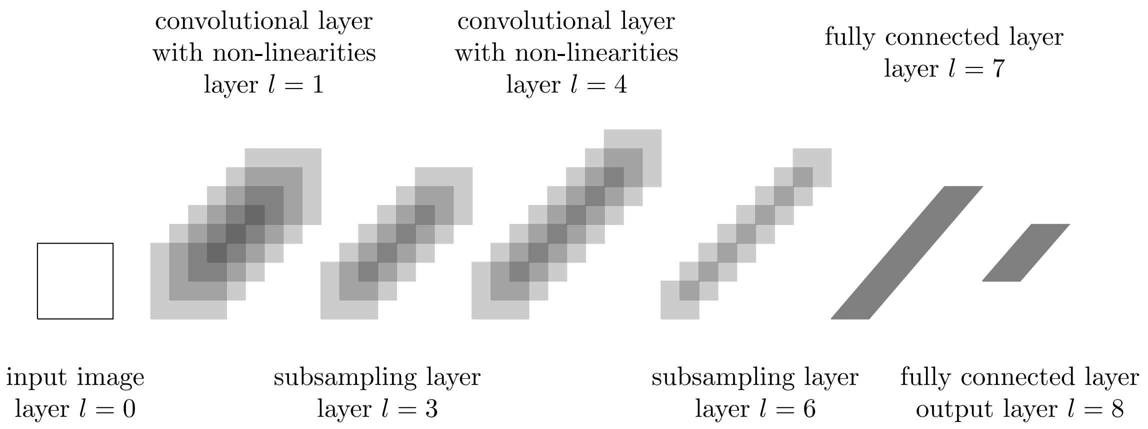


Figure 5: Lenet Architecture - layers



convolutional layer with non-linearities layer $l = 1$
convolutional layer with non-linearities layer $l = 4$
fully connected layer layer $l = 7$
input image layer $l = 0$
subsampling layer layer $l = 3$
subsampling layer layer $l = 6$
fully connected layer output layer $l = 8$

Figure 6: The architecture of the original convolutional neural network, as introduced by LeCun et al. (1989), alternates between convolutional layers including hyperbolic tangent non-linearities and subsampling layers. In this illustration, the convolutional layers already include non-linearities and, thus, a convolutional layer actually represents two layers. The feature maps of the final subsampling layer are then fed into the actual classifier consisting of an arbitrary number of fully connected layers. The output layer usually uses softmax activation functions.

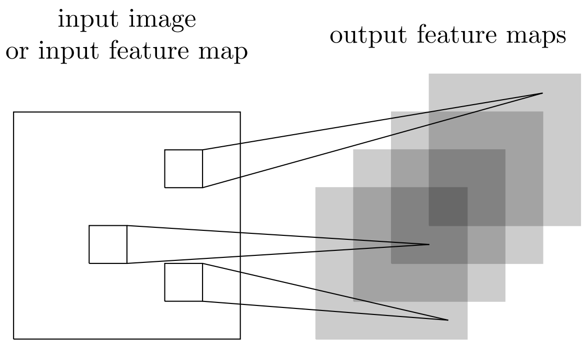


Figure 7: Illustration of a single convolutional layer. If layer l is a convolutional layer, the input image (if $l = 1$) or a feature map of the previous layer is convolved by different filters to yield the output feature maps of layer l.

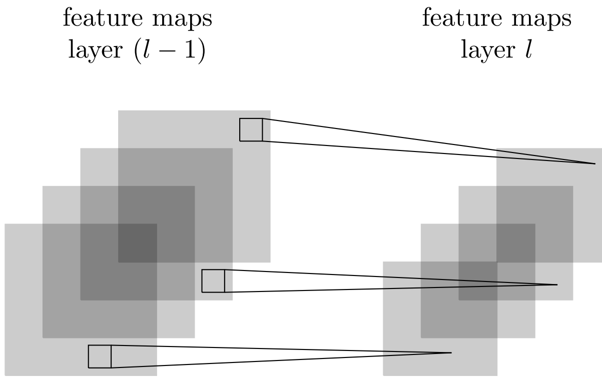


Figure 8: Illustration of a pooling and subsampling layer. If layer l is a pooling and subsampling layer and given $m_1^{(l-1)} = 4$ feature maps of the previous layer, all feature maps are pooled and subsampled individually. Each unit in one of the $m_1^{(l)} = 4$ output feature maps represents the average or the maximum within a fixed window of the corresponding feature map in layer $(l-1)$.