LeNet

Paper and Implementation Notes

June 23, 2020

Erik Rosten

1 Paper

AlexNet is a paper proposed in 1998, as one of the first publications using a deep learning convolutional neural net [1].

1.1 Dataset

LeNet is evaluated on the MNIST dataset, a subset of the NIST dataset. From the paper, it is constructed with Nist standard datasets SD-3 and SD-1. SD-1 contains 58,527 digit images by 500 different writers. SD-3 contains blocks of data from each writer in sequence.

The training set is composed of the first 250 writers (around 30,000) and the rest is filled in from SD-3 to make 60,000 training samples. The test set is composed of 5,000 images from each set.

The dataset was processed so they are size normalized to fit in a 20x20 pixel box while preserving aspect ratio. The resulting images contain gray levels as a result of image interpolation techniques of the normalization algorithm. They are centered by computing the center of mass of the pixels, and translating the image such that the center of mass is in the center of a 28x28 field. This is the MNIST dataset.

1.2 Preprocessing

- White pixels scaled to -0.1
- Black pixels scaled to 1.175
- Makes the mean roughly 0 and the variance roughly 1

1.3 Important Concepts

1.3.1 Learning from Data

Theoretical and experimental work has shown that the gap between the expected test error and training error approximately decreases as

$$E_{test} - E_{train} = k \left(\frac{h}{P} \right)^{\alpha}$$

where

- P is the number of training samples
- h is the "effective capacity" of the machine
- α is between 0.5 and 1.0
- k is a constant

As the capacity h increases, the training error E_{train} decreases, but there is a trade-off between the decrease of E_{train} and the increase of the gap. There is an optimal value of h that achieves the lowest generalization error E_{test}. Regularization (structural risk minimization) is a tool to help with this and can be implemented as minimizing

$$E_{train} + \beta H(W)$$

where H(W) is a regularization function, and β is a constant. H(W) typically takes large values on parameters W, which limits the capacity of the accessible subset of the parameter space.

1.3.2 Convolutional Networks

Convolutional Networks combine 3 ideas to ensure some degree of shift scale and distortion invariance: local receptive fields, shared weights and spatial or temporal sub-sampling.

Using local receptive fields, neurons can extract elementary visual features, which are then combined in subsequent layers to detect higher order features. By sharing weights among a set of units whose receptive fields are located at different places on the image, we force these feature detectors to be useful on the entire image. Each feature detector with the same weight is dubbed a feature map. A unit in a feature map that has a filter size of 5x5 would have 25 inputs, and it's receptive field would be the 5x5 area in the input. Thus, one unit has 25 trainable coefficients and 1 bias unit. The receptive fields of neighboring units in a feature map overlap int he original image, but share weights and detect the same feature at all locations in the input. Then if the image is shifted, the feature map output will be shifted by the same amount and left unchanged otherwise. This leads convolutional networks to be robust with respect to shifts and distortions of the input.

Once a feature is detected, its exact location is not as important as its position in relation to other features and may in fact be harmful as the exact location with respect to different features may change depending on the image given. In terms of MNIST, the example of a horizontal line at the top and corner in the top right for a 7 is given. These locations may be different for different 7's given. To reduce the spatial resolution of the feature maps, sub-sampling layers (pooling) are formed, which reduce the resolution of the feature map and further reduce the sensitivity of the network to shifts and distortions. LeNet uses average pooling, and multiplies the result by a coefficient and adds a bias. These parameters control the effect of the sigmoid nonlinearity. If the coefficient is small, the sub-sampling layer blurs the input. If the coefficient is large, they perform a "noisy OR" or "noisy AND" depending on the bias value.

In LeNet-5, the centers of the receptive fields of the last convolutional layer form a 20x20 area in the center of the input image.

1.4 Architecture

1.4.1 Activation Function

The **sigmoid** function is used in the output layer as it accelerates training time.

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

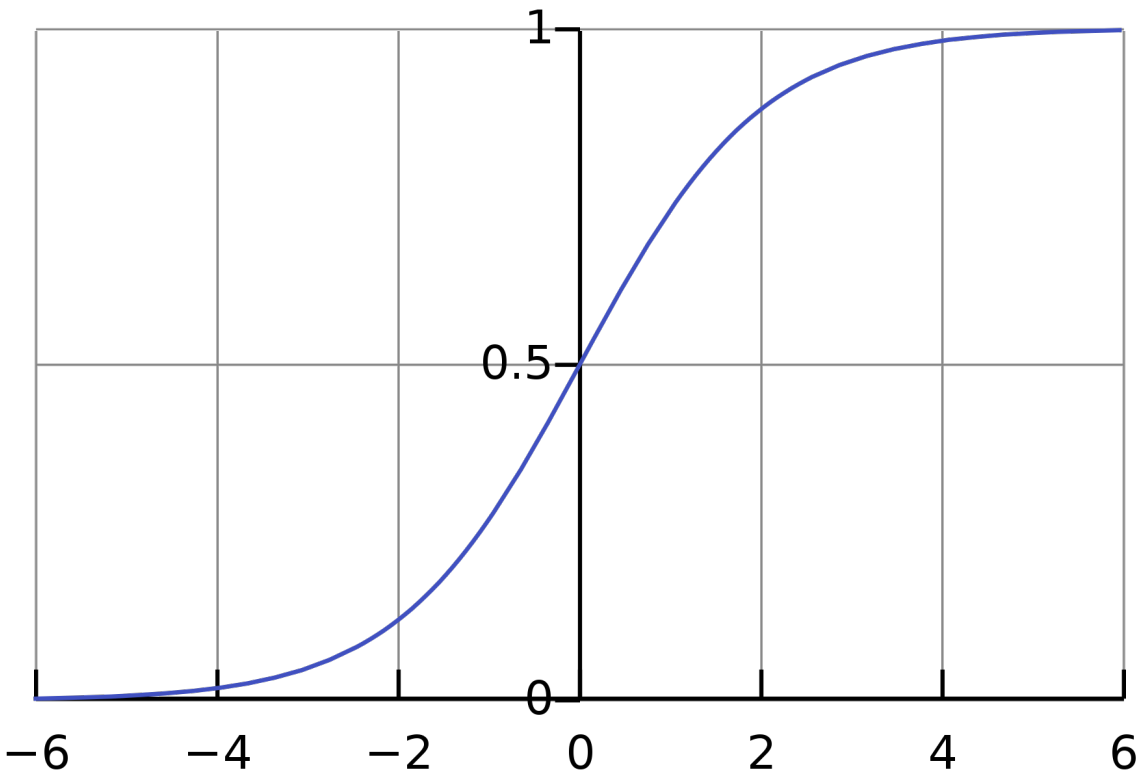


Figure 1: Sigmoid Function

The **Tanh** function is used in all other layers, which is a rescaled sigmoid function

$$tanh(x) = 2S(x) - 1$$

A comparison is below where the blue is the sigmoid, and red is tanh

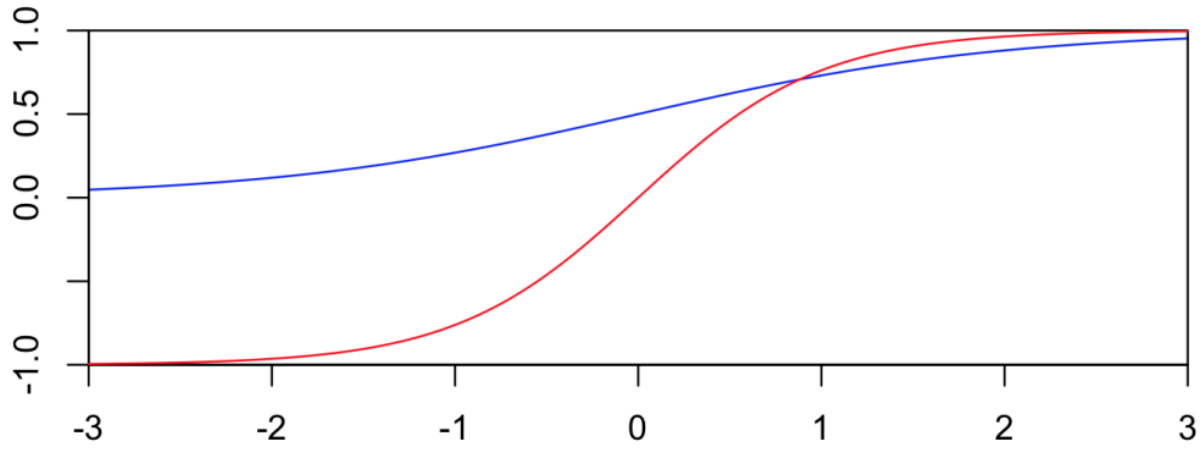


Figure 2: Tanh vs Sigmoid Functions

ReLU was not available at the time of the publishing of the paper.

1.4.2 Final Architecture

Resulting image size is calculated as

$$ceil\left(\frac{W-K+2P}{S}\right)+1 \tag{1}$$

where

- W is the input width
- K is the kernel size
- P is the padding
- S is the stride

Trainable parameter calcluations from convolutional layers can be calculated as

$$f \cdot K^2 \cdot C$$

where K is the kernel size, f is the number of feature maps of that layer, and C is the number of channels in the previous layer

Trainable parameters calculations from fully connected layers can be calculated as

$$C_{in} \cdot C_{out}$$

where C represents number of input and output channels.

Layer Name	Feature Maps	Kernel Size	Stride	Resulting Image Size	Trainable Parameters
Conv1	6	5 x 5	1	28 x 28 x 6	156
Average Pooling	6	2	2	14 x 14 x 6	12
Conv2	16	5 x 5	1	10 x 10 x 16	1,516**
Max Pooling	16	2	2	5 x 5 x 16	32
Conv3	120	5 x 5	1	1 x 1 x 120	48,120
FC1	-	-	-	84	10,164
RBF Output Layer	-	-	-	-	-
Total	-	-	-	-	60,000

Table 1: Summary of LeNet-5

** The second convolutional layer combines different pooling feature maps in a different scheme. The table below shows this scheme

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

Thus, the number of trainable parameters as usually computed assumes all input feature maps are mapped to each output feature map. However, here we have that the 1st feature map of C3 connects 3 feature maps of S2. Then, for this particular connection we have

$$5 \cdot 5 \cdot 3 = 75 + 1 = 76 \text{ Trainable Parameters}$$

where the 3 comes from the bias. We have 76 trainable parameters for each C3 feature maps 0,1,2,3,4,5.

There are 4 S2 feature maps in C3 feature maps 6,7,8,9,10,11,12,13,14 with $100 + 1$ trainable parameters each. Finally, there are 6 S2 feature maps in C3 feature map 15 giving 151 trainable parameters.

Then, the total number of trainable parameters for this layer are

$$(76 \cdot 6) + (101 \cdot 9) + (151 \cdot 1) = 1516 \text{ Trainable Parameters}$$

This type of connection scheme is unusual in todays CNN's, but was motivated in the paper as forcing the symmetry in the network to be broken and forcing different C3 feature maps to extract different complementary features. It had the added benefit of reducing the number of connections and trainable parameters. Using the more common fully connected scheme results in

$$(5 \cdot 5 \cdot 6 + 1) \cdot 16 = 2416 \text{ Trainable Parameters}$$

2 Implementation

The implementation of LeNet5 differs from the paper in several ways summarized below

- The connections for C3 use the more common fully connected scheme.
- Max Pooling instead of average pooling
- ReLU instead of tanh/sigmoid
- A fully connected into softmax output layer is used instead of the RBF's in the original paper

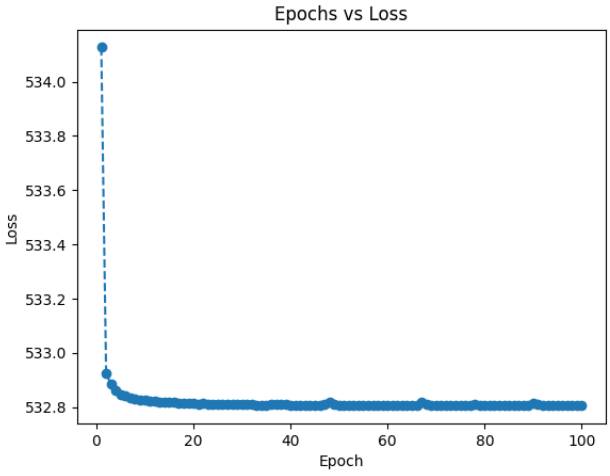
2.1 MNIST Training

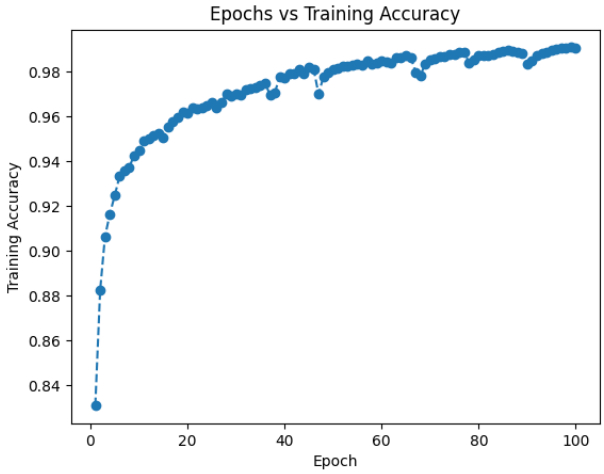
As a baseline, we use cross entropy loss with vanilla stochastic gradient descent of learning rate 1 and train for 30 epochs.

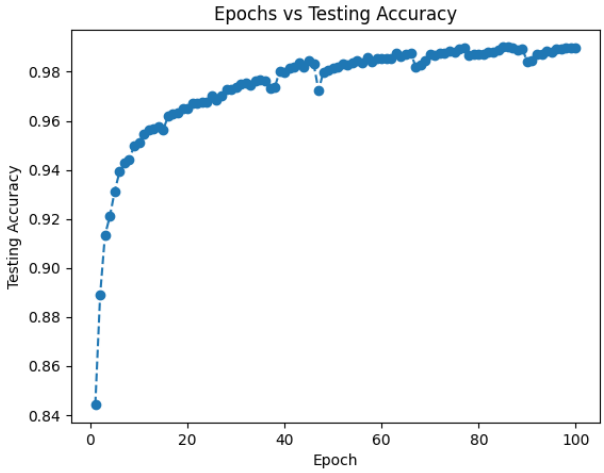
Optimizer	Total Epochs	Training Accuracy	Test Accuracy	Experiment #
Vanilla SGD	30	87.04	88.23	1
SGD w/ Classic Momentum	30	93.43	93.78	2
SGD w/ Nesterov Momentum	30	93.99	94.49	3
Adam	30	97.02	97.36	4
Adam	60	98.49	98.58	5
Adam	100	99.08	98.99	6

Table 2: Training Results with Different Optimizers

Training plots for the last 100 epoch run of Adam are shown below

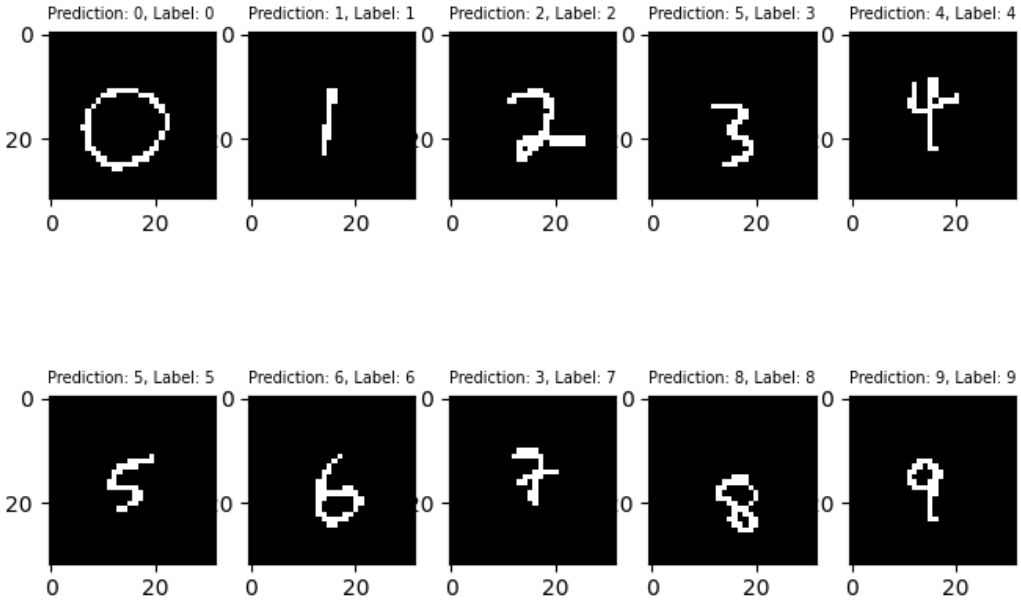






2.2 Testing

I test the trained model with my own handwritten digits, and the results are shown below. The trained model correctly predicted 8/10 of my handwritten digits, misclassifying '3' as '5' and '7' as '3'. Both these misclassifications are relatively understandable, and the original paper actually shows examples of missclassification where many of them misclassify '3' as '5'.


Prediction: 0, Label: 0
Prediction: 1, Label: 1
Prediction: 2, Label: 2
Prediction: 5, Label: 3
Prediction: 4, Label: 4
Prediction: 5, Label: 5
Prediction: 6, Label: 6
Prediction: 3, Label: 7
Prediction: 8, Label: 8
Prediction: 9, Label: 9
0
20

References

[1] *LeCun, Yann, Leon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-Based Learning Applied to Document Recognition," n.d..*

Code