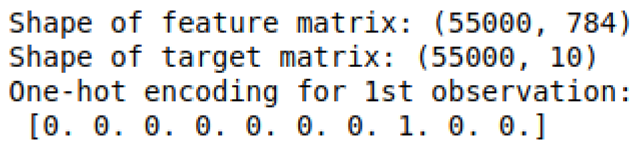
CVML Assignment – I

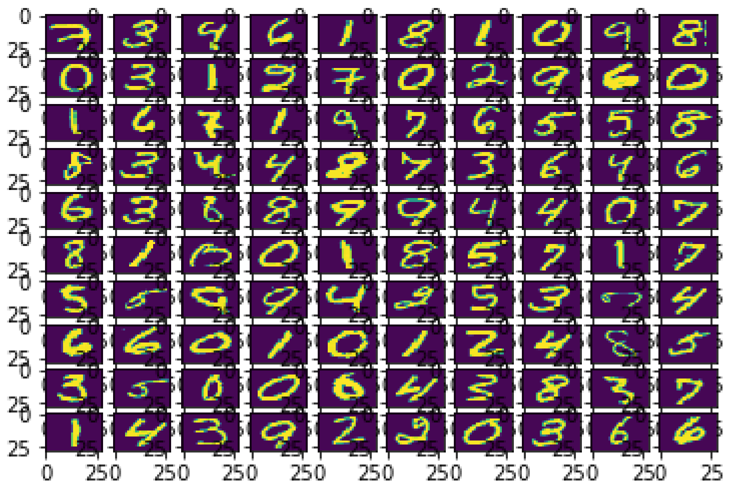
Aim of the assignment was to classsify MNIST sample images into one of the 10 possible classes (0-9 numbers), which we were to perform using the softmax regression with and without convolution layers.

The training data is represented by the following vectors:
Feature matrix, which is the vector of 55000 images of size 28x28 (784) unrolled.
Target matrix, same no.of images into given 10 classes.
Represented as a one-hot encoding for each image.



```
Shape of feature matrix: (55000, 784)
Shape of target matrix: (55000, 10)
One-hot encoding for 1st observation:
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
```

Input images are shown as a subplot, taken for 100 images.

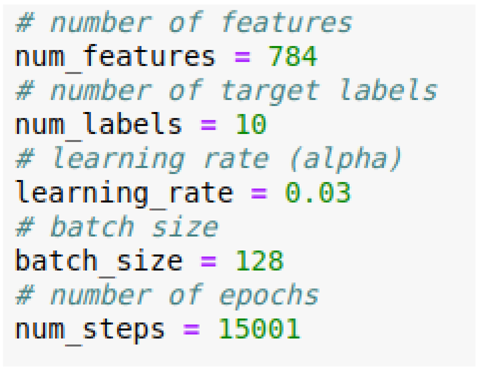


For the first part of the assignment softmax regression without convolution layers.

The parameters of softmax regression are set as follows:

- The number of features is the number of pixels in the 28x28 image which is 784.
- learning rate is the step size for the gradient descent.

- batch size is the number of training examples taken at a time for doing stochastic gradient descent.

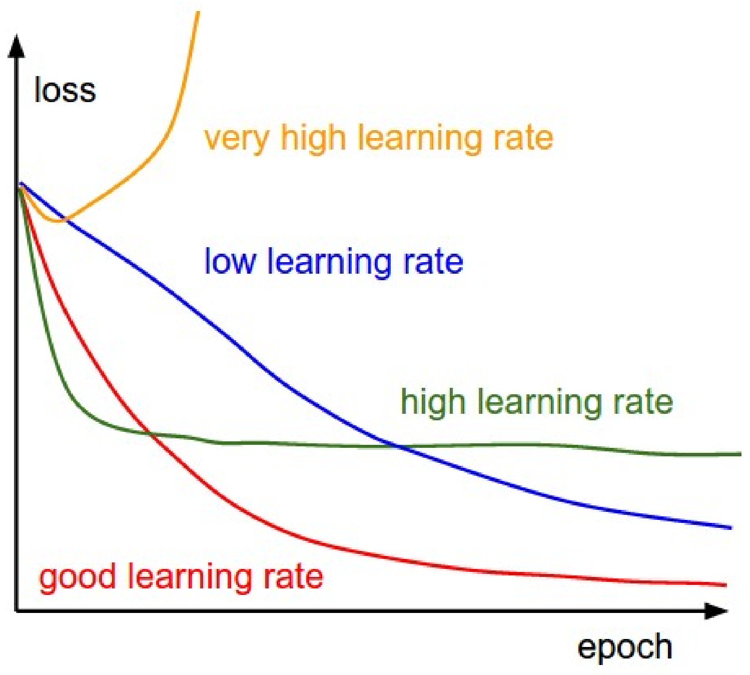


```
# number of features
num_features = 784
# number of target labels
num_labels = 10
# learning rate (alpha)
learning_rate = 0.03
# batch size
batch_size = 128
# number of epochs
num_steps = 15001
```

Instead of using traditional gradient descent method we used stochastic gradient descent. Here instead of using the entire training data to calculate the loss and then updating the weights we use some k training examples at a time to calculate the loss and then update the weights. So, the k training examples taken during training at any time is called the minibatch. In every iteration, a minibatch is selected by choosing a random offset value.

The benefit of using stochastic gradient descent is that we get better time complexity however the accuracy is bit degraded as we are not considering all the training examples all at the same time instead using minibatch to update weights. In contrast to traditional gradient descent which converges to a more accurate point, stochastic gradient descent may jump between values near the convergence point.

Relationship between epochs vs loss with different learning rates.



EPOCH	**LEARNING RATE**	**ACCURACY(%)**
15001	0.3	88.6
	0.9	90.3
	0.5	92.0
	1.0	90.4
	10	82.9
30000	0.5	92.0

It is observed that 15001 epochs, the highest accuracy is achieved at 92.0% with learning rate 0.5. So even on increasing the number of epochs there is no considerable change in the accuracy with the same learning rate (0.5).
So, we are not getting better accuracy than ~92.0% using just soft max regression.

Softmax regression with convolution

For this part we start with the following CNN architecture:

1. **Convolutional Layer #1**: Applies 32 5x5 filters (extracting 5x5-pixel subregions), with ReLU activation function

2. **Pooling Layer #1**: Performs max pooling with a 2x2 filter and stride of 2 (which specifies that pooled regions do not overlap)

3. **Convolutional Layer #2**: Applies 64 5x5 filters, with ReLU activation function

4. **Pooling Layer #2**: Again, performs max pooling with a 2x2 filter and stride of 2

5. **Dense Layer #1**: 1,024 neurons, with dropout regularization rate of 0.25 (probability of 0.4 that any given element will be dropped during training)

6. **Dense Layer #2 (Logits Layer)**: 10 neurons, one for each digit target class (0–9).

We provide an overview of why each layer is included:

Convolution Layer: Neurons in the convolution layer are connected only to pixels in their receptive field. Different possible set of weights, as a convolution kernel (or filter) are applied.

Some intuition in this regard is that lower level convolution layers infer low level features like edge detection while higher level layers infer more complex functions.

Pooling Layer: Their goal is to subsample the input image in order to recude the computational load, memory usage and no.of parameters. A pooling layer has no weights, all it does is aggregate the inputs using an aggreagation function like max or avg.

Dense layer: It is just a regular feed forward neural network, composed of Fully connected layers whose final layer outputs the raw class scores which when fed to the softmass function gives the classs probabilities.

Typically all of the hyperparameters we change have been appllied to the first layer of the architecture i.e. conv_layer 1. Although backprop might give few varying results on multiple runs overall accuracy is failry similar.

Relationship between type of activation function and accuracy

Observations: The final % accuracies by changing the activation function are as follows.

ACTIVATION FUNCTION	**ACCURACY(%)**
Softmax	97.04
Sigmoid	90.31
Relu	98.68
Tanh	97.14
elu	97.14
Selu	97.27
Softplus	93.41
Softsign	97.57

Sigmoid is performing the least accurate, whereas ReLU is performing the best.

Sigmoid function

$$h(x) = \quad 1/(1+e^{-x})$$

which has the gradient:

$$h'(x)=h(x)*(1-h(x))$$

When using sigmoids, the gradient of becomes increasingly small as the absolute value of x increases. The max value of the derivative is pretty small(0.25). This causes the learning to slow down rapidly. This is also known as the problem of **vanishing gradient**.

Tanh

To solve that, we can use the Tanh function, which has maxed derivative as one. Which improved the accuracy greatly.

ReLU – Rectified Linear Unit

relu(x) = max(0 , x)

which has the gradient:

$$relu'(x) = \begin{cases} 0 \text{ if } x \le 0 \text{ ;} \\ 1 \text{ if } x > 0 \text{ ;} \end{cases}$$

The gradient of relu has a constant value. Thus not allowing gradient to vanish. The fact that the gradient is zero for negative x’s might be seen as an issue at first but it actually helps to make the network sparse keeping the useful links. Sparsity helps to keep the network less dense and decreases the computations.

ELU – Exponential Linear Unit

$$elu(x) = \begin{cases} \alpha(\exp(x)-1) & \text{if } x \le 0 \text{ ;} \\ x & \text{if } x>0 \text{ ;} \end{cases}$$

where $\alpha=1$

ELU decreases the bias shift by pushing the mean activation towards zero.
Exponential linear units try to make the mean activations closer to zero,
allowing for faster learning while maintaining almost as much accuracy as ReLU’s.

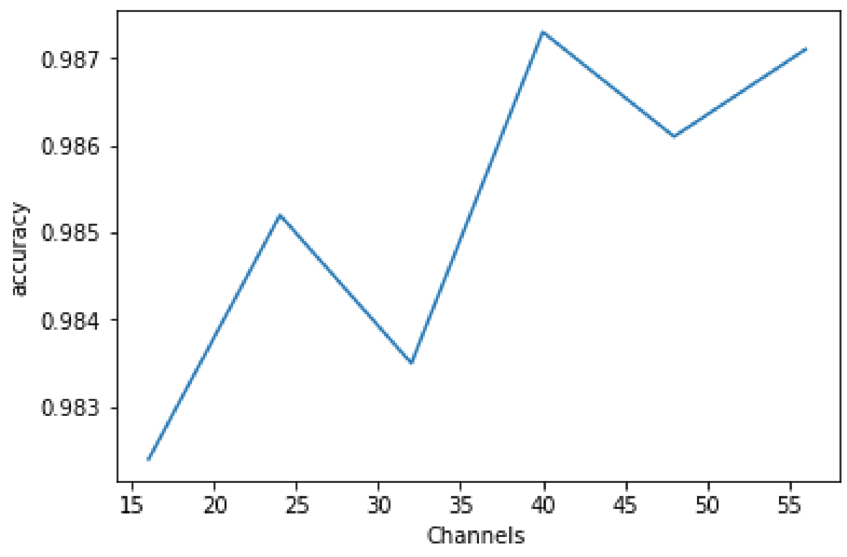
SELU – Scaled ELU

selu(x) = λ*elu(x)

where $\lambda=1.0507$ and $\alpha=1.6733$

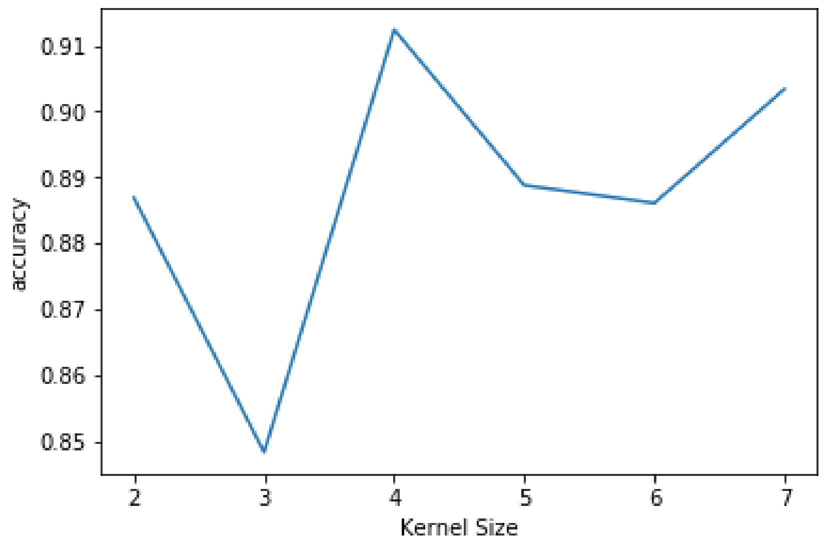
SELU has self-normalising properties because the activations that are close to zero mean and unit variance, propagated through many network layers, will converge towards zero mean and unit variance. This, in particular, makes the learning highly robust and allows to train networks that have many layers.

Relationship between no of kernel channels and accuracy:



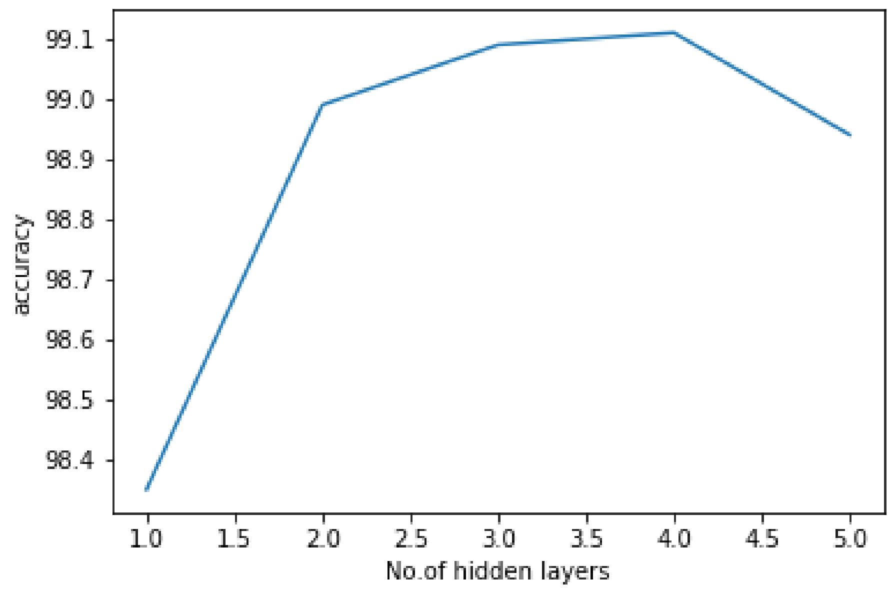
In this case each kernel will learn some feature and thus, increasing the number of kernels might cause overfitting and thus cause the accuracy to fall. In our case, it might also be the case that the test images resemble the training images to a high degree however no obvious pattern is visible tradeoff between increasing computational complexity and change in accuracy needs to be considered to decide on the number of kernels.

Relationship between size of convolution kernels and accuracy



So, increasing the kernel size doesn't follow an obvious pattern with the accuracy. However, no obvious pattern is visible so the tradeoff between increasing computational complexity and change in accuracy needs to be considered to decide on the kernel size.

Relationship between number of hidden layers and accuracy



The accuracy peaks at a given no.of hidden layers beyond which it reduces, showing that overfitting occurs and it does not generalize well to the testing data.

Overfitting:

Overfittting refers to the architecture that models training data far too well.
Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model.

We were asked to deal with the overfitting problem and elucidate possible solutions.
One solution we have implemented in the model is a Regularization Technique called

Dropout:
- It's algorithm states that at every training step, every neuron has a probability 'p' of being 'dropped out' i.e. being deactivated or ignored in that training step. This algorithm albeit being very simple works in deep neural nets.

Another, is the number of

Hidden layers:

- Increasing the number of hidden units and/or layers may lead to overfitting because it will make it easier for the neural network to memorize the training set, we have seen before that hidden layers infer more complex structures in higher level layers and many hidden layers infer the training set perfecly but do not generalize to new data.

This conforms to the graph we observed comparing the accuracies and the number of hidden layers.

Finally, we see the

Num_Epochs:

The number of epochs is the number of times you iterate over the whole training set, as a result, if your network has a large capacity (a lot of hidden units and hidden layers) the longer you train for the more likely you are to overfit.

To address this issue you can use early stopping which is when you train you neural network for as long as the error on an external validation set keeps decreasing instead of a fixed number of epochs.

Final Pointers:

We did not initialize any weights to the model, hence they are set at random.
Hence, though the accuracies may vary slighly in between multiple runs of the algorithm, they seem to oscillate about a fixed value.

The increase in accuracy values from using convolution layer is evident from the fact that they seperate the input image into deeper constructs i.e. into feature maps helping the model learn far better.

The inferences have been elucidated near the plots for better understanding.
Some notable ones: were the ReLu actiavation function being better, accuracy peaking when the hidden layers is 4 and then decreasing probably showing overfitting.