Optimizer selection for CNN in handwritten digit recognition

Wenting Liu, Shuangyu Zhao, Yuhan Zhu, Tiancheng Liu, Yan Lyu

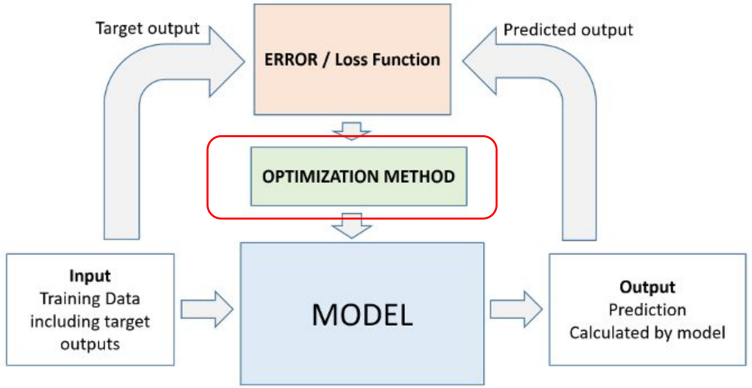
Michigan State University

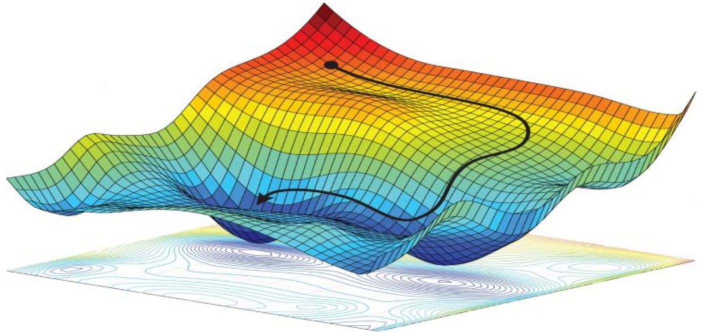
outline

- Introduction
- Optimizer selection
  - “SGD”
  - “momentum”
  - “RMSprop”
  - “ADAM
- Conclusion and future discussion

Introduction

- The objective of no matter machine learning models or deep learning models is prediction, and a crucial aspect of achieving better predictions is identifying a dependable **optimizer**.
- In this project, we aim to study different techniques for **optimizing Convolutional Neural Networks (CNNs) in handwritten digit recognition**.

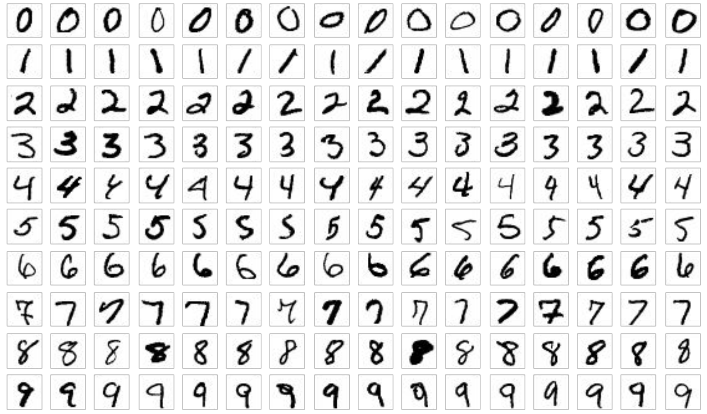




https://www.deepnetts.com/blog/from-basic-machine-learning-to-deep-learning-in-5-minutes.html
https://medium.com/analytics-vidhya/optimization-acb996a4623c

Introduction – Dataset

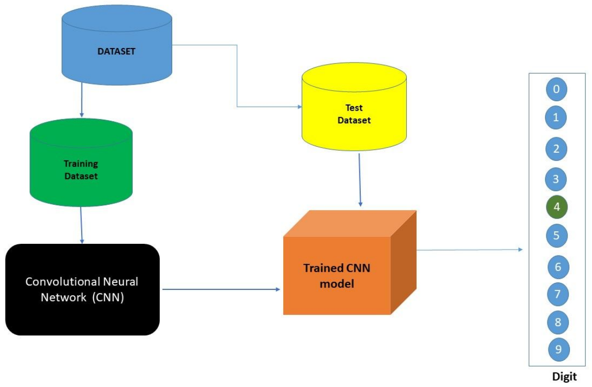
- The MNIST dataset, consisting of 70,000 handwritten digit images, with **60,000** images in the **training** set and **10,000** images in the **testing** set, has been used extensively for evaluating and comparing various machine learning algorithms, particularly for image classification tasks.



https://www.analyticsvidhya.com/blog/2021/07/classification-of-handwritten-digits-using-cnn/

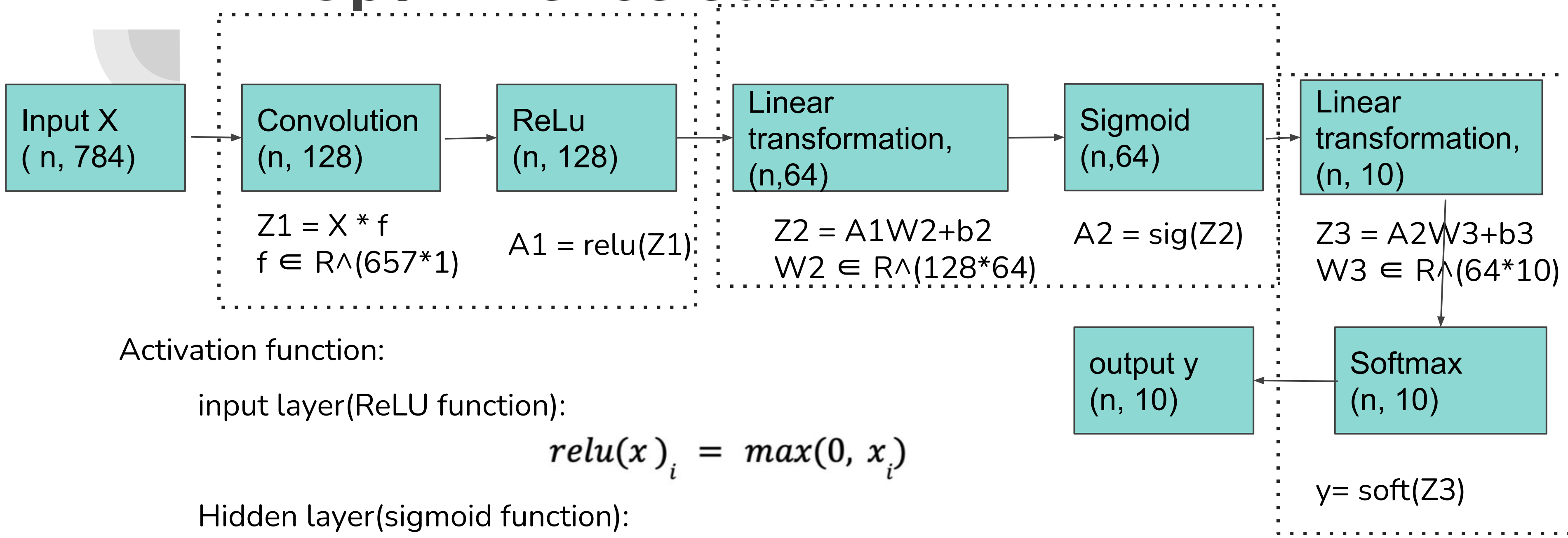
Introduction – Convolutional Neural Network (CNNs)

- CNNs is a deep learning technique to classify the input automatically. It has shown **remarkable success** in image recognition.
- However, the **high computational cost** and **memory requirements** of CNNs have become a major challenge.
- Therefore, the **optimization** of CNNs is crucial to reduce the computational complexity and memory footprint while maintaining their accuracy.



https://www.analyticsvidhya.com/blog/2021/07/classification-of-handwritten-digits-using-cnn/

• Optimizer selection



Activation function:

input layer(ReLU function):

$$relu(x)_i = max(0, x_i)$$

Hidden layer(sigmoid function):

$$sig(x)_i = \frac{1}{1+e^{-x_i}}$$

Output layer(softmax function)

$$soft(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

- Optimizer in CNN is used to calculate the vector for convolution, weights, biases and learning rate in order to reduce losses

    - Stochastic Gradient Descent (SGD)
    - Momentum
    - RMSprop
    - Adam

$$\begin{tikzcd}
\min \sum_{i}^{nrows} \| y_i - y_{targeti} \|_2^2 \arrow[d, "\text{soft}(y) = y"] \\
\min \sum_{i}^{nrows} \| y_i - Z_{3i} \|_2^2 \arrow[d] \\
\boxed{\min_{f, W_2, b_2, W_3, b_3} \sum_{i}^{nrows} \| y_i - \{sig[relu(X * f)W_2 + b_2]W_3 + b_3\}_i \|_2^2}
\end{tikzcd}$$

SGD

- According to the grid search, the best drop out rate is 0.1,
- Algorithm
  - Set batch size=200, so choose subsets with 200 rows and do gradient descent by subset.

$$f_j := f_j - t_{jk}\frac{\partial \sum_{i}^{nrows} \|y_i - \{sig[relu(X^*f)W_2+b_2]W_3+b_3\}_i\|_2^2}{\partial f}$$

$$W_{2j} := W_{2j} - t_{jk}\frac{\partial \sum_{i}^{nrows} \|y_i - \{sig[relu(X^*f)W_2+b_2]W_3+b_3\}_i\|_2^2}{\partial W_2}$$

$$b_{2j} := b_{2j} - t_{jk}\frac{\partial \sum_{i}^{nrows} \|y_i - \{sig[relu(X^*f)W_2+b_2]W_3+b_3\}_i\|_2^2}{\partial b_2}$$

$$W_{3j} := W_{3j} - t_{jk}\frac{\partial \sum_{i}^{nrows} \|y_i - \{sig[relu(X^*f)W_2+b_2]W_3+b_3\}_i\|_2^2}{\partial W_3}$$

$$b_{3j} := b_{3j} - t_{jk}\frac{\partial \sum_{i}^{nrows} \|y_i - \{sig[relu(X^*f)W_2+b_2]W_3+b_3\}_i\|_2^2}{\partial b_3}$$

$$f = avg(f_j)$$

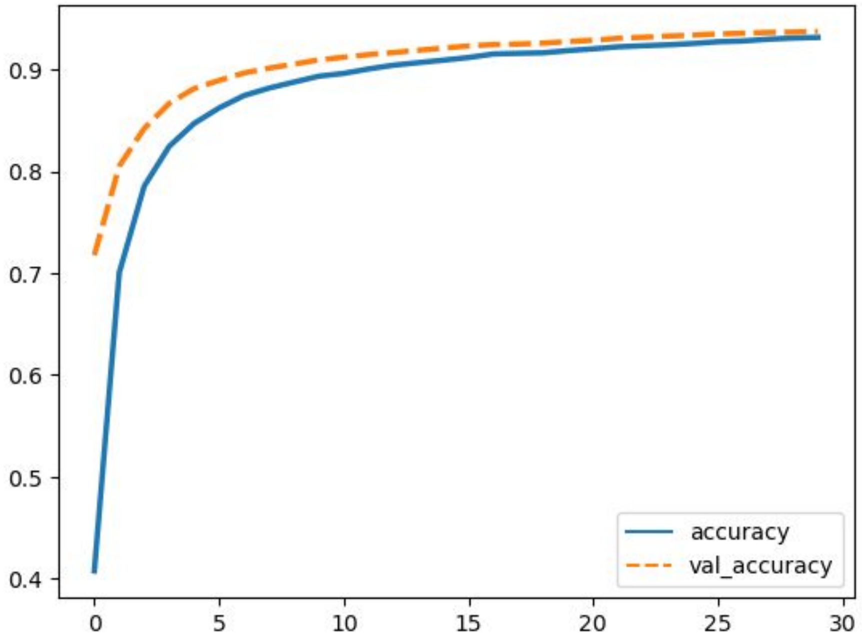
$$W_2 = avg(W_{2j})$$

$$b_2 = avg(b_{2j})$$

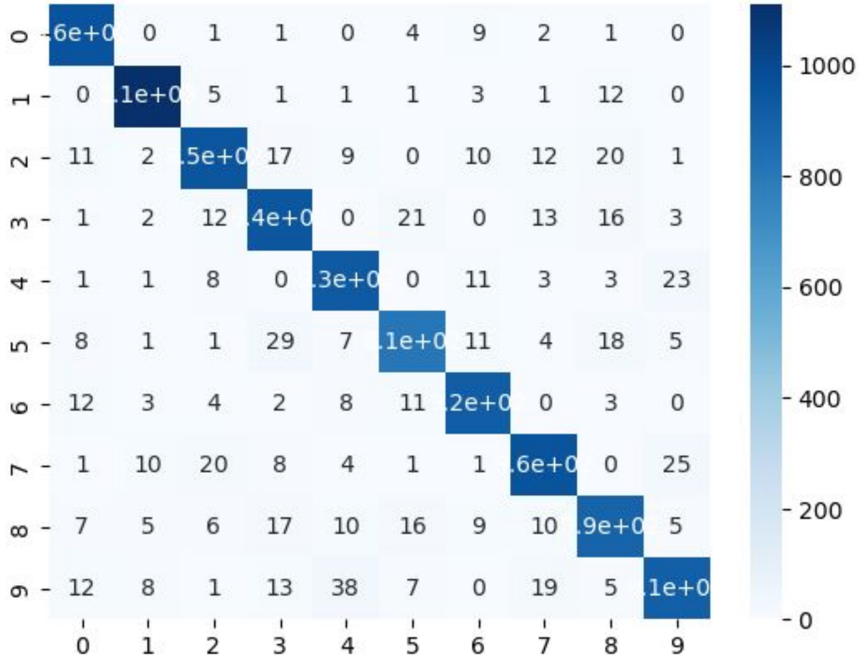
$$W_3 = avg(W_{3j})$$

$$b_3 = avg(b_{3j})$$

- SGD
  - Result visualization



true



	0	1	2	3	4	5	6	7	8	9
0	.6e+0	0	1	1	0	4	9	2	1	0
1	0	.1e+0	5	1	1	1	3	1	12	0
2	11	2	.5e+0	17	9	0	10	12	20	1
3	1	2	12	.4e+0	0	21	0	13	16	3
4	1	1	8	0	.3e+0	0	11	3	3	23
5	8	1	1	29	7	.1e+0	11	4	18	5
6	12	3	4	2	8	11	.2e+0	0	3	0
7	1	10	20	8	4	1	1	.6e+0	0	25
8	7	5	6	17	10	16	9	10	.9e+0	5
9	12	8	1	13	38	7	0	19	5	.1e+0

prediction

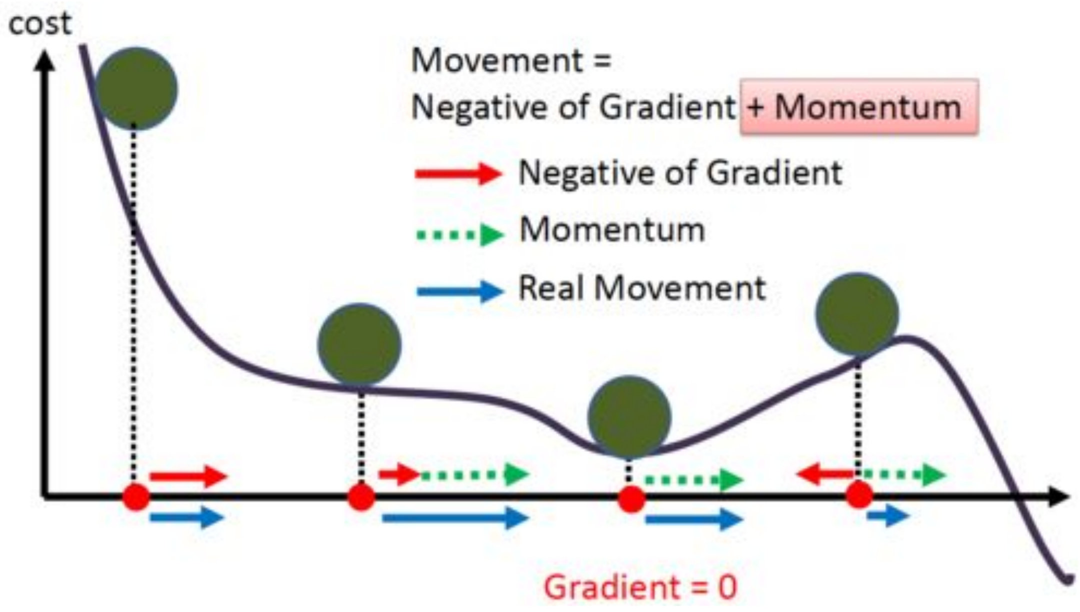
- SGD
  - Model estimation

	precision	recall	F1 score	support
0	0.9478	0.9816	0.9644	980
1	0.9720	0.9789	0.9754	1135
2	0.9425	0.9205	0.9314	1032
3	0.9146	0.9327	0.9235	1010
4	0.9237	0.9491	0.9362	982
5	0.9298	0.9058	0.9177	892
6	0.9443	0.9552	0.9497	958
7	0.9374	0.9319	0.9345	1028
8	0.9193	0.9127	0.9160	974
9	0.9360	0.8979	0.9165	1009
accuracy	0.9373			10000

Momentum

What is momentum?

- In Physics
- In Optimization



Momentum

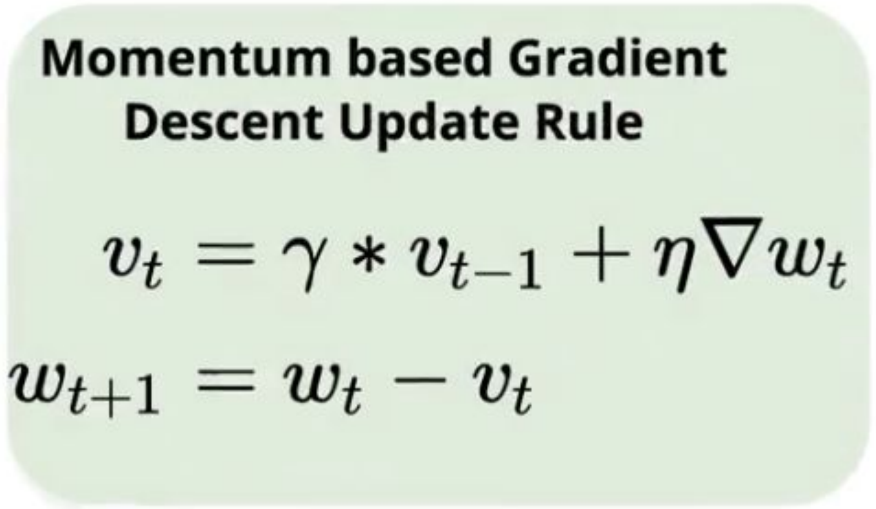
γ(Gamma): momentum parameter

η(Eta): learning rate

ω_t(Omega): current value for model

parameter

v_t: update vector at time step “t”



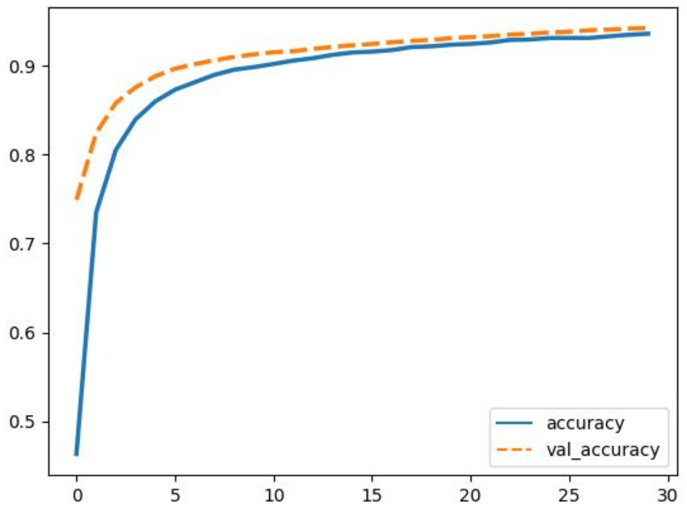
Momentum based Gradient Descent Update Rule

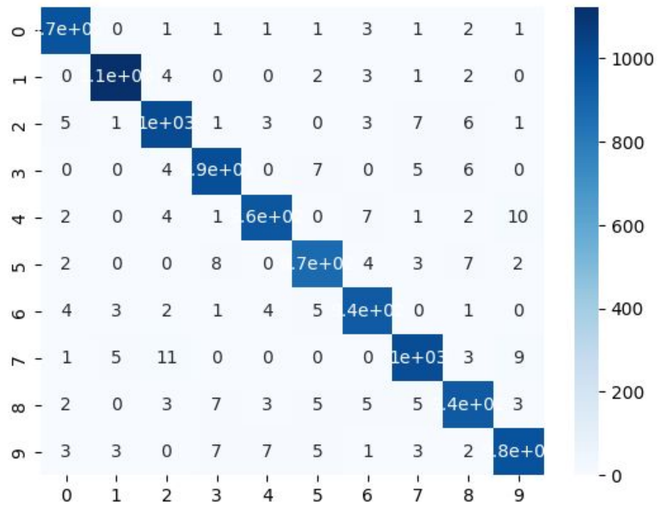
$$v_t = \gamma * v_{t-1} + \eta \nabla w_t$$

$$w_{t+1} = w_t - v_t$$

Result

Best result is 94.23% accuracy, with batch size of 200 and drop rate of 0.1.

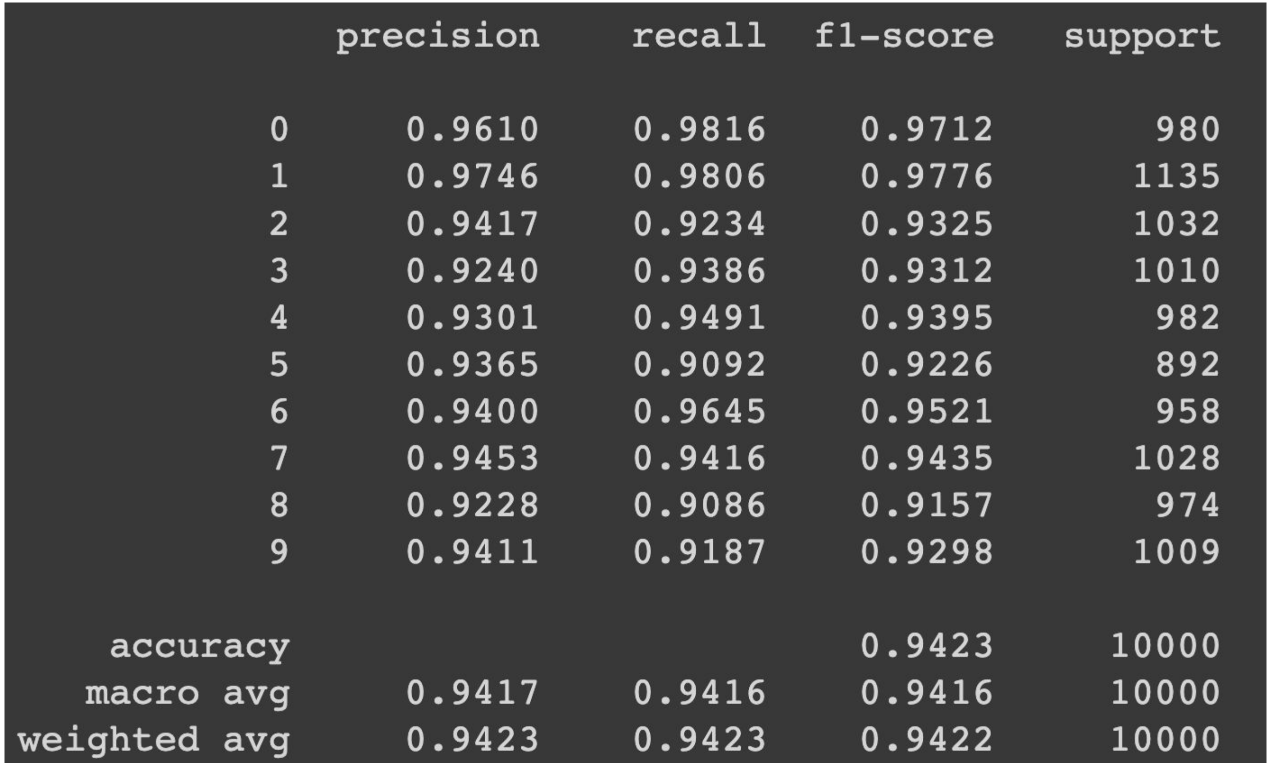




	0	1	2	3	4	5	6	7	8	9
0	.7e+0	0	1	1	1	1	3	1	2	1
1	0	.1e+0	4	0	0	2	3	1	2	0
2	5	1	1e+03	1	3	0	3	7	6	1
3	0	0	4	.9e+0	0	7	0	5	6	0
4	2	0	4	1	.6e+0	0	7	1	2	10
5	2	0	0	8	0	.7e+0	4	3	7	2
6	4	3	2	1	4	5	.4e+0	0	1	0
7	1	5	11	0	0	0	0	1e+03	3	9
8	2	0	3	7	3	5	5	5	.4e+0	3
9	3	3	0	7	7	5	1	3	2	.8e+0

Momentum

- **Model estimation**



```
              precision    recall  f1-score   support

           0     0.9610    0.9816    0.9712       980
           1     0.9746    0.9806    0.9776      1135
           2     0.9417    0.9234    0.9325      1032
           3     0.9240    0.9386    0.9312      1010
           4     0.9301    0.9491    0.9395       982
           5     0.9365    0.9092    0.9226       892
           6     0.9400    0.9645    0.9521       958
           7     0.9453    0.9416    0.9435      1028
           8     0.9228    0.9086    0.9157       974
           9     0.9411    0.9187    0.9298      1009

    accuracy                         0.9423     10000
   macro avg     0.9417    0.9416    0.9416     10000
weighted avg     0.9423    0.9423    0.9422     10000
```

● RMSprop(Root Mean Square Propagation)

Wt = weights at time t

Wt+1 = weights at time t+1

αt = learning rate at time t

∂L = derivative of Loss Function

∂Wt = derivative of weights at time t

Vt = sum of square of past gradients. [i.e sum(∂L/∂Wt-1)] (initially, Vt = 0)

β = Moving average parameter (const, 0.9)

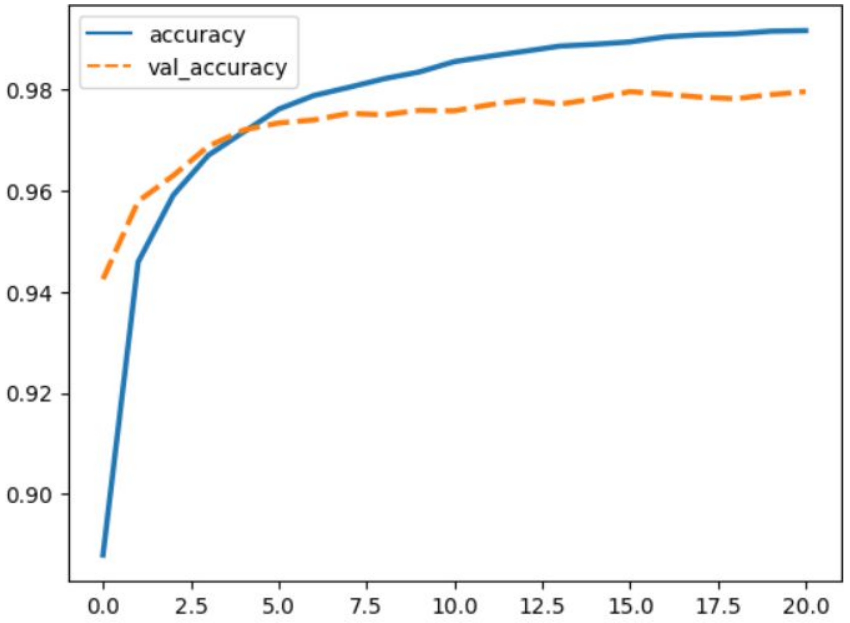
 = A small positive constant (10-8)

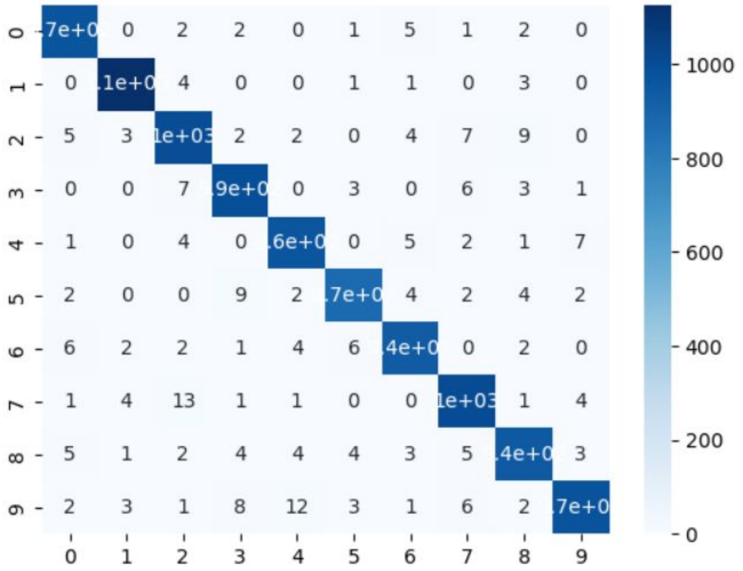
$$v_t = \beta v_{t-1} + (1-\beta) * \left[\frac{\delta L}{\delta w_t}\right]^2$$

$$w_{t+1} = w_t - \frac{\alpha_t}{(v_t+\varepsilon)^{1/2}} * \left[\frac{\delta L}{\delta w_t}\right]$$

● RMSprop

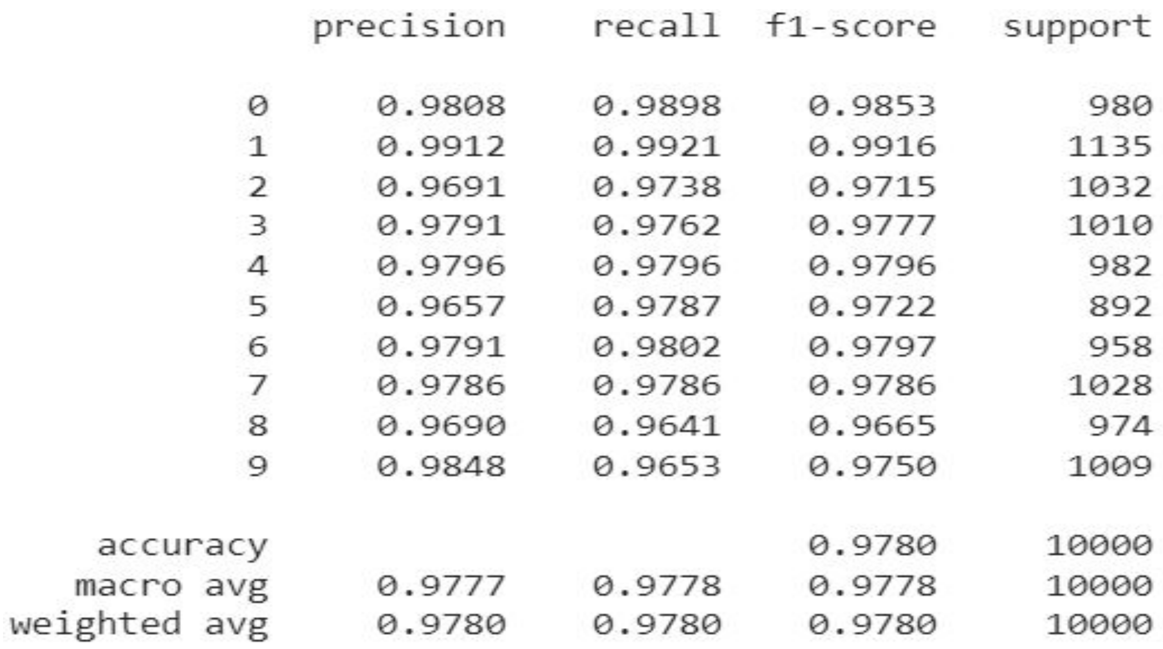
With `'batch_size': 200, 'dropout_rate': 0.1, 'epochs': 10`





- **RMSprop**

Model estimation



```
              precision    recall  f1-score   support

           0     0.9808    0.9898    0.9853       980
           1     0.9912    0.9921    0.9916      1135
           2     0.9691    0.9738    0.9715      1032
           3     0.9791    0.9762    0.9777      1010
           4     0.9796    0.9796    0.9796       982
           5     0.9657    0.9787    0.9722       892
           6     0.9791    0.9802    0.9797       958
           7     0.9786    0.9786    0.9786      1028
           8     0.9690    0.9641    0.9665       974
           9     0.9848    0.9653    0.9750      1009

    accuracy                         0.9780     10000
   macro avg     0.9777    0.9778    0.9778     10000
weighted avg     0.9780    0.9780    0.9780     10000
```

Optimizer selection

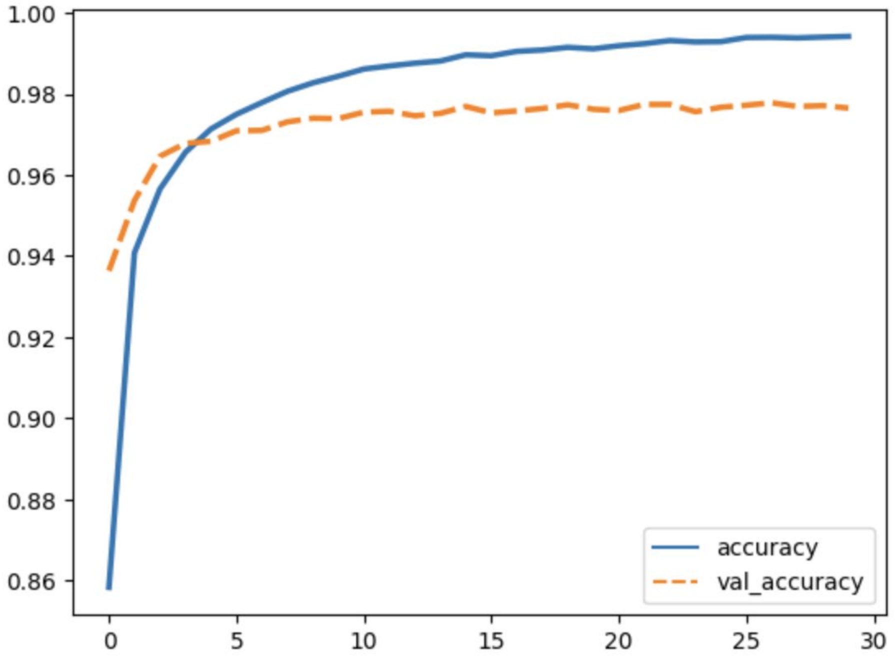
- Adam

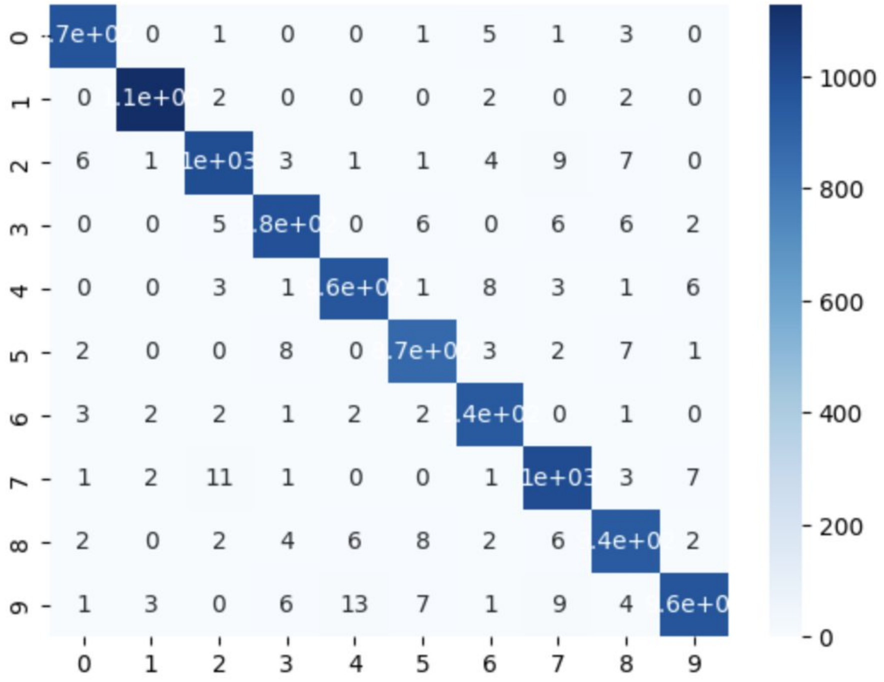
Adaptive Moment Estimation is variant of stochastic gradient descent (SGD) that combines ideas from two other optimization methods, AdaGrad and RMSProp

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left[\frac{\delta L}{\delta w_t} \right] v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left[\frac{\delta L}{\delta w_t} \right]^2$$

Adam

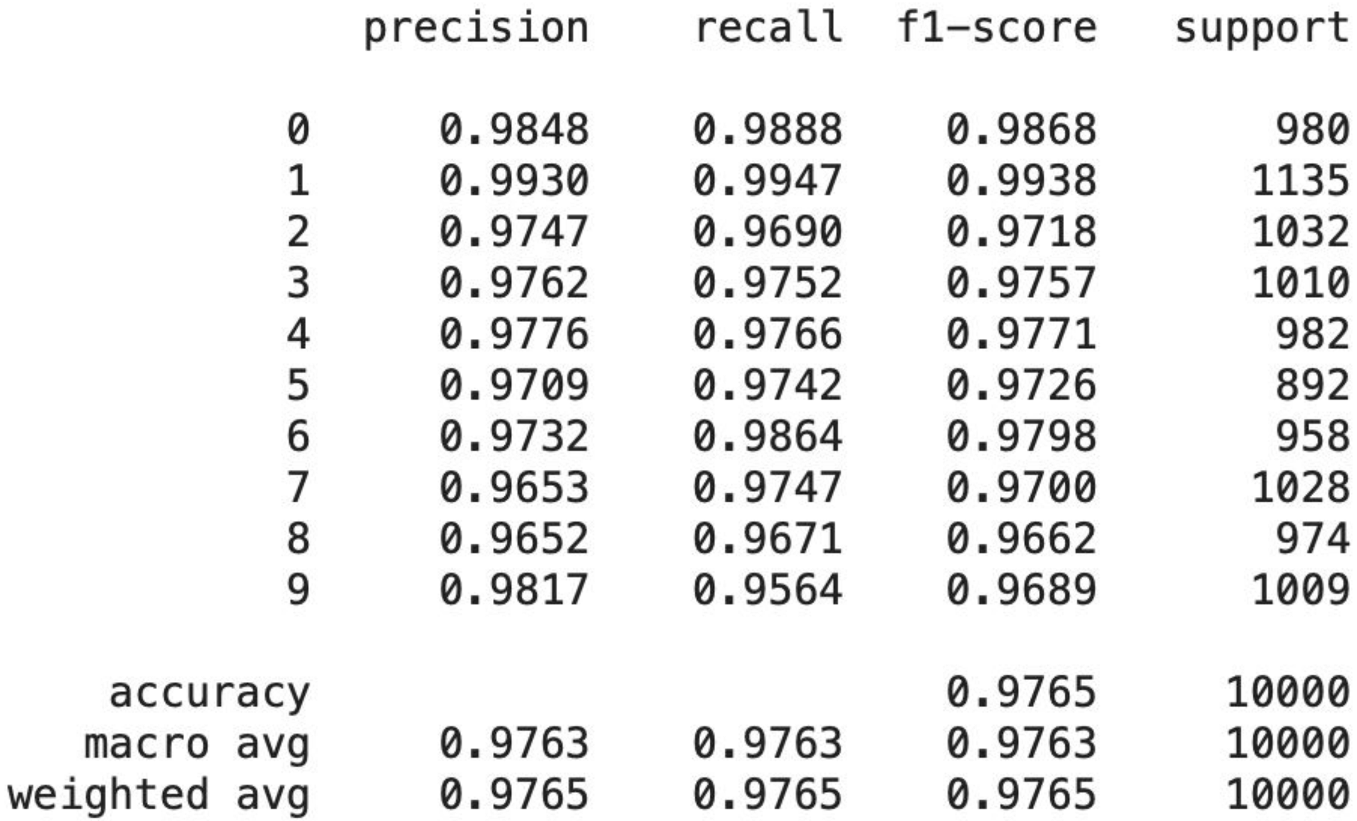
```
When 'batch_size': 200, 'dropout_rate': 0.1
```





	0	1	2	3	4	5	6	7	8	9
0	.7e+0	0	1	0	0	1	5	1	3	0
1	0	.1e+0	2	0	0	0	2	0	2	0
2	6	1	1e+03	3	1	1	4	9	7	0
3	0	0	5	.8e+0	0	6	0	6	6	2
4	0	0	3	1	.6e+0	1	8	3	1	6
5	2	0	0	8	0	.7e+0	3	2	7	1
6	3	2	2	1	2	2	.4e+0	0	1	0
7	1	2	11	1	0	0	1	1e+03	3	7
8	2	0	2	4	6	8	2	6	.4e+0	2
9	1	3	0	6	13	7	1	9	4	.6e+0

Adam



```
              precision    recall  f1-score   support

           0     0.9848    0.9888    0.9868       980
           1     0.9930    0.9947    0.9938      1135
           2     0.9747    0.9690    0.9718      1032
           3     0.9762    0.9752    0.9757      1010
           4     0.9776    0.9766    0.9771       982
           5     0.9709    0.9742    0.9726       892
           6     0.9732    0.9864    0.9798       958
           7     0.9653    0.9747    0.9700      1028
           8     0.9652    0.9671    0.9662       974
           9     0.9817    0.9564    0.9689      1009

    accuracy                         0.9765     10000
   macro avg     0.9763    0.9763    0.9763     10000
weighted avg     0.9765    0.9765    0.9765     10000
```

Conclusion and Future Discussion

Optimizer	**Accuracy**	**Recall**	**Precision**	**F1**	**Convergence Iteration Epochs**
SGD	0.9373	0.9376	**0.9376**	0.9375	5
Momentum	0.9423	0.9416	**0.9417**	0.9416	5
RMSprop	***0.9780***	***0.9778***	***0.9777***	***0.9778***	***2.5***
ADAM	0.9763	0.9763	**0.9763**	0.9765	3

Conclusion and Future Discussion

- Try ***Orthogonal Learning Chaotic Grey Wolf Optimization (CNN-OLCGWO)*** Method, which comes from “*An effective digit recognition model using enhanced convolutional neural network based chaotic grey wolf optimization*”.

Thank for your listening

Any Question?😁