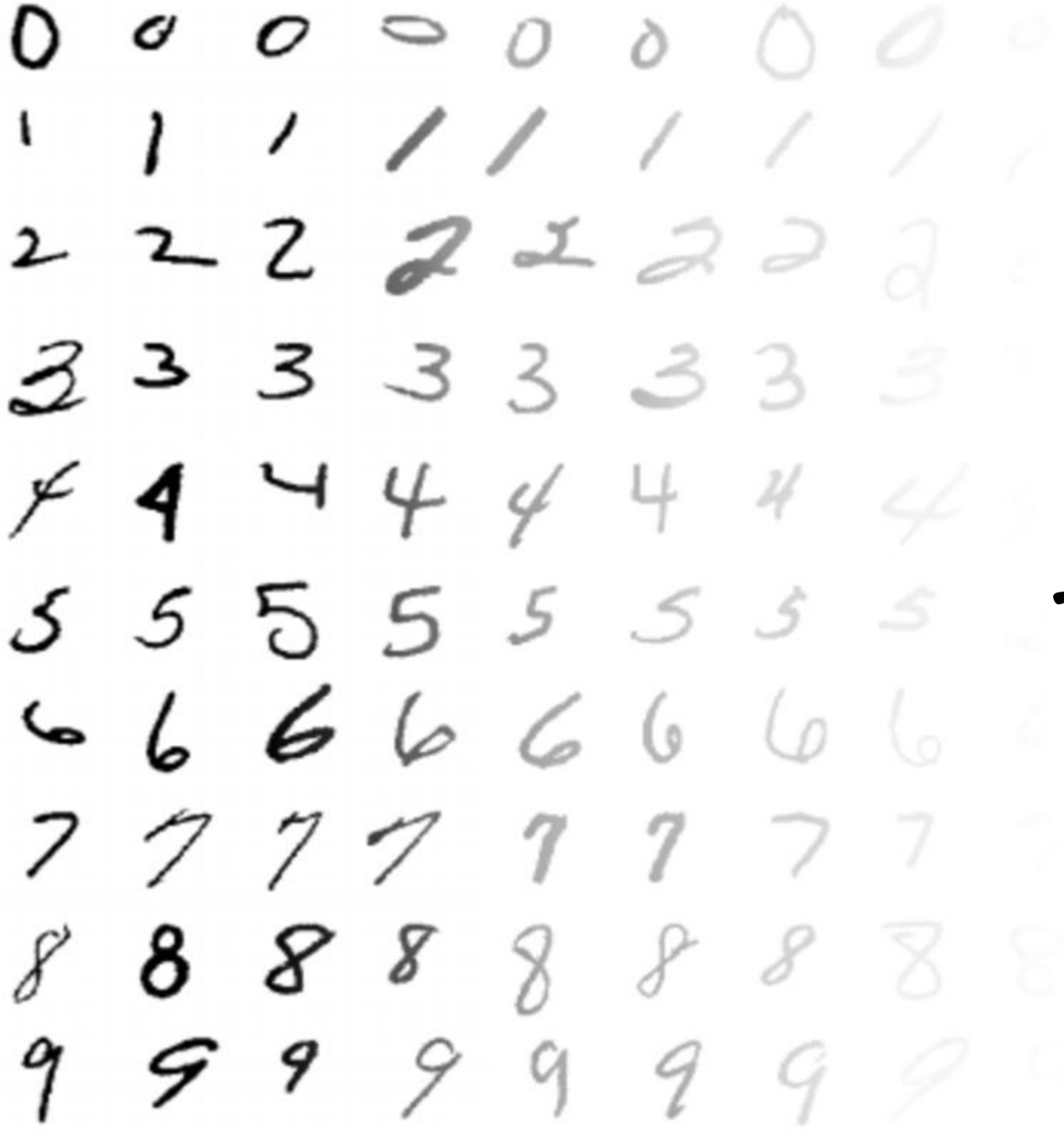
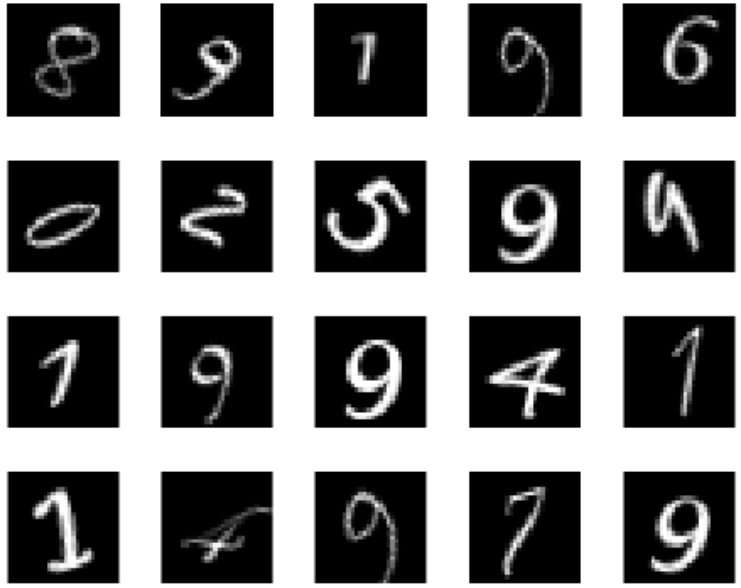


Matlab for DS and DL

Summary

1. Intro:
    a. MNIST dataset
    b. Optimizers
    c. Network Structure
2. Results:
    a. SGDM at different Batch-Sizes
    b. SGDM-ADAM-RMSProp comparison
3. Conclusions

Intro – MNIST dataset



The MNIST database (Modified National Institute of Standards and Technology) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning.

It was created by "re-mixing" the samples from NIST's original datasets. The black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.

The MNIST database contains 60,000 training images and 10,000 testing images.

Intro - optimizers

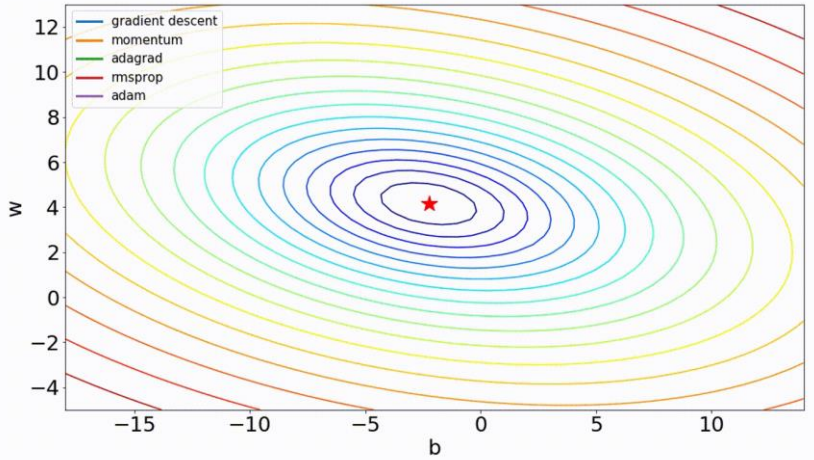
SGDM (Stochastic Gradient Descent with Momentum): The standard gradient descent algorithm updates the network parameters (weights and biases) to minimize the loss function by taking small steps at each iteration in the direction of the negative gradient of the loss. The stochastic gradient descent algorithm can oscillate along the path of steepest descent towards the optimum. Adding a momentum term to the parameter update is one way to reduce this oscillation. The stochastic gradient descent with momentum (SGDM) update is

$$V_t = V_{t-1} + \alpha \nabla_W L(W, X, y)$$
$$W_{t+1} = W_t - V_t$$

RMSProp (Root Mean Squared Propagation): Stochastic gradient descent with momentum uses a single learning rate for all the parameters. Other optimization algorithms seek to improve network training by using learning rates that differ by parameter and can automatically adapt to the loss function being optimized. RMSProp is one such algorithm. It keeps a moving average of the element-wise squares of the parameter gradients

$$V_t = \beta V_{t-1} + (1-\beta)[\nabla_W L(W, X, y)]^2$$
$$W_{t+1} = W_t - \frac{\alpha_t}{(V_t + \varepsilon)^{1/2}}[\nabla_W L(W, X, y)]$$

β is the decay rate of the moving average. Common values of the decay rate are 0.9, 0.99, and 0.999.
The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster.



ADAM (Adaptive Moment Estimation): This optimizer uses a parameter update that is similar to RMSProp, but with an added momentum term. This algorithm calculates the exponential moving average of gradients and square gradients.

$$V_t = \beta V_{t-1} + (1-\beta)\nabla_W L(W, X, y)$$
$$W_{t+1} = W_t - \alpha V_t$$

The optimizer is called Adam because uses estimations of the first and second moments of the gradient to adapt the learning rate for each weight of the neural network. Adam is proposed as the most efficient stochastic optimization which only requires first-order gradients where memory requirement is too little.

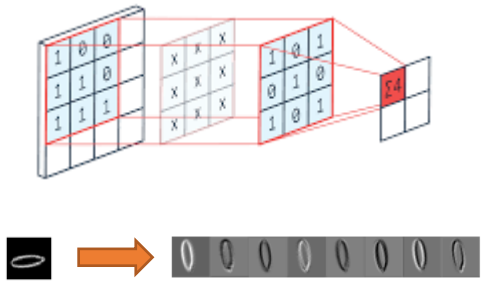
Intro - Network structure 1

Name	Type
imageinput 28×28×1 images with 'zerocenter' norm...	Image Input
conv_1 8 3×3 convolutions with stride [1 1] and ...	2-D Convolution
batchnorm_1 Batch normalization	Batch Normalization
relu_1 ReLU	ReLU
maxpool_1 2×2 max pooling with stride [2 2] and pa...	2-D Max Pooling
conv_2 16 3×3 convolutions with stride [1 1] and...	2-D Convolution
batchnorm_2 Batch normalization	Batch Normalization
relu_2 ReLU	ReLU
maxpool_2 2×2 max pooling with stride [2 2] and pa...	2-D Max Pooling
conv_3 32 3×3 convolutions with stride [1 1] and...	2-D Convolution
batchnorm_3 Batch normalization	Batch Normalization
relu_3 ReLU	ReLU
fc 10 fully connected layer	Fully Connected
softmax softmax	Softmax
classoutput crossentropyex	Classification Output

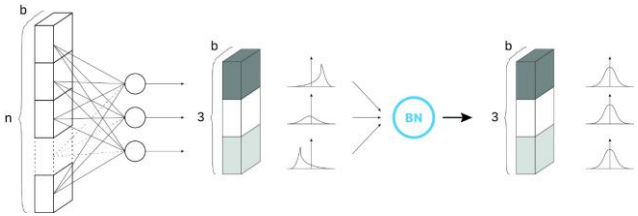
Image Input: An image input layer inputs 2-D images to a network and applies data normalization. Here is specified the size of the input data, as a row vector of integers [h w c], where h, w, and c correspond to the height, width, and number of channels respectively. In our case, is [28, 28, 1].

2-D Convolution: this layer applies sliding convolutional filters to 2-D input. The layer convolves the input by moving the filters along the input vertically and horizontally and computing the dot product of the weights and the input, and then adding a bias term.

- Filtersize: specified as a vector [h w], where h is the height and w is the width of the filter.
- Number of filters corresponds to the number of neurons in the layer that connect to the same region in the input. This parameter determines the number of filters in the layer output.
- Padding with option 'same', calculates the size of the padding at training time so that the output has the same size as the input.

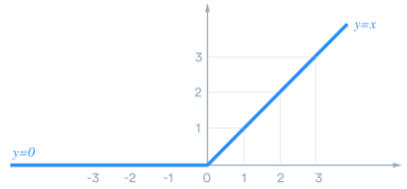


Batch Normalization: this layer normalizes a mini-batch of data across all observations for each channel independently. To speed up training of the convolutional neural network and reduce the sensitivity to network initialization, use batch normalization layers between convolutional layers and nonlinearities, such as ReLU layers.



ReLU: this layer performs a threshold operation to each element of the input, where any value less than zero is set to zero.
This operation is equivalent to:

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

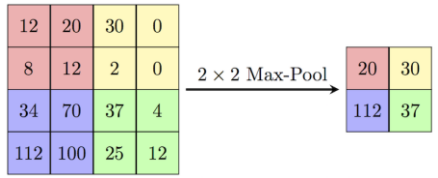


Intro - Network structure 2

Name	Type
imageinput 28×28×1 images with 'zerocenter' norm...	Image Input
conv_1 8 3×3 convolutions with stride [1 1] and ...	2-D Convolution
batchnorm_1 Batch normalization	Batch Normalization
relu_1 ReLU	ReLU
maxpool_1 2×2 max pooling with stride [2 2] and pa...	2-D Max Pooling
conv_2 16 3×3 convolutions with stride [1 1] and...	2-D Convolution
batchnorm_2 Batch normalization	Batch Normalization
relu_2 ReLU	ReLU
maxpool_2 2×2 max pooling with stride [2 2] and pa...	2-D Max Pooling
conv_3 32 3×3 convolutions with stride [1 1] and...	2-D Convolution
batchnorm_3 Batch normalization	Batch Normalization
relu_3 ReLU	ReLU
fc 10 fully connected layer	Fully Connected
softmax softmax	Softmax
classoutput crossentropyex	Classification Output

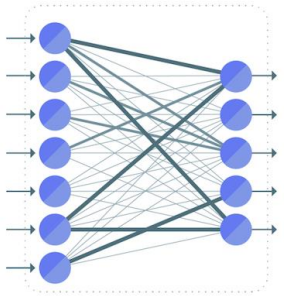
2-D Max Pooling: this layer performs downsampling by dividing the input into rectangular pooling regions, then computing the maximum of each region.

- Dimensions of the pooling regions, specified as a vector [h w], where h is the height and w is the width.
- Step size for traversing the input vertically and horizontally, specified as a vector [a b], where a is the vertical step size and b is the horizontal step size. Specifying Stride as a scalar we can use the same value for both dimensions.

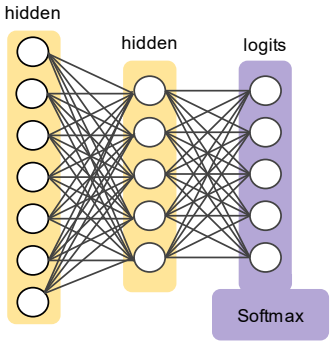


Fully Connected: the fully connected layer has neurons connected to all neurons in the preceding layer. This layer multiplies the input by a weight matrix and then adds a bias vector. It combines all the learned features to identify the larger patterns and to classify the images.

- OutputSize is equal to the number of classes in the target data. Here, the output size is 10, corresponding to the 10 classes.

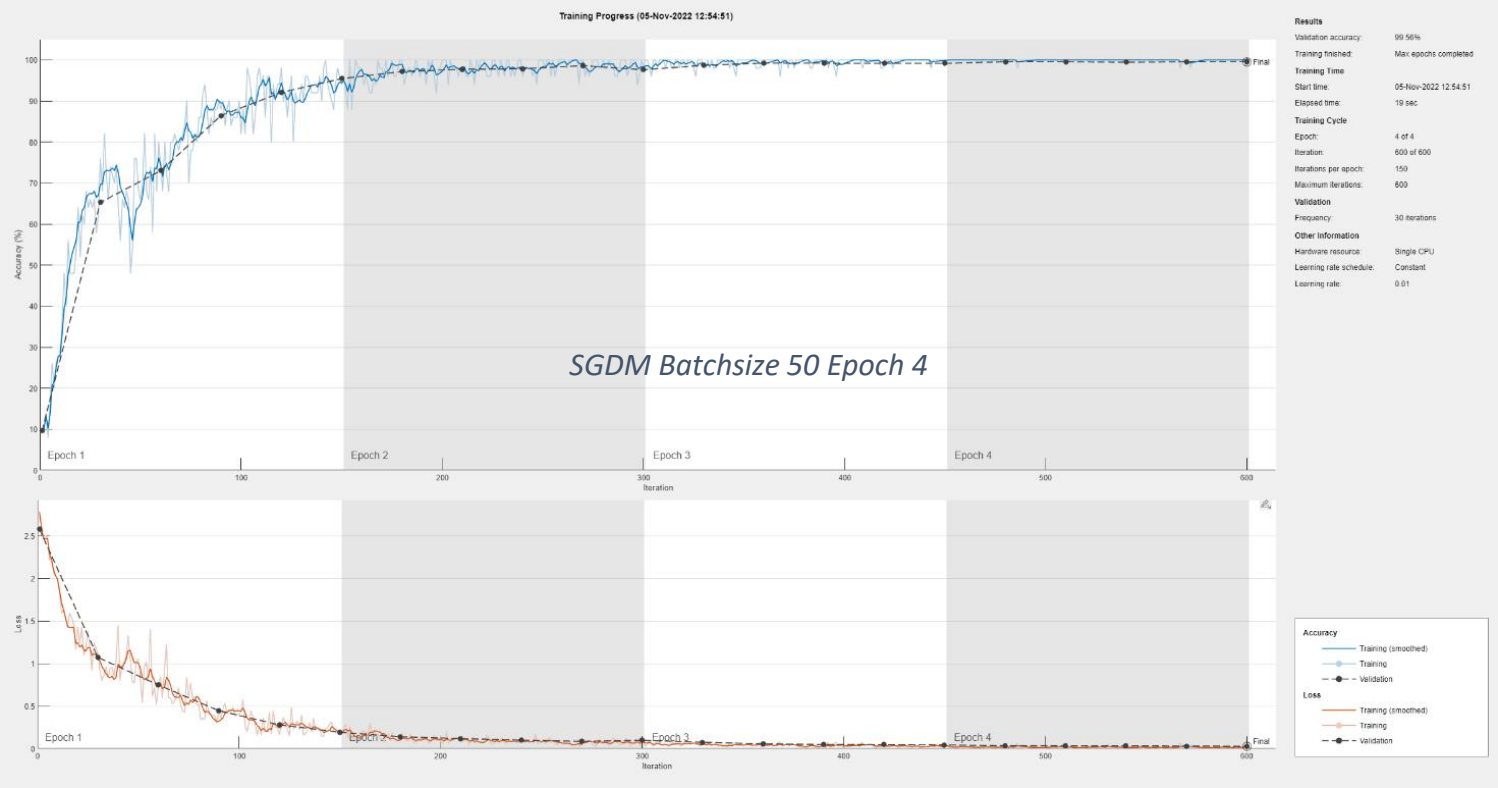


Softmax: softmax activation function normalizes the output of the fully connected layer. The output of the softmax layer consists of positive numbers that sum to one, which can then be used as classification probabilities by the classification layer.

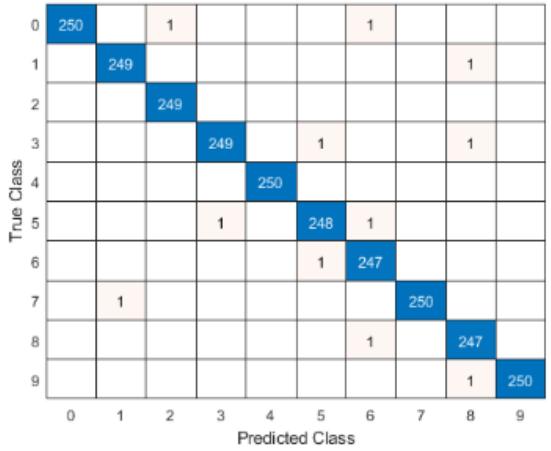


Classification Output: A classification layer computes the cross-entropy loss for classification and weighted classification tasks with mutually exclusive classes. The layer infers the number of classes from the output size of the previous layer.

Results - SGDM



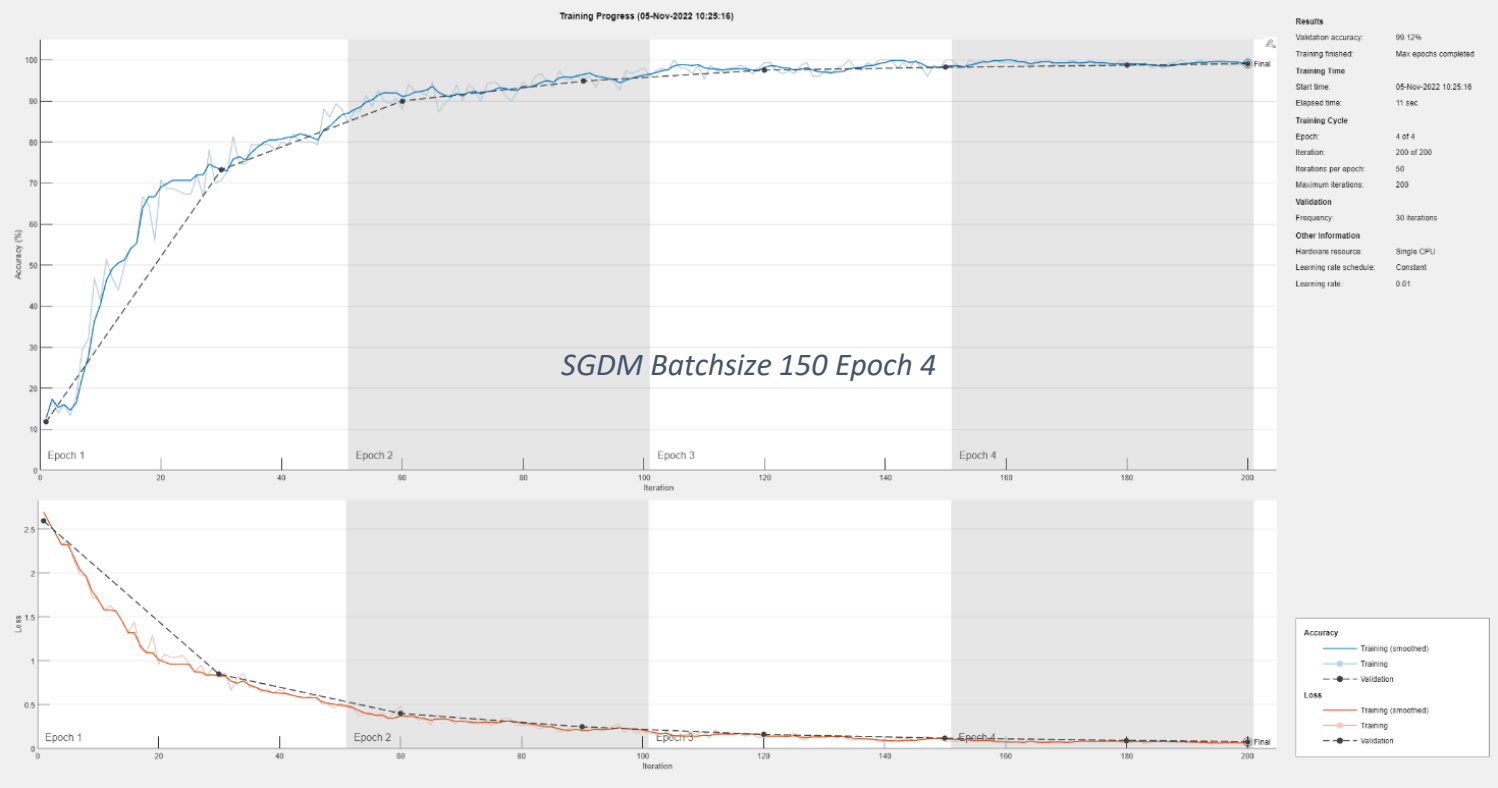
Parameter	Value
Epochs	4
Iteration	600
Iteration/Epoch	150
Elapsed time	19 sec
Val. Accuracy	99,56%
Batch-size	50



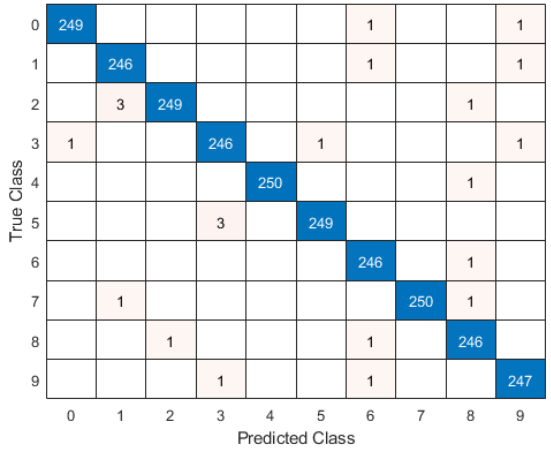
We observe:

- **Accuracy**: after about 3 epochs we reach the maximum accuracy value which is around 99.6%. The latter is very close to the training one and this means that the model is performing well even on new data.
- **Oscillations**: the oscillations are more pronounced at the beginning of the training process while they remain very low once the maximum is reached.
- **Confusion matrix:** the confusion matrix shows that the model is making very few errors on the test data.

Results - SGDM



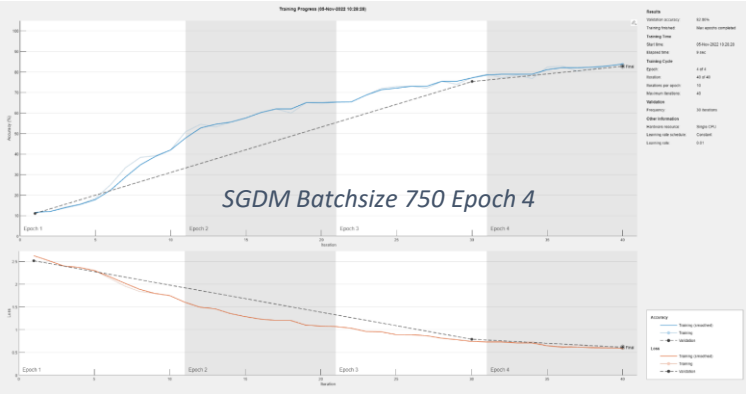
Parameter	Value
Epochs	4
Iteration	200
Iteration/Epoch	50
Elapsed time	11 sec
Val. Accuracy	99,12%
Batch-size	150

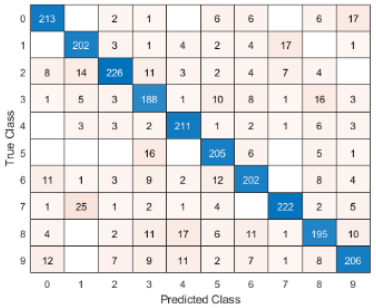


We observe:

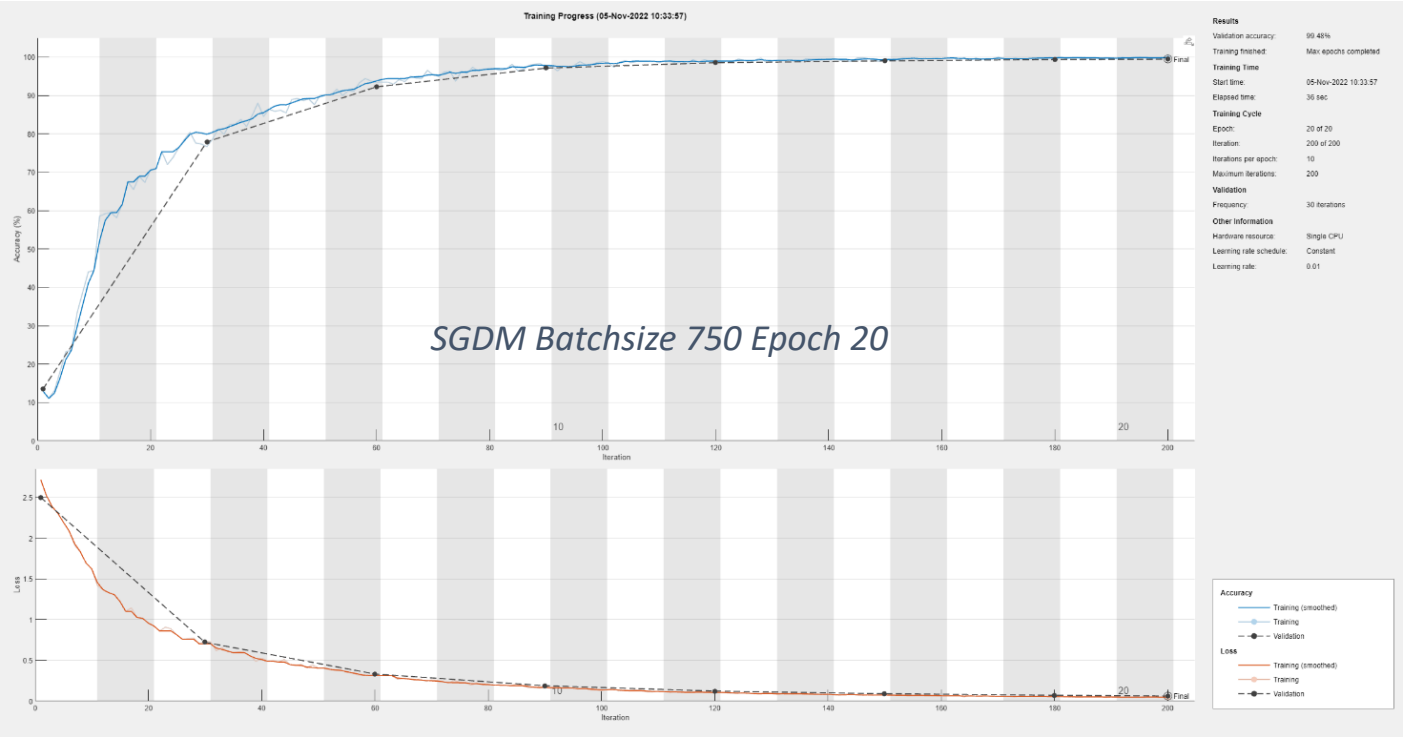
- **Accuracy**: after about 4 epochs we reach the maximum accuracy value which is around 99%. The latter is very close to the training one and this means that the model is performing well even on new data.
- **Oscillations**: the oscillations are more accentuated at the beginning of the training but remain albeit mitigated throughout the training.
- **Confusion matrix:** the confusion matrix shows that the model is making few errors on the test data. The digits on which it is more difficult to make a prediction are "1", "3", "6" and "8".

Results - SGDM

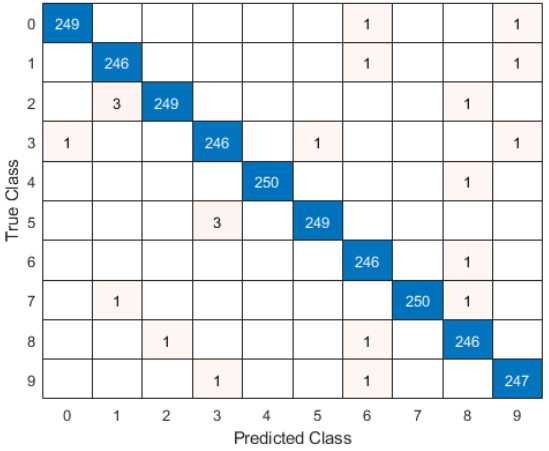




In the case with batchsize 750 we observed that 4 epochs were too few to be able to train the model properly; we observe that accuracy has not yet reached its maximum and the confusion matrix highlights many prediction errors. We decided to increase the number of epochs to 20.

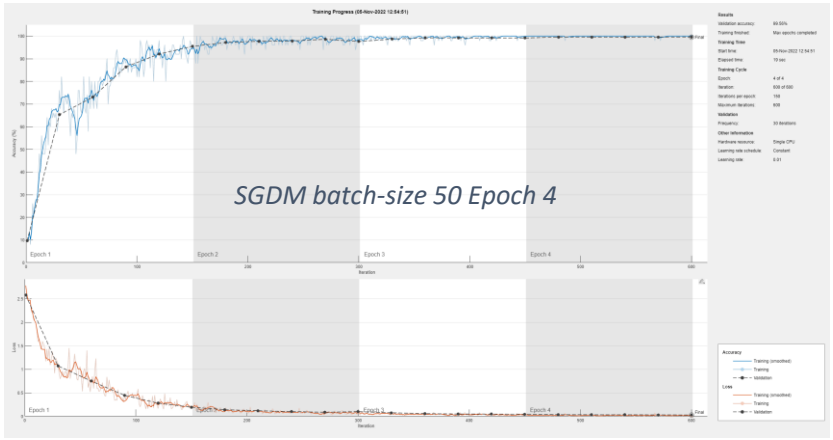


Parameter	Value
Epochs	20
Iteration	200
Iteration/Epoch	10
Elapsed time	36 sec
Val. Accuracy	99,48%
Batch-size	750

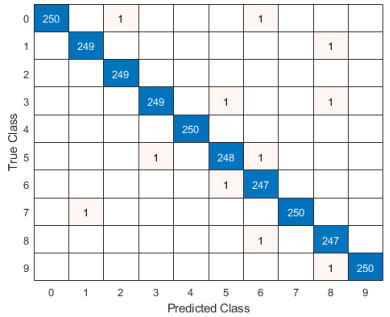


We Observe:

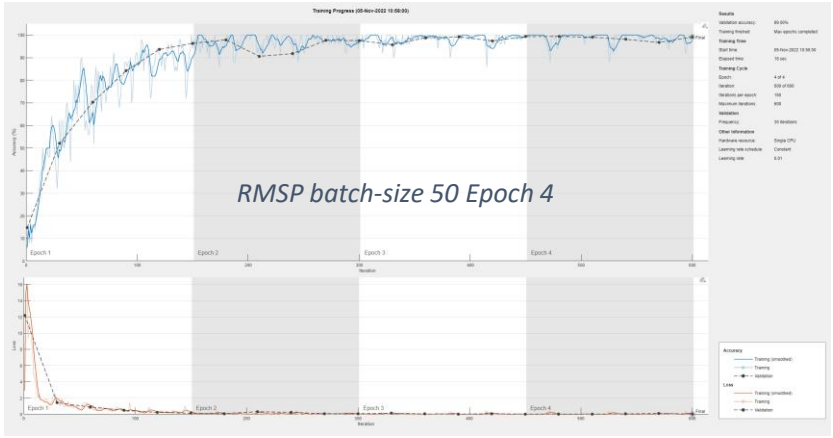
- **Accuracy**: after about 12 epochs we reach the maximum accuracy value which is around 99%. The latter is very close to the training one and this means that the model is performing well even on new data on which it has not trained.
- **Oscillations**: this is the configuration that has the least fluctuations since the beginning of the training.
- **Confusion matrix:** the confusion matrix shows that the model is correctly classifying the test data; compared to the previous case we have more prediction errors but always in minimal quantities. The digits on which it is most difficult to make predictions are "1", "3", "6" and "8".



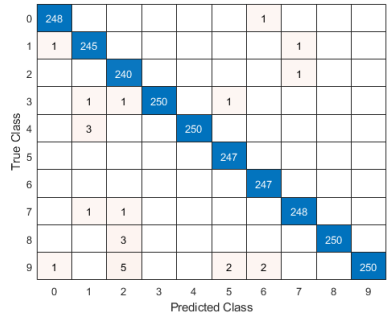
SGDM



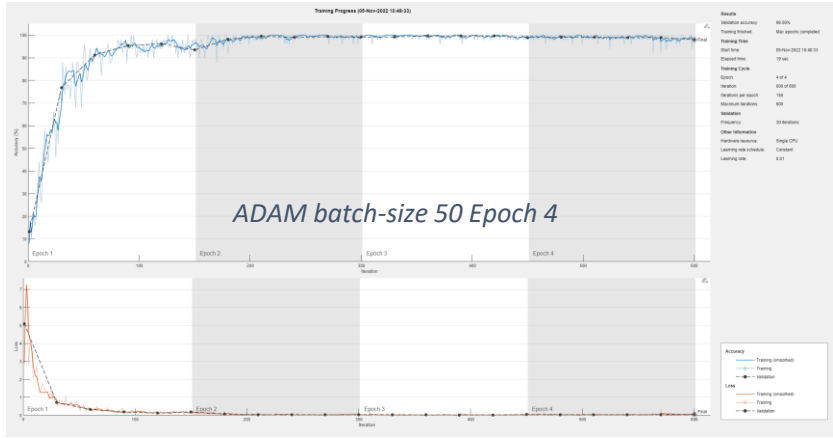
Parameter	Value
Epochs	4
Iteration	600
Iteration/Epoch	150
Elapsed time	19 sec
Val. Accuracy	99,56%
Batch-size	50



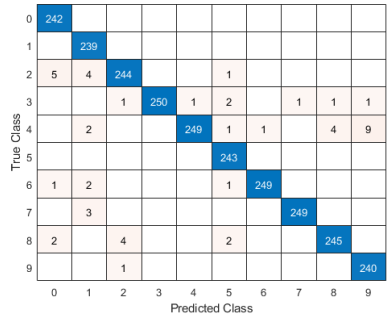
RMSP



Parameter	Value
Epochs	4
Iteration	600
Iteration/Epoch	150
Elapsed time	18 sec
Val. Accuracy	99%
Batch-size	50



ADAM



Parameter	Value
Epochs	4
Iteration	600
Iteration/Epoch	150
Elapsed time	19 sec
Val. Accuracy	98%
Batch-size	50

Results - Optimizers comparison

With the same batch-size and number of epochs we observe that the three optimization algorithms examined show the following characteristics:

- **Accuracy:** The model that shows greater accuracy is the SGDM with a value of 0.9956. However, all three models reach very high levels, all affirming themselves between 98 and 99% accuracy. This means that in all models in approximately 99% of cases the digits are correctly labeled.
- **Confusion matrix:** through the confusion matrix we can highlight which models make greater prediction errors. Adam, who is the model with the least accuracy, shows the highest number of errors.; digits 0 and 9 are the most difficult to label.
- **Elapsed time:** as regards the time taken to complete all 4 epochs, it is clear that all the models are very fast: the elapsed time is about 19 seconds.
- **Oscillations**: the RMSP algorithm seems to show greater fluctuations along the entire duration of the training.

Summary of results

Scenario	Algorithm	Minibatch size	Epochs	Oscillations	Accuracy	Elapsed time (sec)
A-1	SGDM	50	4	High	0.9956	19
A-2	SGDM	50	10	High	0.9984	46
A-3	SGDM	150	4	High	0.9912	11
A-4	SGDM	150	10	High	0.9968	24
A-5	SGDM	750	4	Low	0.8280	9
A-6	SGDM	750	20	Low	0.9948	36
B-1	ADAM	50	4	High	0.98	19
B-2	ADAM	150	4	High	0.9980	10
B-3	ADAM	750	20	Medium	0.9980	36
C-1	RMSP	50	4	Very High	0.99	18
C-2	RMSP	150	10	Very High	0.9976	24
C-3	RMSP	750	20	Very High	0.9976	36

Conclusions:

We performed the training and validation of the models using the SGDM, ADAM and RMSProp optimization algorithms with different values of the configuration parameters.
Here is a summary table with all the parameters and performances achieved in terms of accuracy, elapsed time and fluctuations.

Using the same neural network structure, we changed the following parameters: the number of epochs and the size of the minibatches.

Increasing the number of epochs we observe an oscillations decrease.

As regards the minibatch size, if we increase it significantly (e.g. from 150 to 750), it is necessary to increase the number of epochs to keep the accuracy high.

There are no appreciable differences on the elapsed-time since the dataset we are using has a limited size.