Neural Networks - 1, 2, 3 Hidden Layers

Exploring the neural network training from the perspective of maximum likelihood and maximum a posteriori parameter learning.

1. Demonstrating that a neural network to maximize the log likelihood of observing the training data is one that has softmax output nodes and minimizes the criterion function of the negative log probability of training data set.

2. Parameters for Neural Network:

    - 1 hidden layer of 30 sigmoid nodes, and an output 10 softmax nodes from 1000 training images (100 images per digit)
    - Training the network for 30 complete epochs, using mini-batches of 10 training examples at a time
    - Learning rate = 0.1
    - Plotting the training error, testing error, criterion function on training data set, criterion function on testing data set of a separate 1000 testing images (100 images per digit), and the learning speed of the hidden layer
    - 2 hidden layers of 30 sigmoid nodes each
    - 3 hidden layers of 30 sigmoid nodes each
    - Performing the above with and without L2 regularization, $\lambda = 5$

Tools & Liblaries: Python 3, Tensorflow, Numpy, Matplotlib

1 Proof

Desired output is a posterior probability distribution over the ten digit classes.

$J_o(w) = -\log p(\{(x_n, t_n) : n = 1, 2..9\}; w) = -log \prod_n \prod_{m=0}^{9} p(t_n = m|x_n, w)$

w - synapses weights;
10 softmax output nodes generating $\log p(t = m|x; w)$, where $m = 0, 1, ..., 9$;
t_n - label of x_n image;
n - index of a pattern in the training data set.

1. Probabilistic approach

We have 10 classes $m = 0, 1, 2, ...9$
Prior probability that a label(t) = m:

$$\pi_m = Pr[t = m] \tag{1}$$

Conditional distribution - probability that an image belongs to class m:

$$P_m(x) = P(x|y = m) \tag{2}$$

Thus, an overall distribution:

$$P(x) = \sum_{m=0}^{9} \pi_m P_m(x) \tag{3}$$

In our case the prior probability is equal for each classes, as we consider the same constant number of images (100) per each label. So we can omit constant π_m and the posterior probability is:

$$P(x) = \sum_{m=0}^{9} P_m(x) = 1 \tag{4}$$

2. Suppose that we are given a data set of pairs $D = (x_i, y_i$ consisting of pairs of inputs x_i and corresponding outputs y_i for $i = 1, 2, , N$.

The log likelihood function for neural network is going to be $P(y|x;\omega)$, where ω is the set of adjustable parameters of the model i.e., the weights and biases of the network.

The log likelihood function for a dataset of D is:

$$F(O) = \sum_{m=0}^{9} \log p(y_m|x;\omega) \tag{5}$$

3. The logistic output function can only be used for the classification between two target classes t=1 and t=0. We have a multiclass categorical probability distribution, for this case we can use the softmax function, that returns values between 0 and 1, which allows us to interpret the outputs as probabilities. This function is a normalized exponential and is defined as:

$$P(t = m \mid \mathbf{x};\omega) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_m}}{\sum_{m=1}^{M} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_m}}, \tag{6}$$

where denominator acts as regularizer to assure, that $\sum_{m=0}^{9} y_m = 1$

4. We apply criterion function over the output from softmax. Softmax in this case play a key role that helps to omit logarithms with exponent while doing a backpropagation, thus it is easier to perform interpretation and optimization.

Cross-entropy defines as follows:

$$J_o = -\sum_{m=0}^{9} y_m \log p_m, \tag{7}$$

Where p_m is the output from the softmax.

In this case we use negative sign, as p_m is a number within the range $[0, 1]$ and the logarithm of it is negative, in order to keep the loss function positive number we add a negative sign to it.

p_m is one-hot encoded labels, where one label = 1 and all the rest is 0.

$$\sum_{m=0}^{9} y_m = 1 \tag{8}$$

Thus our loss function is:

$$J_o = -\sum_{m=0}^{9} \log p_m \tag{9}$$

We have a large distribution, thus we need to take a sum over all samples:

$$J_o(\omega) = -\sum_{n} \sum_{m=0}^{9} \log p(t_n = m | x_n; \omega), \tag{10}$$

where n is a number of all samples.

Product rule of logarithm defines as follows:

$$\log_b(x * y) = \log_b x + \log_b y \tag{11}$$

Applying product rule to our equation(10), we get:

$$J_o(\omega) = -log \prod_{n} \prod_{m=0}^{9} p(t_n = m | x_n; \omega) \tag{12}$$

2 Building Neural Networks

1. To normalize our parameter we use mean and standard normal deviation.
2. To initialize the parameter, we use $tf.truncated\_normal$ that returns random values from a truncated normal distribution. Truncated function add a property, where values whose magnitude is more than 2 standard deviations from the mean are dropped and re-picked. This helps to overcome situation where if the value is too big/small, the neuron stops learning.
3. We optimize our labels and convert them to 1-hot-encoding.
4. Forward pass step includes the following steps:

   a) Constructing M-linear combinations of the input:

$a_j = \sum_{i=1}^{D} w_{ji} x_i + w_{j0}^{(1)}$

b) Transform activation using non-linear activation function, for our task we are using sigmoid:

$z_j = \frac{1}{1+e^{(-a_j)}}$

c) Combine hidden units to give output unit activations:

$y_p = \sum_{i=1}^{M} w_{kj} z_j + w_{k0}^{(2)}$

d) Each output unit activation is transformed using a sigmoid function:

$o = \frac{1}{1+e^{(-y_p)}}$

5. For calculating the loss we are using function **tf.nn.softmax_cross_entropy_with_logits_v2**. Where softmax defined as:

   $P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$

   Softmax function takes an N-dimensional vector of real numbers and transforms it into a vector of real number in range (0,1).

   Cross-entropy defines as:

   $H_{y"}(y) := -\sum y_i^{"} log(y_i)$ where y_i is the predicted probability value for class i and $y_i^{"}$ is the true probability for that class.

6. Average learning speed defines as:

   $\frac{|w_{t-1} - w_t|}{w_t}/n$

   Below are the results from training a neural network with 1 hidden layer using the following parameters:

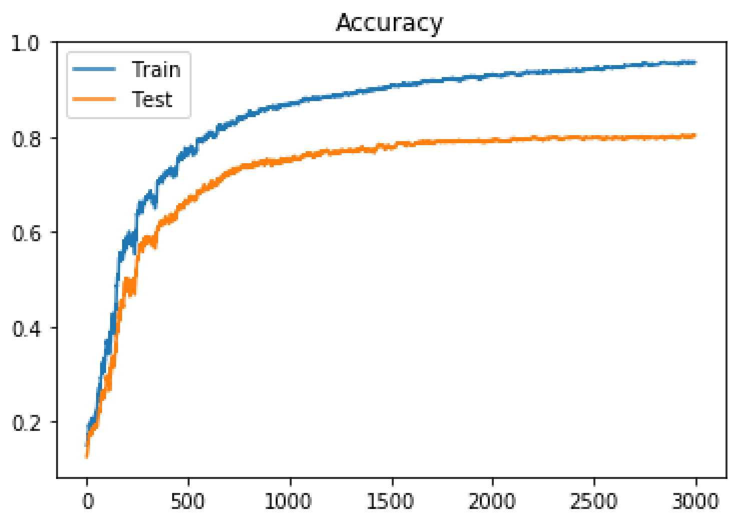
   - Number of hidden nodes = 30
   - Activation function: sigmoid

- Training dataset is organized 100 images per digit
- Number of epoch = 30
- Number of mini-batches = 10
- Learning rate = 0.1
- Lambda value= 5

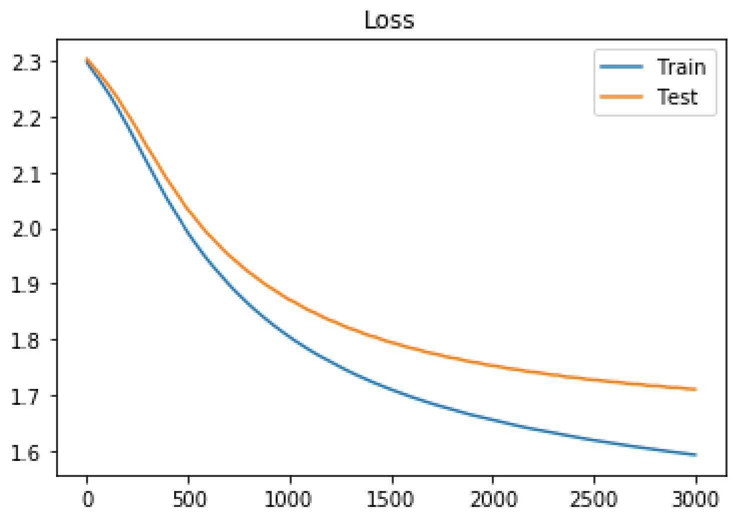
1-hidden layer neural network

Time usage 116 seconds

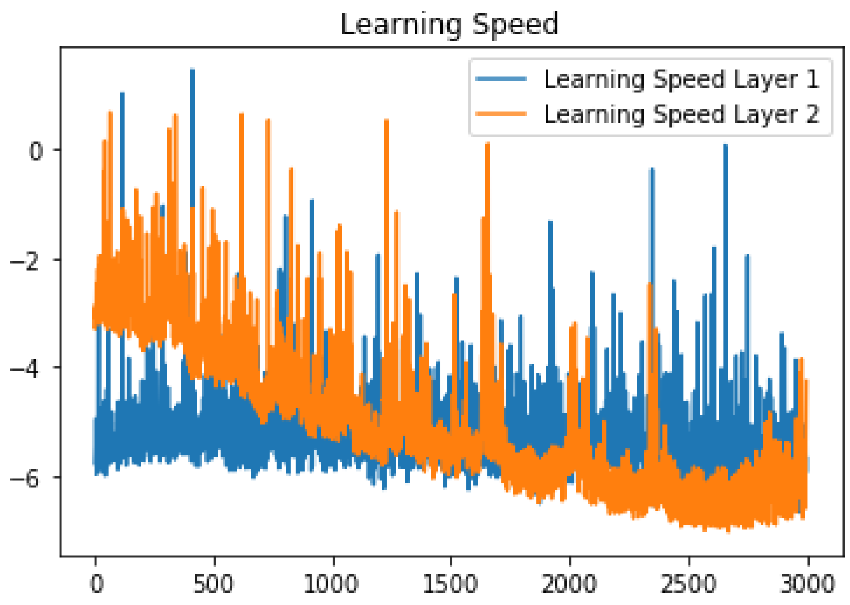
Accuracy
Epoch = 30
Train accuracy = 95.6%
Test accuracy = 80.3%



Loss
Epoch = 30
Train loss = 1.59
Test loss = 1.71



Learning speed
Learning speed of the hidden layer 1: 0.00986997
Learning speed of the hidden layer 2: 0.0218917



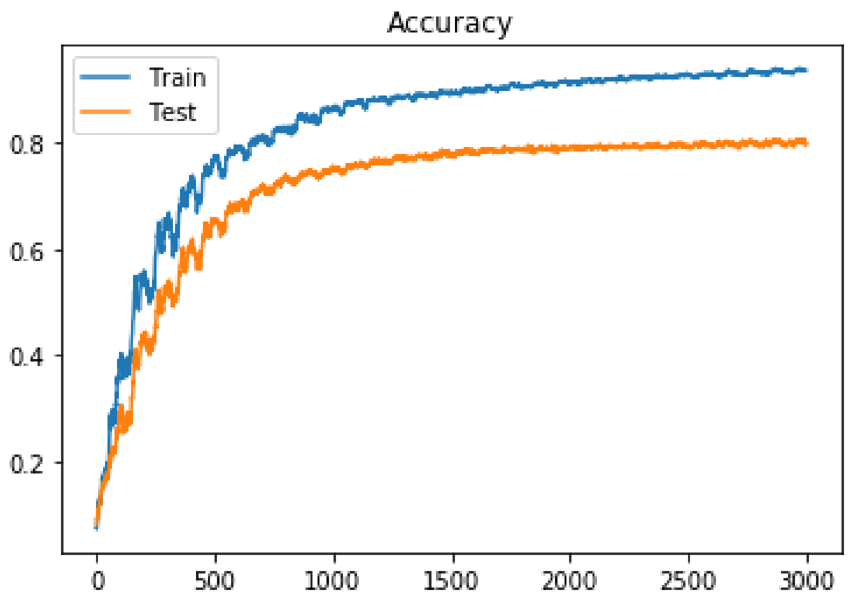
Thus we can see that learning speed is higher for the second layer, this confirms that on the deeper levels vanishing gradient is lower.
For sigmoid activation function backpropagation computes gradients by the chain rule. This has the effect of multiplying small numbers (from 0 to 1) to compute gradients of the front layers, meaning that the gradient decreases exponentially with each computation while the front layers train very slowly.

Overall the accuracy on the test data is 80.3% comparing to the accuracy of train data(95.6%), which shows an overfitting.

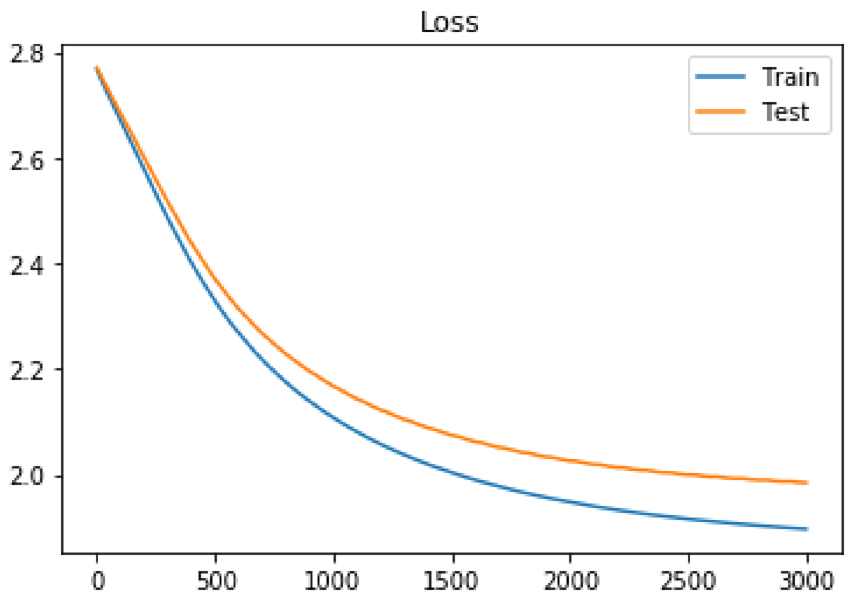
1-hidden layer with l2-regularization

Time usage 118 seconds

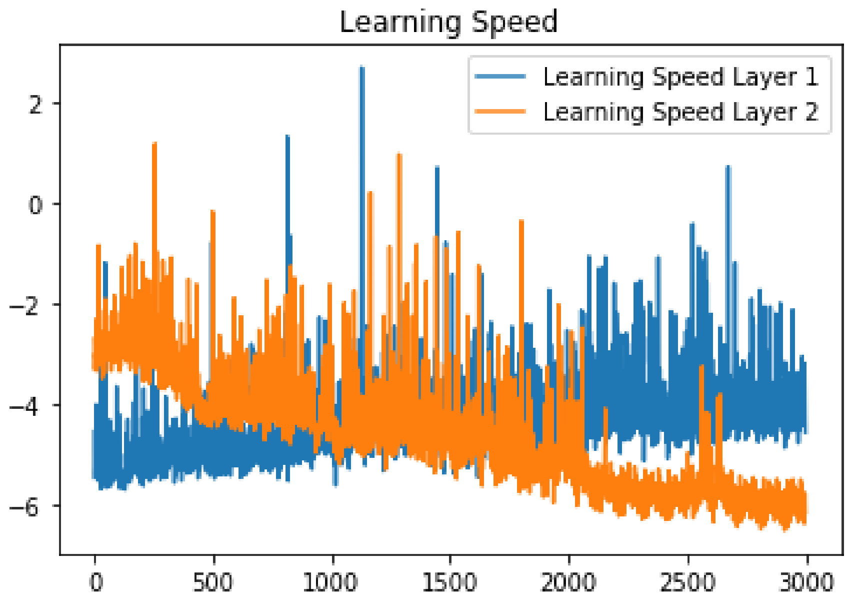
Accuracy
Epoch = 30
Train accuracy = 93.7%
Test accuracy = 79.9%



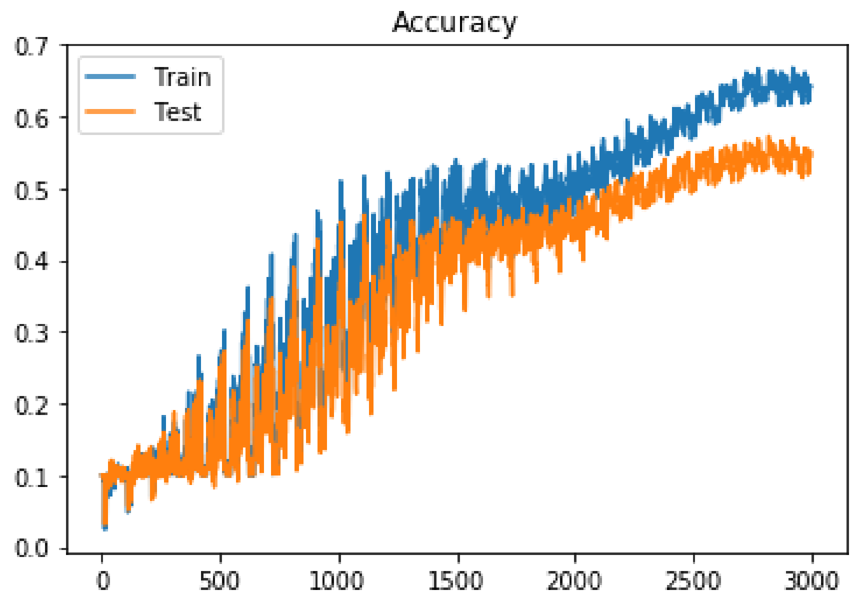
Loss
Epoch = 30
Train loss = 1.9
Test loss = 1.98



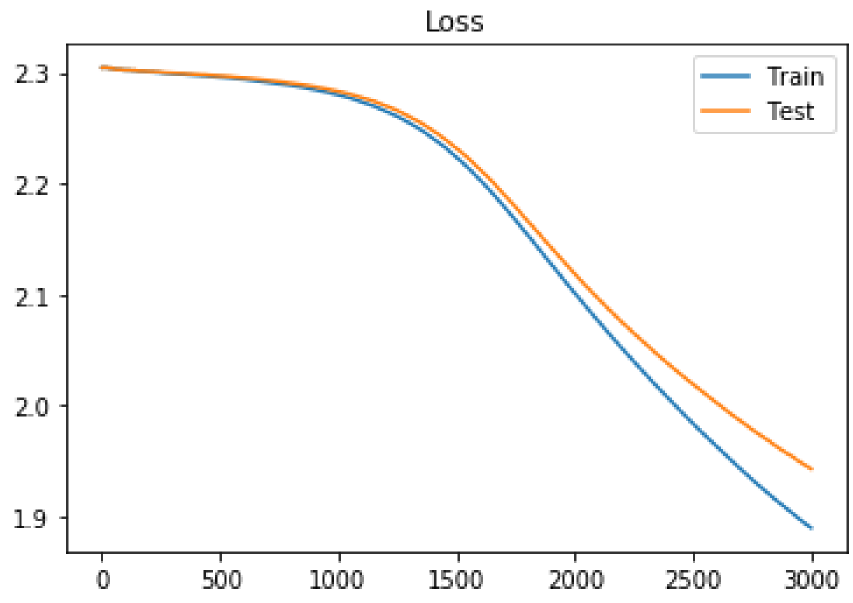
Learning speed
Learning speed of the hidden layer 1: 0.0248862
Learning speed of the hidden layer 2: 0.0233066



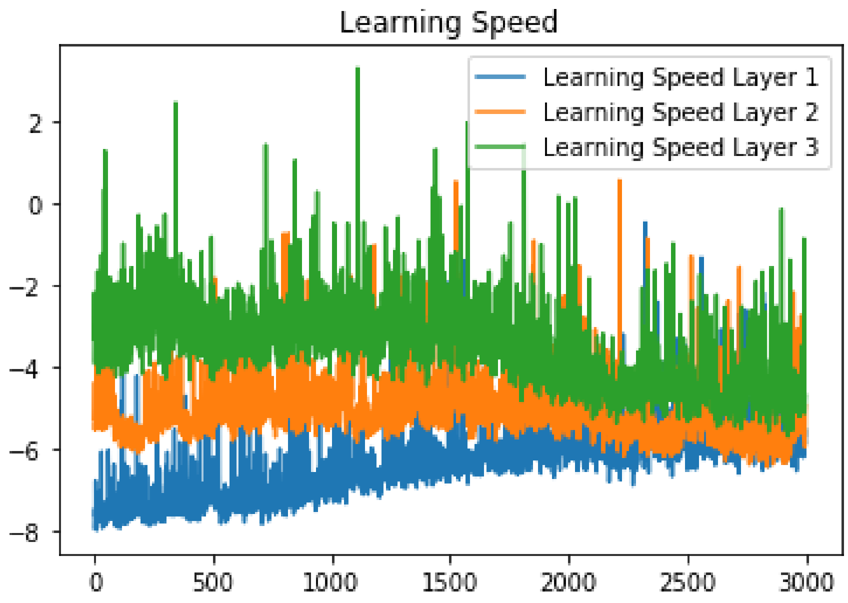
2-hidden layers without regularization Time usage 195 seconds
Accuracy
Epoch = 30
Train accuracy = 64.2%
Test accuracy = 54.4%



Loss
Epoch = 30
Train loss = 1.89
Test loss = 1.94



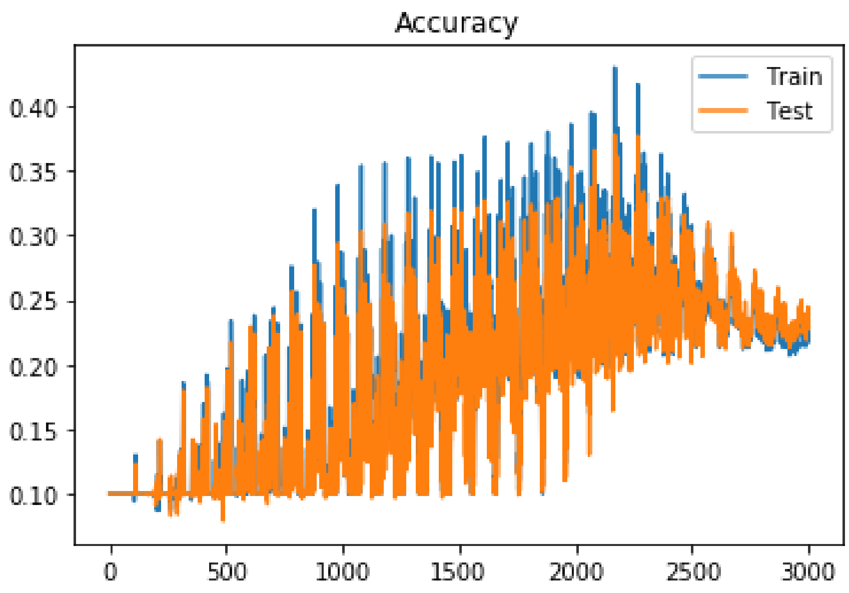
Learning speed:
Learning speed of the hidden layer 1: 0.00329046
Learning speed of the hidden layer 2: 0.010079
Learning speed of the hidden layer 3: 0.0714392



2-hidden layers with l2-regularization

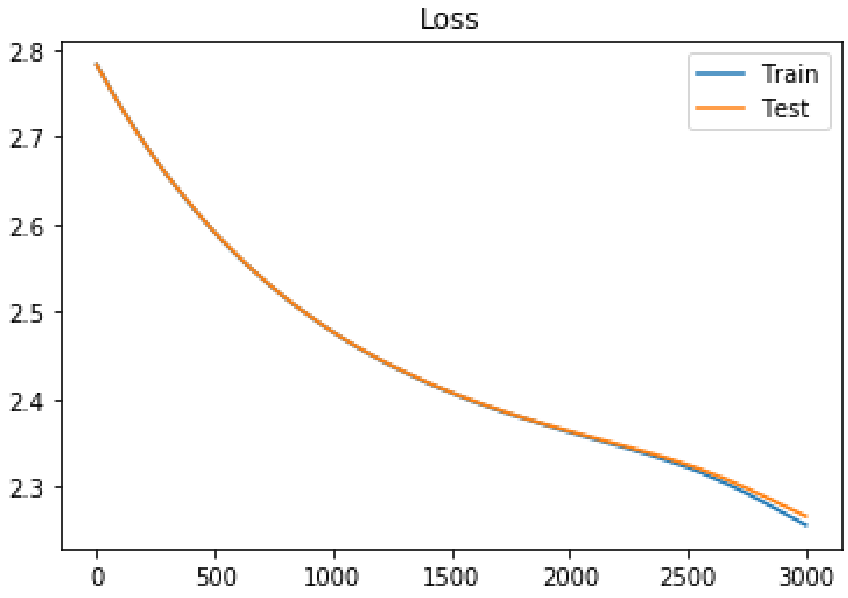
Time usage 118 seconds

Accuracy
Epoch = 30
Train accuracy = 21.9%
Test accuracy = 23.4%

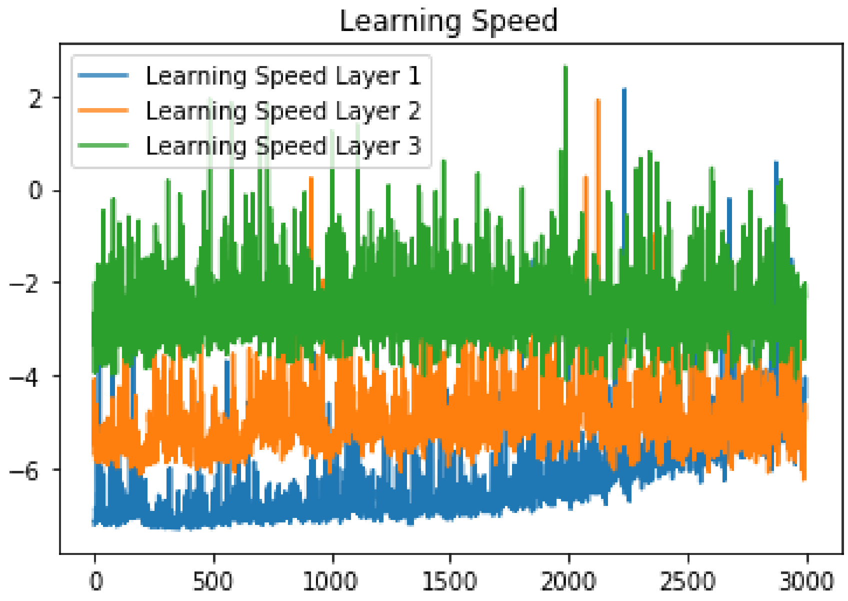


Loss
Epoch = 30
Train loss = 2.26

Test loss = 2.27

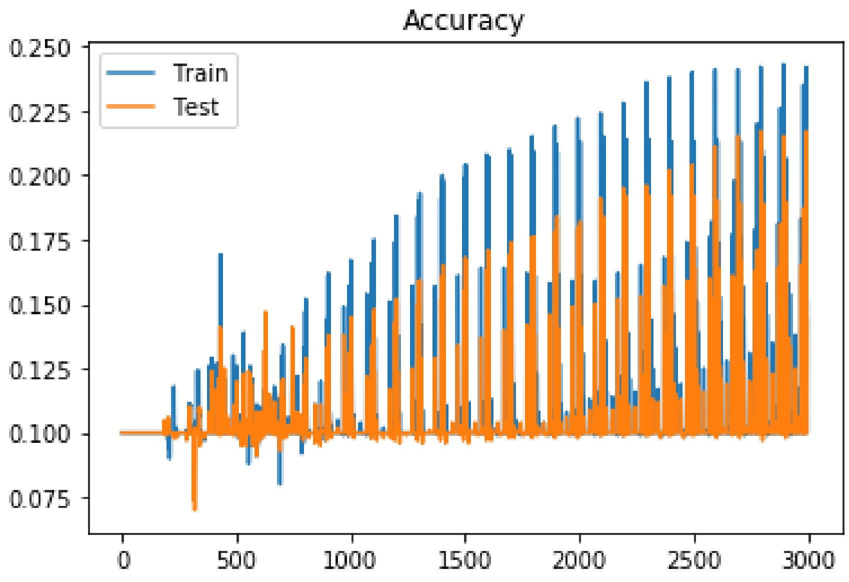


Learning speed:
Learning speed of the hidden layer 1: 0.00670494
Learning speed of the hidden layer 2: 0.011991
Learning speed of the hidden layer 3: 0.119064

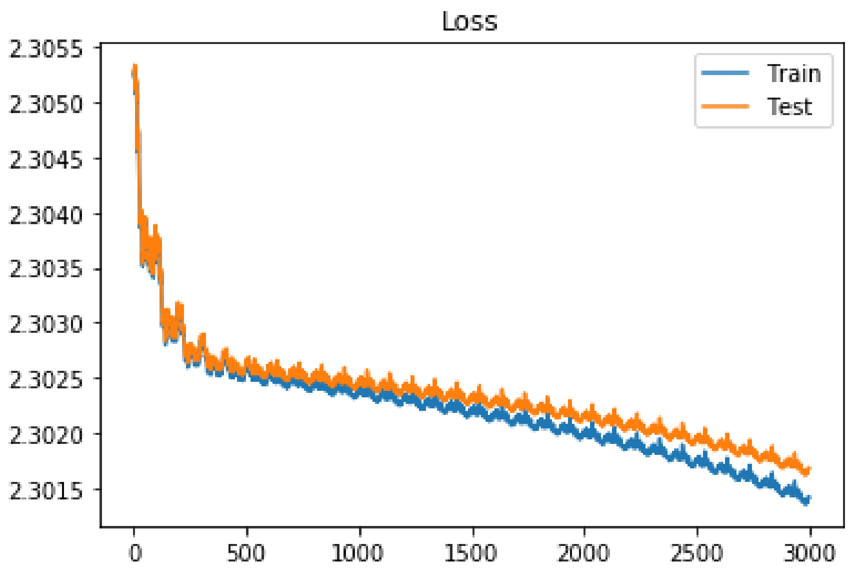


3-hidden layers without regularization

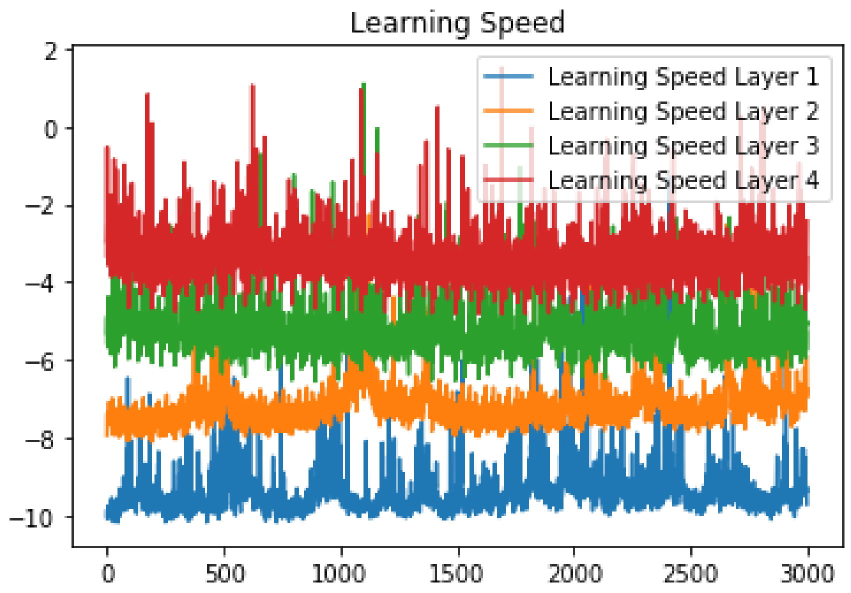
Time usage 250 seconds
Accuracy
Epoch = 30
Train accuracy = 15.1%
Test accuracy = 14.4%



Loss
Epoch = 30
Train loss = 2.3
Test loss = 2.3

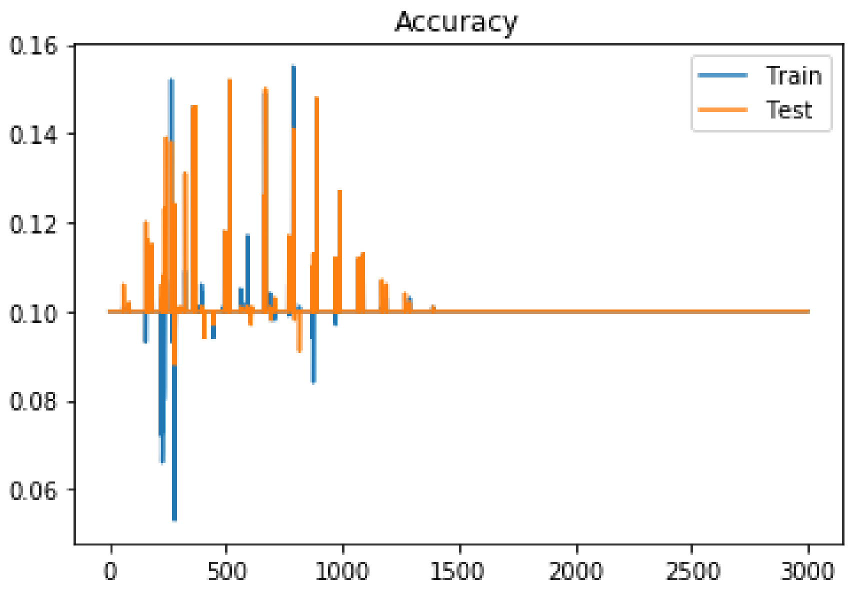


Learning speed:
Learning speed of the hidden layer 1: 0.000242181
Learning speed of the hidden layer 2: 0.00112451
Learning speed of the hidden layer 3: 0.00890441
Learning speed of the hidden layer 3: 0.0509406

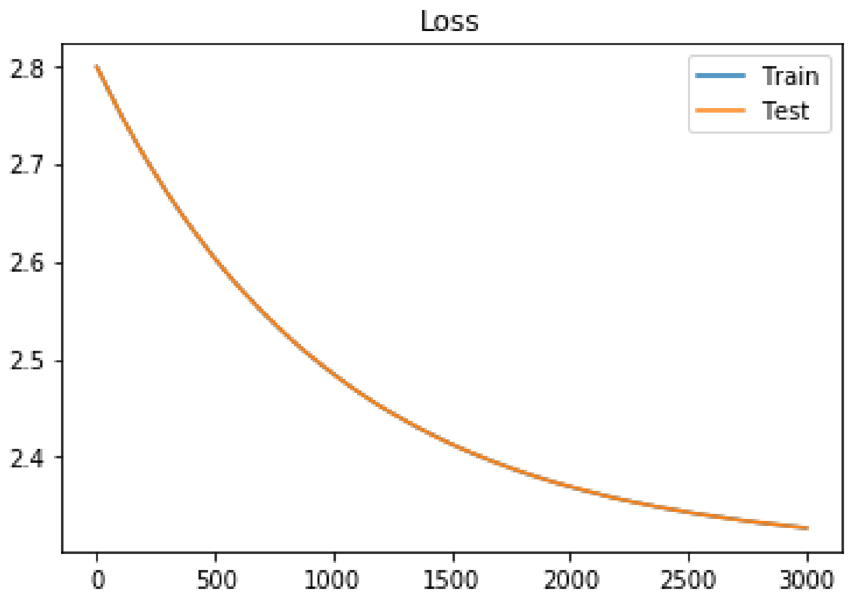


3-hidden layers with l2-regularization

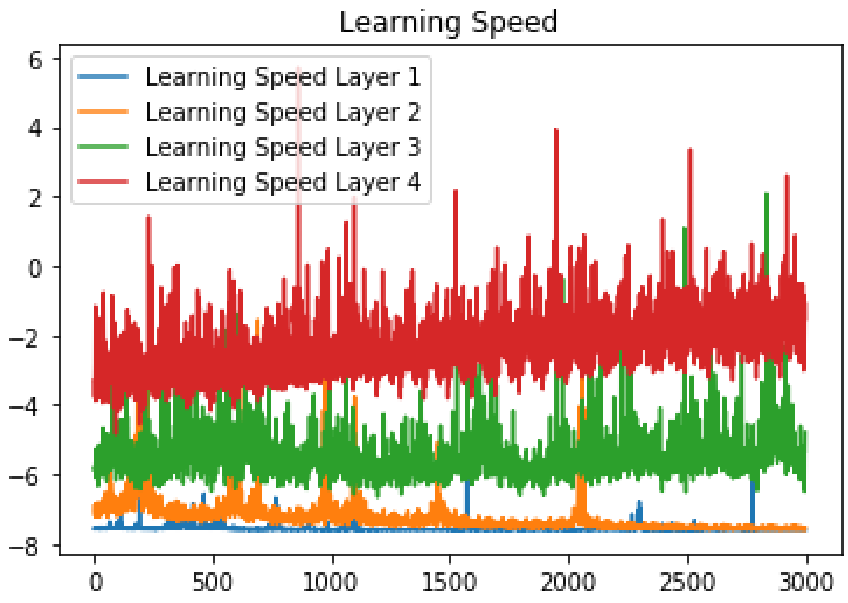
Time usage 250 seconds
Accuracy
Epoch = 30
Train accuracy = 10.0%
Test accuracy = 10.0%



Loss
Epoch = 30
Train loss = 2.33
Test loss = 2.33



Learning speed:
Learning speed of the hidden layer 1: 0.000522608
Learning speed of the hidden layer 2: 0.00106838
Learning speed of the hidden layer 3: 0.00979219
Learning speed of the hidden layer 3: 0.290812



As we can see for MNIST dataset, the best results in terms of accuracy and execution time is given while using neural network with 1 hidden layer without regularization. For neural networks with more hidden layers a vanishing gradient and overtting take place and decreasing the accuracy result for test dataset.