Letter Recognition

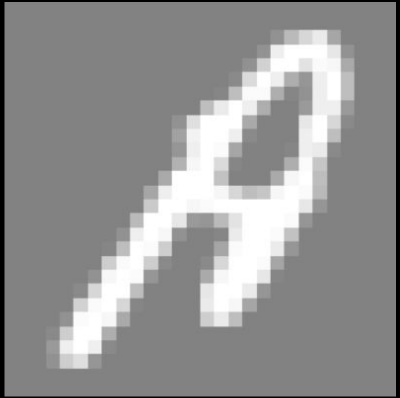
1.
Introduction

Members

MSSV	Name	Work
HE150402	Tô Văn Đức	Model + Application
HE150303	Trần Đức Tuấn	Dataset + Model
HE150258	Nguyễn Trung Nghĩa	Dataset + Application

Problem

- Letter Recognition:
  - Recognize letters from 28x28 images
  - Each image contains only one letter



2.
Dataset

Data preparation and exploration

Dataset

→ Source: https://www.nist.gov/itl/products-and-services/emnist-dataset

Dataset Summary

There are six different splits provided in this dataset. A short summary of the dataset is provided below:

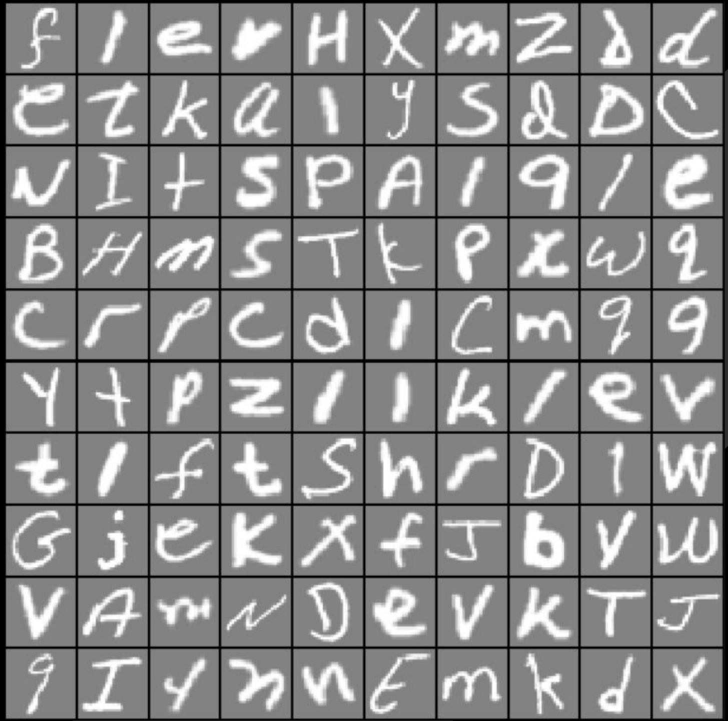
- EMNIST ByClass: 814,255 characters. 62 unbalanced classes.
- EMNIST ByMerge: 814,255 characters. 47 unbalanced classes.
- EMNIST Balanced: 131,600 characters. 47 balanced classes.
- EMNIST Letters: 145,600 characters. 26 balanced classes.
- EMNIST Digits: 280,000 characters. 10 balanced classes.
- EMNIST MNIST: 70,000 characters. 10 balanced classes.

The full complement of the NIST Special Database 19 is available in the ByClass and ByMerge splits. The EMNIST Balanced dataset contains a set of characters with an equal number of samples per class. The EMNIST Letters dataset merges a balanced set of the uppercase and lowercase letters into a single 26-class task. The EMNIST Digits and EMNIST MNIST dataset provide balanced handwritten digit datasets directly compatible with the original MNIST dataset.

Please refer to the EMNIST paper [PDF , BIB]for further details of the dataset structure.

Dataset

➔ Sample:



Dataset

- Overview:
  - Total: **145,600** images
  - Number of classes: **26**
  - Training Set:
    - **124,800** images
    - **4,800** images / class
  - Test Set:
    - **20,800** images
    - **800** images / class

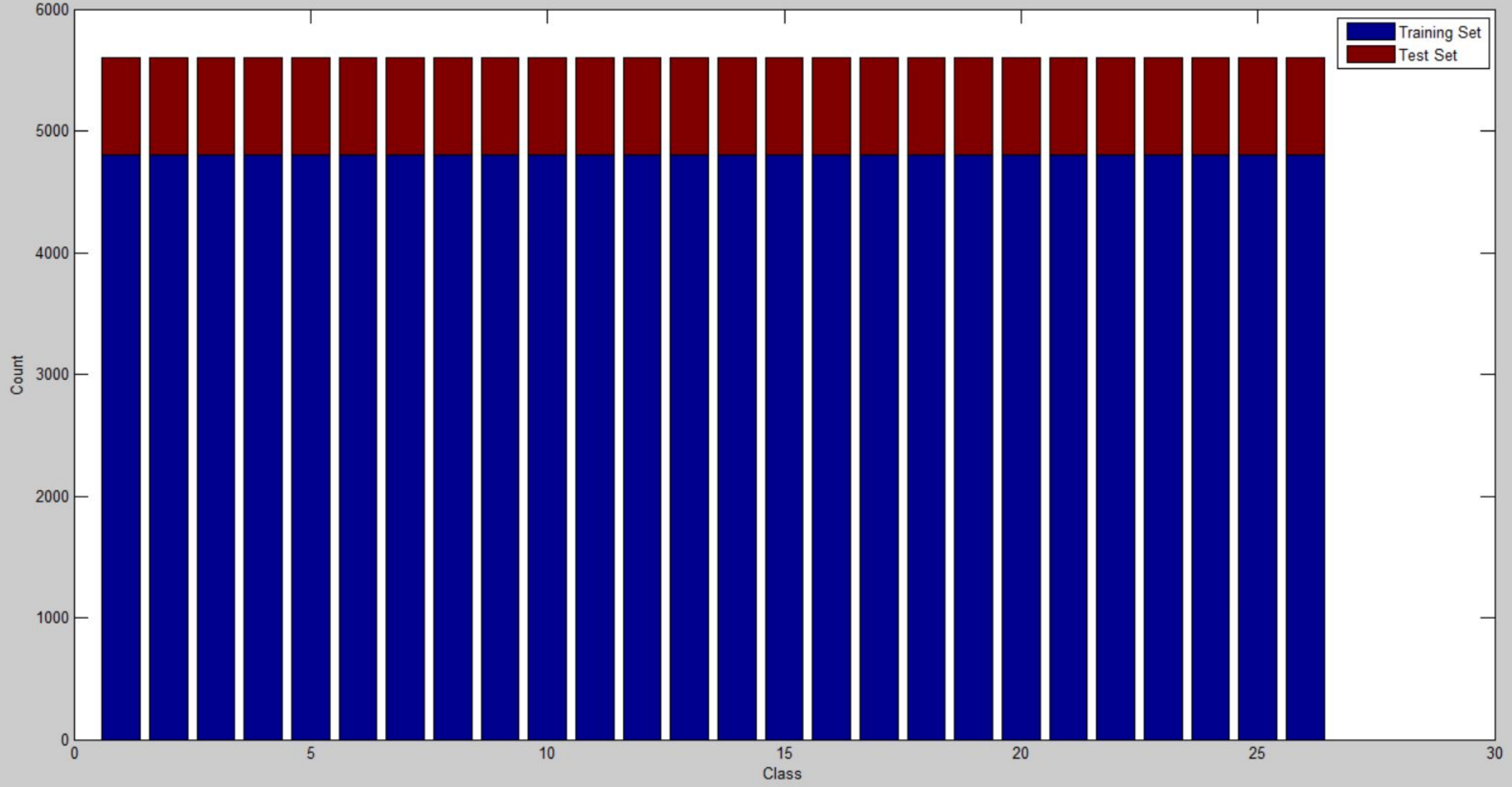
Dataset Summary

There are six different splits provided in this dataset. A short summary of the dataset is provided below:

- EMNIST ByClass: 814,255 characters. 62 unbalanced classes.
- EMNIST ByMerge: 814,255 characters. 47 unbalanced classes.
- EMNIST Balanced: 131,600 characters. 47 balanced classes.
- EMNIST Letters: 145,600 characters. 26 balanced classes.
- EMNIST Digits: 280,000 characters. 10 balanced classes.
- EMNIST MNIST: 70,000 characters. 10 balanced classes.

The full complement of the NIST Special Database 19 is available in the ByClass and ByMerge splits. The EMNIST Balanced dataset contains a set of characters with an equal number of samples per class. The EMNIST Letters dataset merges a balanced set of the uppercase and lowercase letters into a single 26-class task. The EMNIST Digits and EMNIST MNIST dataset provide balanced handwritten digit datasets directly compatible with the original MNIST dataset.

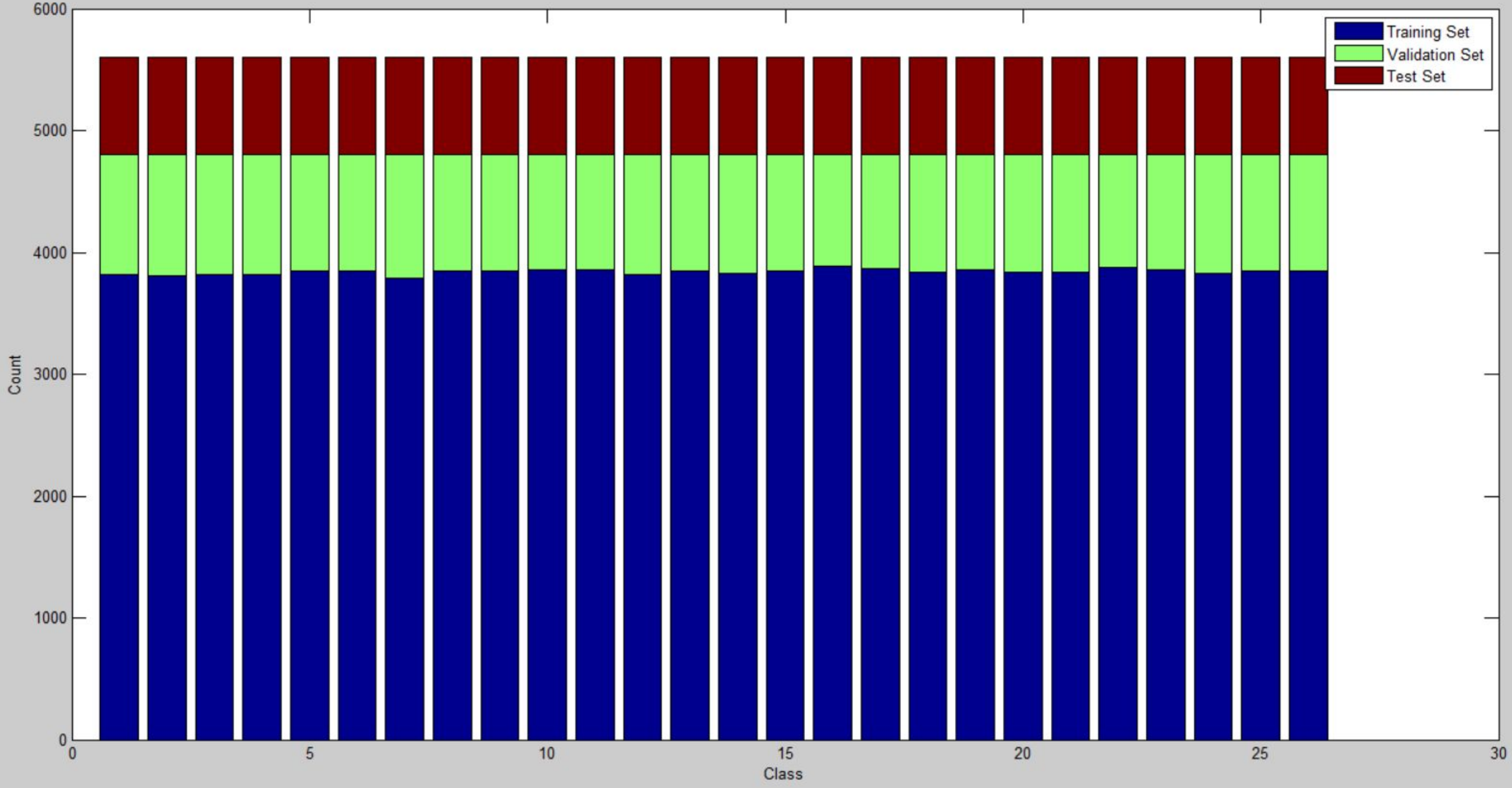
Please refer to the EMNIST paper [PDF, BIB]for further details of the dataset structure.


Training Set
Test Set
Count
Class
0
1000
2000
3000
4000
5000
6000
0
5
10
15
20
25
30

3.
Preparation

Data preparation

- Split data into training set and validation set:
  - Training set: 80%
  - Validation set: 20%
- Normalize data:
  - Scale each value to the range of [0, 1]


Training Set
Validation Set
Test Set
Count
Class
6000
5000
4000
3000
2000
1000
0
0
5
10
15
20
25
30

Label preparation

→ One-hot encoding the labels

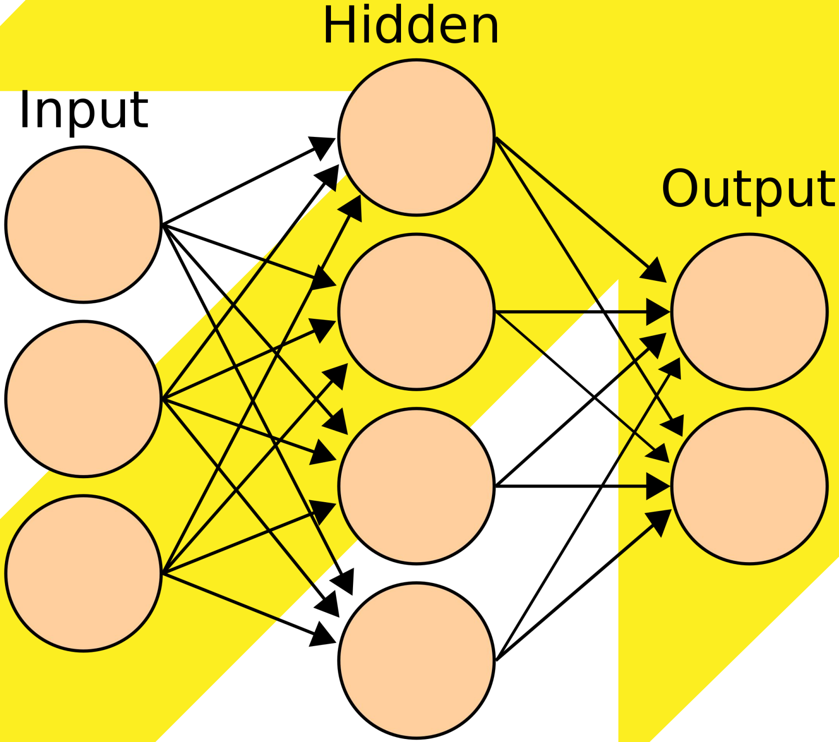
Label
1 (A)
2 (B)
…
26 (Z)

1 (A)	2 (B)	…	26 (Z)
1	0	…	0
0	1	…	0
…	…	…	…
0	0	…	1

4.
Progress

Our Approach

- Model:
  - Neural Network
- Evaluation Metrics:
  - Training Set Accuracy
  - Validation Set Accuracy
  - Test Set Accuracy

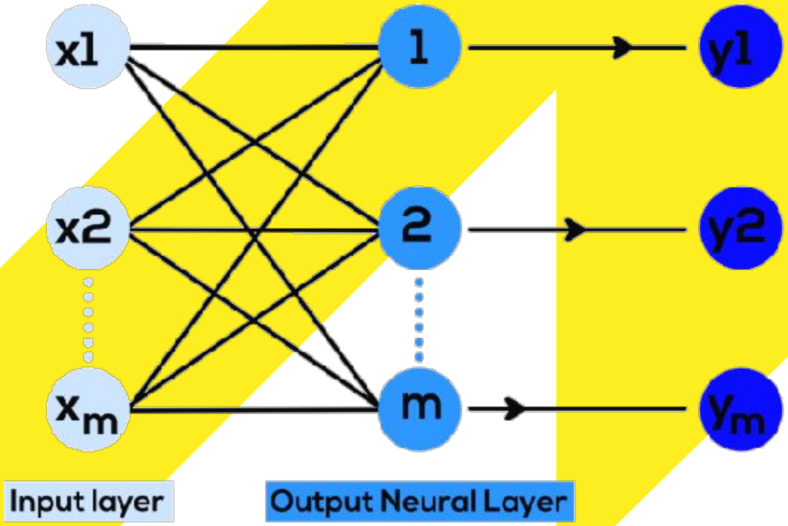


Model parameters

- We try to find the best combination of:
  - Activation function
  - Number of nodes
  - Number of hidden layers

Single Layer Perceptron

- Architecture:
  - 1 input layer (784 nodes)
  - 0 hidden layer
  - 1 output layer (26 nodes)



Weight initialization

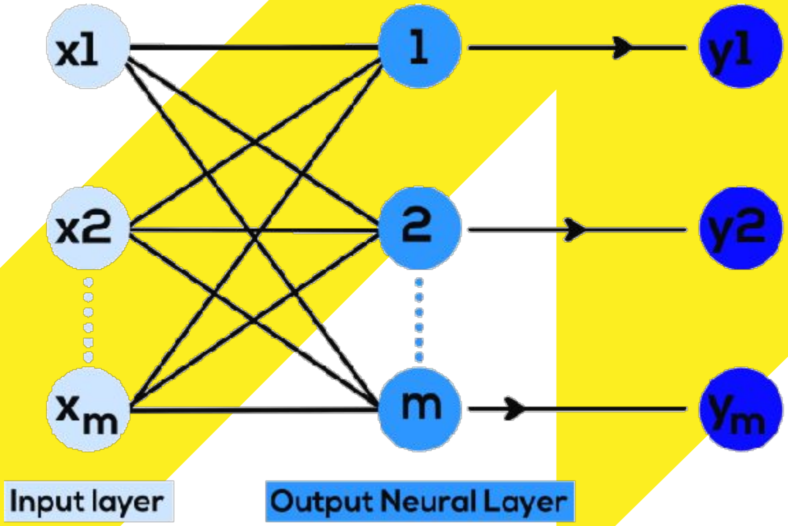
- **Random initialization:**
  - Initialize each weight to the same value will cause all nodes in neural network to learn the same feature while training => worsen model's performance
  - **Symmetry breaking:** initialize each weight randomly so each node will learn different features

Weight initialization

- **Xavier initialization:** Initialize each weight to a random value between [-**ϵ**, **ϵ**]:
  - $$\epsilon = \frac{\sqrt{6}}{\sqrt{L_{in} + L_{out}}}$$
  - $$\Theta^{(l)} = 2\epsilon * rand(L_{out}, L_{in} + 1) - \epsilon$$

Single Layer Perceptron

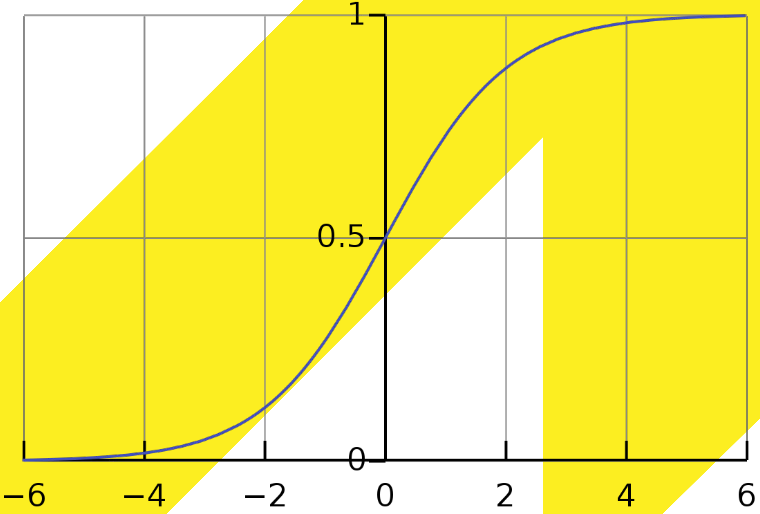
- Output layer:
  - Sigmoid + BCE Loss
  - Softmax + CCE Loss



Sigmoid + BCE Loss

- Formula:

$$\hat{y}_i = \frac{1}{1 + e^{-z}}$$



- Binary Cross Entropy Loss:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} [-y_k^{(i)} \log \hat{y}_k^{(i)} - (1 - y_k^{(i)}) \log(1 - \hat{y}_k^{(i)})]$$

Sigmoid + BCE Loss

- Derivative:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)}$$

Softmax + CCE Loss

→ Formula:

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

→ Categorical Cross Entropy Loss:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log \hat{y}_k^{(i)}$$

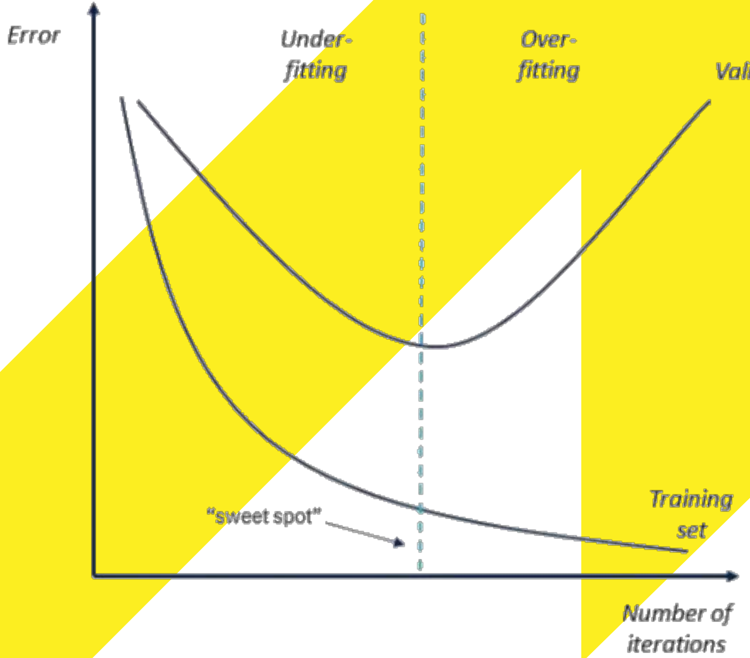
Softmax + CCE Loss

→ Derivative:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})x_j^{(i)}$$

Let’s train

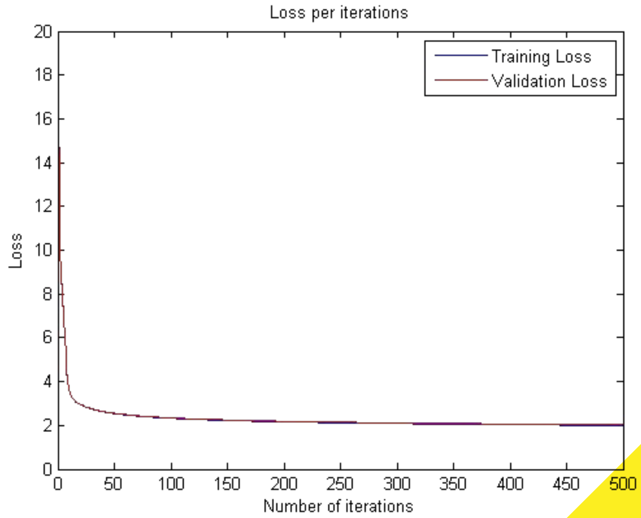
- Learning rate: **0.5**
- Number of iterations: **500**
- Save the weights that minimize validation loss
- **Early stopping technique**: stop training if validation loss does not decrease



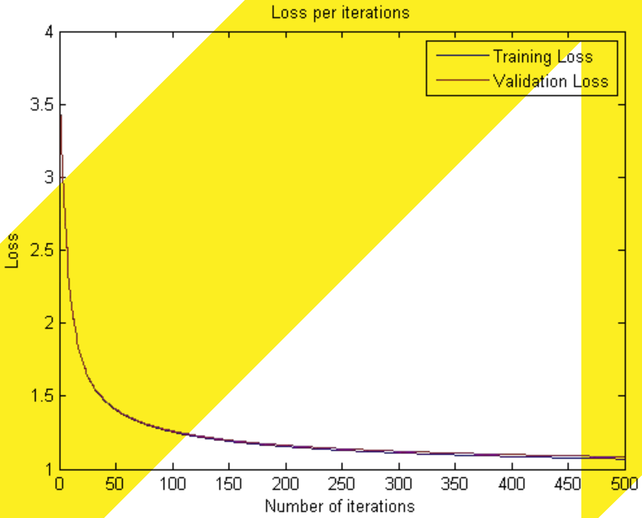
Single Layer Perceptron

Model	Total Parameters	Training Set	Validation Set	Test Set
Perceptron_sigmoid	**20,410**	**68.99 %**	**68.15 %**	**68.66 %**
Perceptron_softmax	**20,410**	**70.15 %**	**69.76 %**	**69.88 %**

Single Layer Perceptron



Perceptron_sigmoid



Perceptron_softmax

Perceptron_softmax

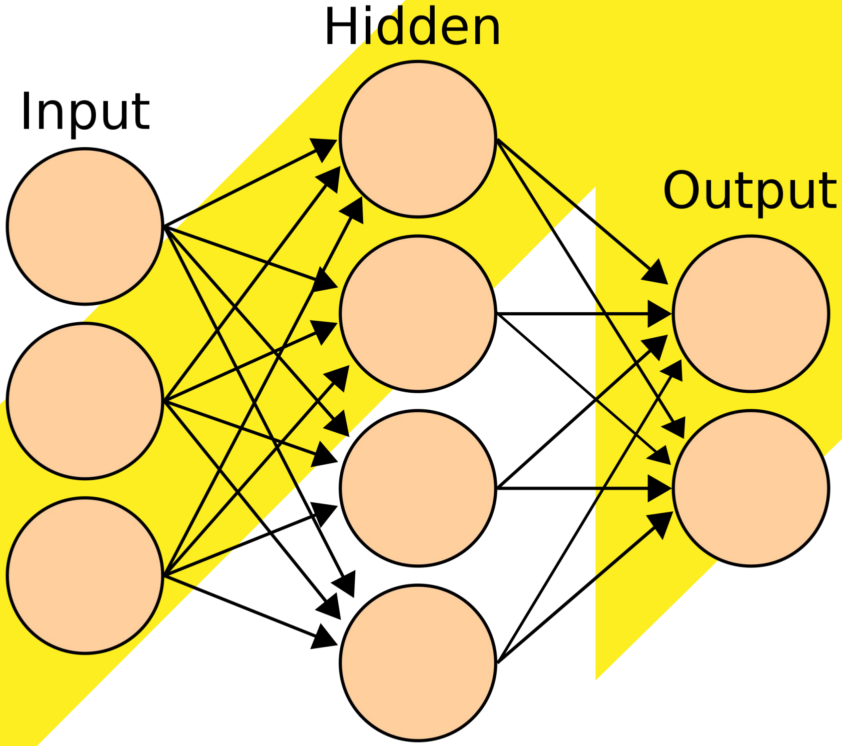
Confusion Matrix

True label	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Z	6	19	5	13	10	2	2	3	6	26	9	14	4	14	0	2	10	4	3	10	7	0	0	21	2	608
Y	1	4	0	3	1	4	16	7	20	23	5	18	5	5	0	7	5	5	5	40	3	46	2	28	544	3
X	17	0	0	4	1	5	5	6	19	1	44	14	4	4	0	2	1	10	3	7	2	35	4	566	41	5
W	5	6	1	13	0	2	0	6	0	0	6	1	6	19	0	0	1	2	0	5	26	6	693	2	0	0
V	1	2	0	5	1	1	0	2	2	0	17	11	4	16	0	4	1	27	0	2	45	593	25	13	27	1
U	34	6	3	25	2	0	3	15	0	5	11	7	0	5	3	0	1	2	1	0	603	41	30	0	0	3
T	2	16	5	5	22	67	7	32	22	5	10	11	3	1	3	18	18	56	2	423	0	3	0	4	62	3
S	10	7	2	1	2	1	25	2	3	47	4	7	2	3	6	0	14	6	638	3	2	1	2	4	1	7
R	34	3	18	0	15	30	1	6	3	0	38	3	6	11	1	14	18	507	0	38	0	5	7	13	23	6
Q	44	7	8	13	7	23	68	1	5	8	8	7	0	2	7	16	483	2	8	43	12	0	2	5	16	5
P	3	0	0	3	3	11	4	2	1	0	0	0	1	6	1	677	7	26	0	33	1	0	1	2	18	0
O	8	2	10	15	0	2	14	1	0	1	0	0	4	10	705	3	14	3	2	3	2	0	1	0	0	0
N	41	4	0	10	3	0	2	27	0	1	31	0	58	499	2	6	6	9	0	2	11	27	53	4	3	1
M	7	0	0	1	0	0	0	6	0	0	1	0	716	43	1	2	5	0	0	2	3	3	4	1	4	1
L	0	12	20	3	0	4	1	2	217	4	21	389	0	2	0	3	1	1	0	19	23	13	0	27	31	7
K	8	10	20	2	11	3	1	47	10	2	527	21	6	9	0	5	2	32	1	11	6	14	4	36	8	4
J	1	9	2	55	1	7	13	2	18	581	0	20	0	0	0	3	8	5	23	13	5	3	0	5	14	12
I	1	3	0	1	1	9	0	2	511	24	4	151	0	0	0	1	7	4	9	8	2	6	0	35	7	14
H	21	11	2	21	3	3	3	520	16	4	51	24	15	60	0	2	0	3	0	1	12	2	13	8	1	4
G	29	27	28	6	8	12	381	7	0	41	2	4	3	5	2	10	135	3	43	19	6	1	4	2	14	8
F	4	3	4	0	5	588	3	2	8	10	2	8	1	3	0	59	13	25	0	39	0	7	0	5	8	3
E	20	18	56	1	593	9	6	1	0	1	16	3	1	8	7	3	7	23	7	7	3	0	1	3	1	5
D	14	55	3	484	1	1	8	28	3	16	8	26	2	5	71	11	12	0	3	0	16	8	7	2	1	15
C	6	2	635	4	43	1	15	0	3	2	8	6	0	4	16	7	5	15	1	5	13	1	1	0	1	6
B	6	599	0	30	12	2	31	26	4	10	6	14	3	4	9	5	5	1	3	3	6	0	1	2	3	15
A	471	5	4	27	20	13	28	38	1	2	10	6	14	38	22	11	27	5	1	5	16	2	7	8	4	15

Predicted label

Neural Network

- Architecture:
  - 1 input layer (784 nodes)
  - 1 hidden layer
  - 1 output layer (26 nodes)



Neural Network

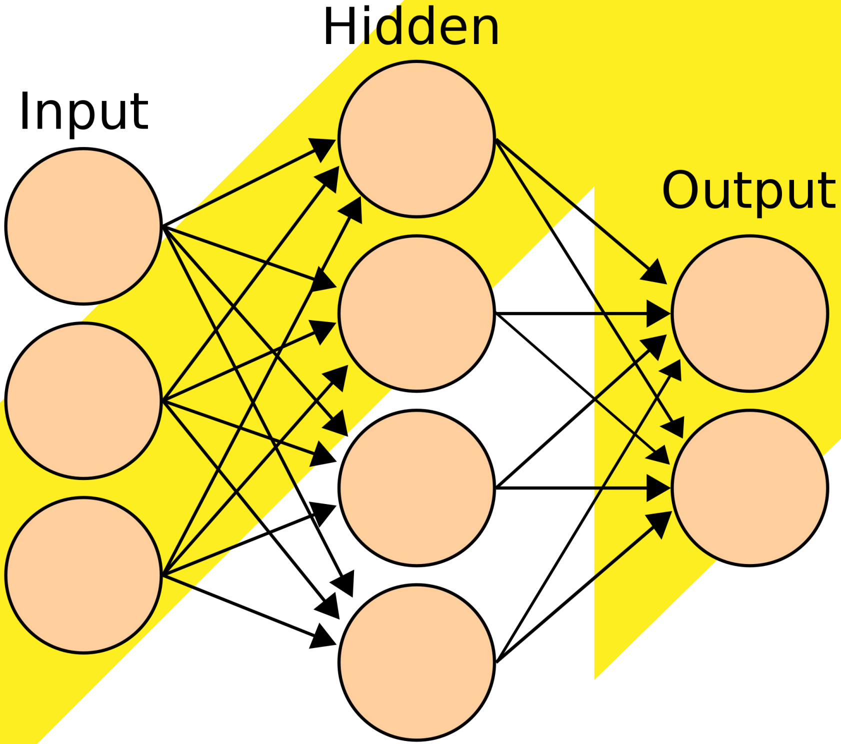
- Forward propagation:
  - $a^{(1)} = x$
  - $z^{(l+1)} = a^{(l)} * \Theta^{(l)} \qquad (l : 1 \rightarrow L)$
  - $a^{(l+1)} = g(z^{(l+1)}) \qquad (l : 1 \rightarrow L)$
  - $J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \log \hat{y}_k^{(i)} \qquad (\hat{y} = a^{(L+1)})$

Neural Network

- Backward propagation:
  - $\delta^{(L+1)} = \dfrac{\partial J}{\partial z^{(L+1)}} = a^{(L+1)} - y$
  - $\dfrac{\partial J}{\partial a^{(l)}} = \dfrac{\partial J}{\partial z^{(l+1)}} * \dfrac{\partial z^{(l+1)}}{\partial a^{(l)}} = \delta^{(l+1)} * \Theta^{(l)}$ $\quad (l : L \to 1)$
  - $\delta^{(l)} = \dfrac{\partial J}{\partial z^{(l)}} = \dfrac{\partial J}{\partial a^{(l)}} * \dfrac{\partial a^{(l)}}{\partial z^{(l)}} = (\delta^{(l+1)} * \Theta^{(l)}) \;.* \; g'(z^{(l)})$ $\quad (l : L \to 1)$
  - $\dfrac{\partial J}{\partial \Theta^{(l)}} = \dfrac{\partial J}{\partial z^{(l+1)}} * \dfrac{\partial z^{(l+1)}}{\partial \Theta^{(l)}} = \delta^{(l+1)} * a^{(l)}$ $\quad (l : L \to 1)$
  - $\Delta^{(l)} = \dfrac{1}{m} * \dfrac{\partial J}{\partial \Theta^{(l)}}$

Neural Network

- Hidden layer:
  - Sigmoid
  - Tanh
  - ReLU
  - Leaky ReLU
  - Swish
  - Mish



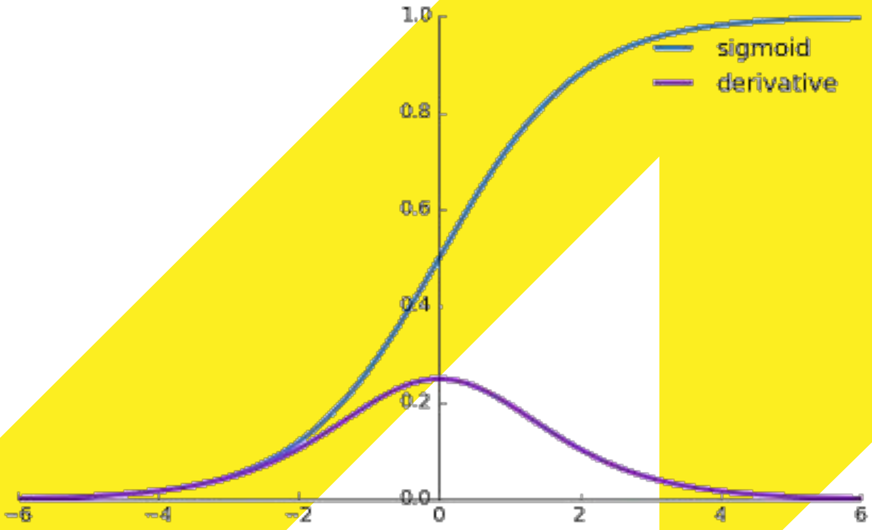
Sigmoid activation

→ Formula:

$$g(z) = \frac{1}{1 + e^{-z}}$$

→ Derivative:

$$g'(z) = g(z)(1 - g(z))$$



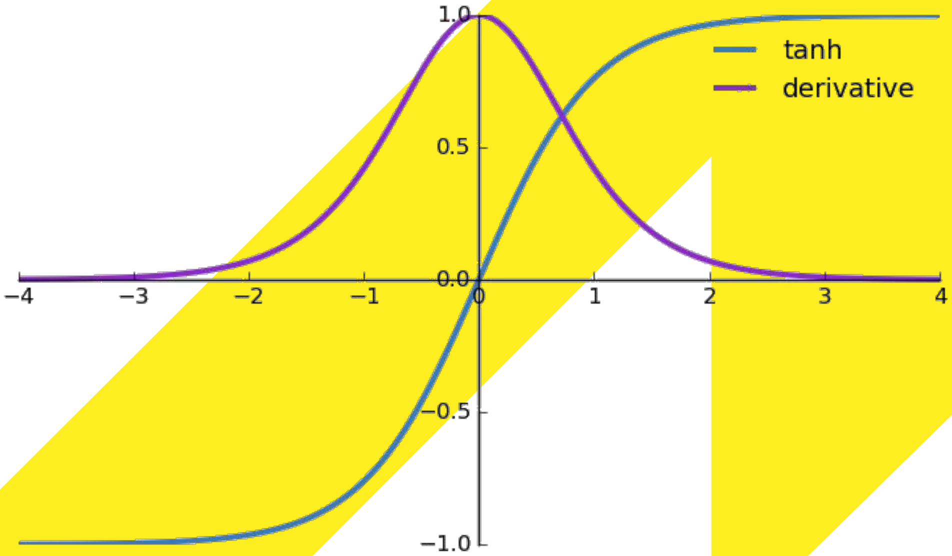
Tanh activation

➔ Formula:

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

➔ Derivative:

$$g'(z) = 1 - g(z)^2$$



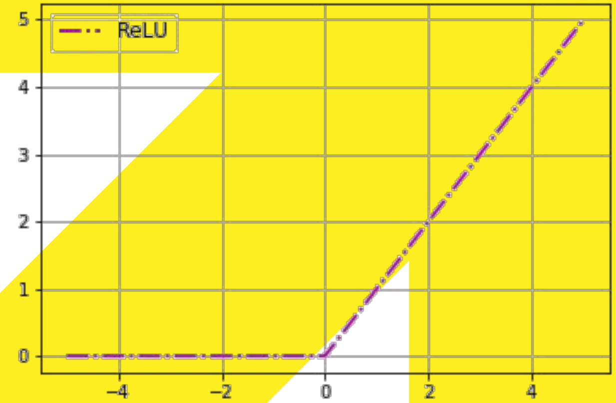
ReLU activation

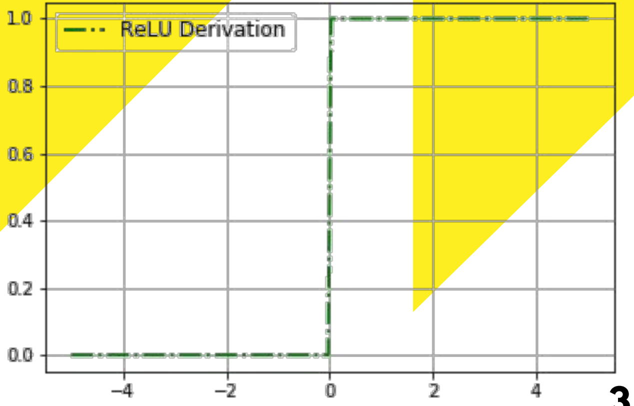
- → Formula:

$$g(z) = \begin{cases} 0 & \text{for} \quad z < 0 \\ z & \text{for} \quad z \geq 0 \end{cases}$$

- → Derivative:

$$g'(z) = \begin{cases} 0 & \text{for} \quad z < 0 \\ 1 & \text{for} \quad z \geq 0 \end{cases}$$





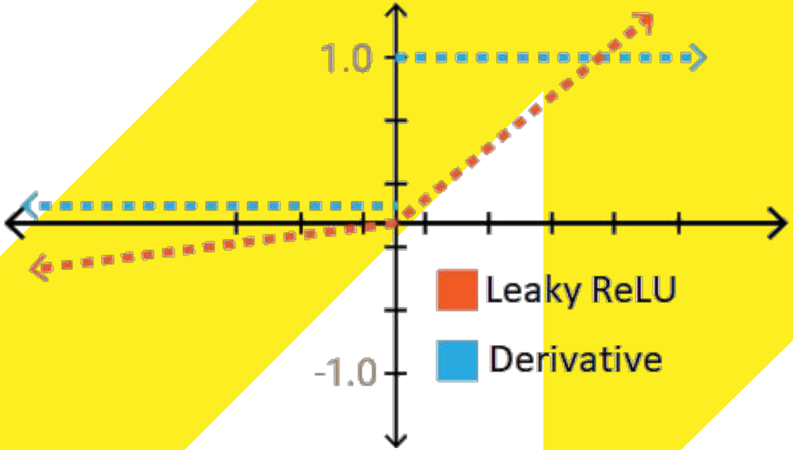
Leaky ReLU activation

➔ Formula:

$$g(z) = \begin{cases} \alpha z & \text{for} \quad z < 0 \\ z & \text{for} \quad z \geq 0 \end{cases}$$

➔ Derivative:

$$g'(z) = \begin{cases} \alpha & \text{for} \quad z < 0 \\ 1 & \text{for} \quad z \geq 0 \end{cases}$$



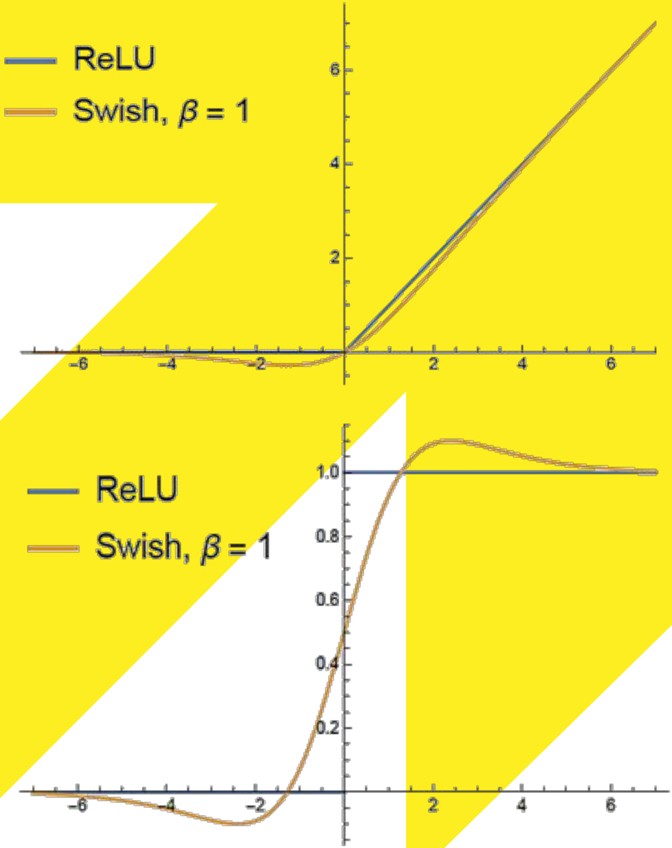
Swish activation

➔ Formula:

$$g(z) = z * sigmoid(z)$$

➔ Derivative:

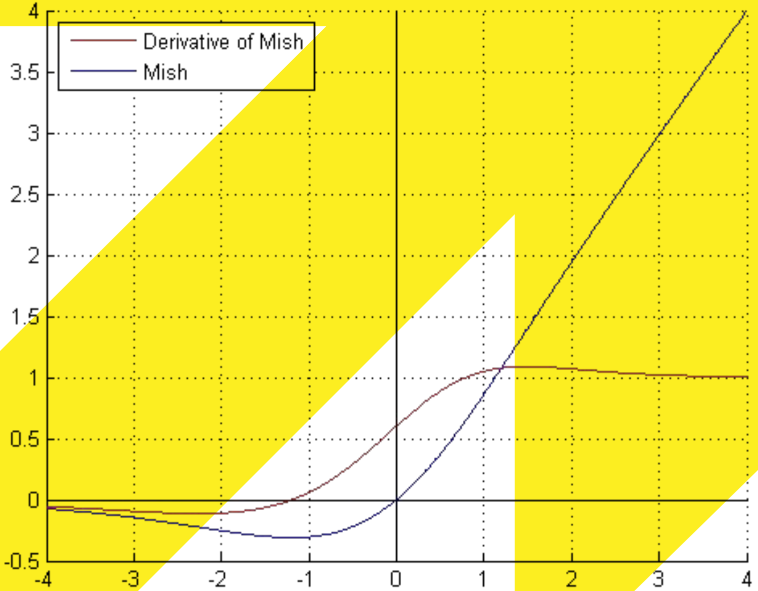
$$g'(z) = g(z) + sigmoid(z) * (1 - g(z))$$



Mish activation

➔ Formula:

$$g(z) = z * \tanh(\ln(1 + e^z))$$

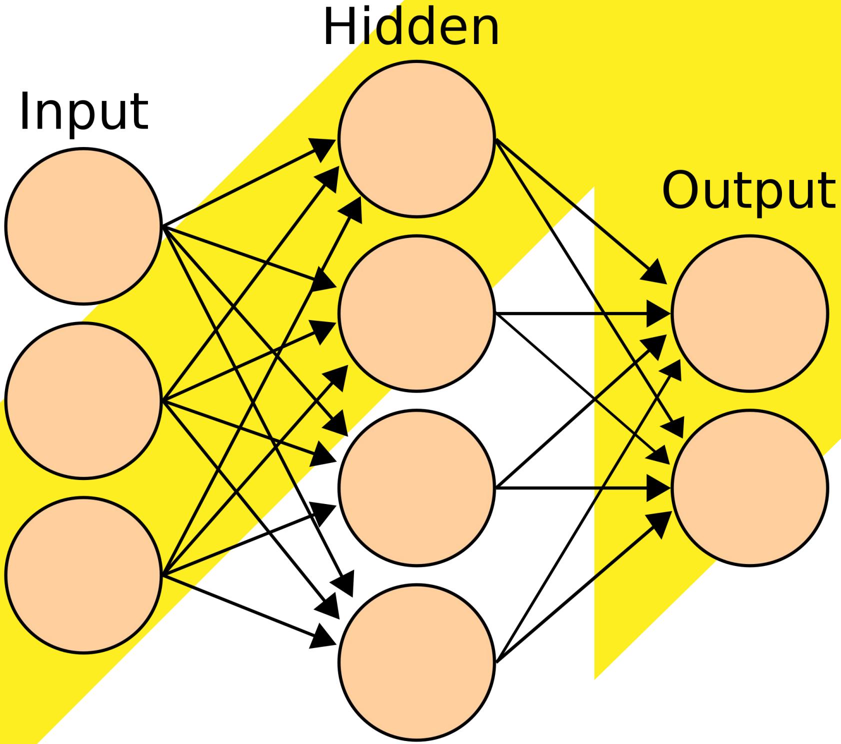


➔ Derivative:

$$g'(z) = \frac{e^z \omega}{\delta^2} = \frac{e^z(4e^{2z} + e^{3z} + 4(1 + z) + e^z(6 + 4z))}{(2 + 2e^z + e^{2z})^2}$$

Neural Network

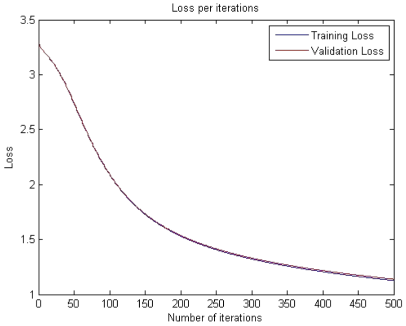
- Hidden layer:
  - 64 nodes
  - 128 nodes
  - 256 nodes
  - 512 nodes



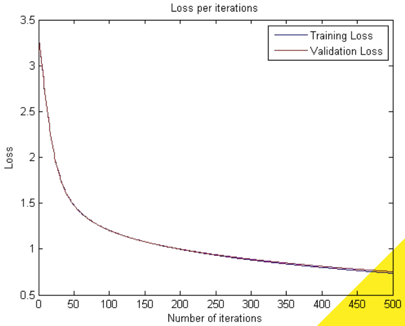
Neural Network

Model	Total Parameters	Training Set	Validation Set	Test Set
NN_sigmoid_softmax_64	**51,930**	**67.95 %**	**68.27 %**	**67.81 %**
NN_tanh_softmax_64	**51,930**	**79.13 %**	**78.97 %**	**78.53 %**
NN_relu_softmax_64	**51,930**	**82.12 %**	**81.98 %**	**81.74 %**
NN_leakyrelu_softmax_64	**51,930**	**81.64 %**	**81.91 %**	**80.96 %**
NN_swish_softmax_64	**51,930**	**81.77 %**	**80.86 %**	**80.92 %**
NN_mish_softmax_64	**51,930**	**82.31 %**	**81.85 %**	**81.75 %**

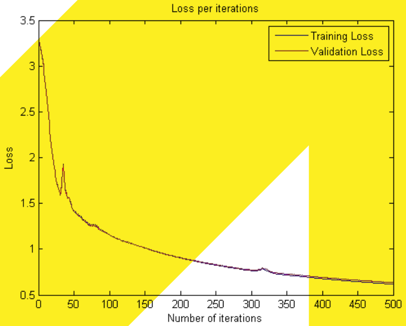
Neural Network



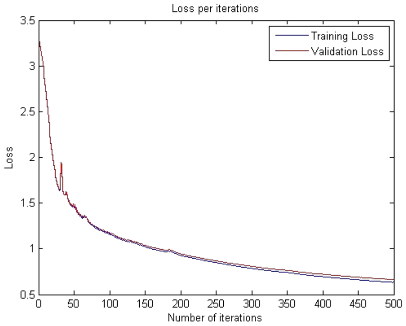
NN_sigmoid_softmax_64



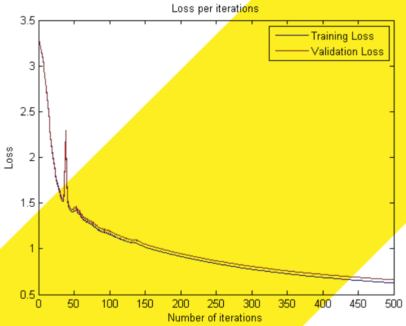
NN_tanh_softmax_64



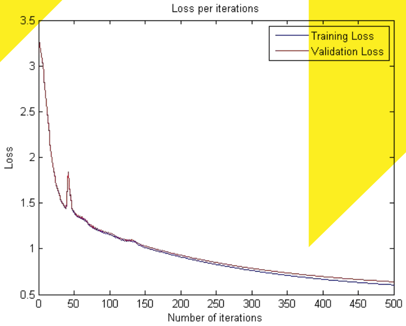
NN_relu_softmax_64



NN_leakyrelu_softmax_64



NN_swish_softmax_64



NN_mish_softmax_64

Neural Network

Model	Total Parameters	Training Set	Validation Set	Test Set
NN_relu_softmax_128	**103,834**	**84.06 %**	**83.37 %**	**83.19 %**
NN_leakyrelu_softmax_128	**103,834**	**84.14 %**	**82.67 %**	**82.95 %**
NN_swish_softmax_128	**103,834**	**83.71 %**	**82.84 %**	**83.02 %**
NN_mish_softmax_128	**103,834**	**83.78 %**	**83.05 %**	**83.14 %**

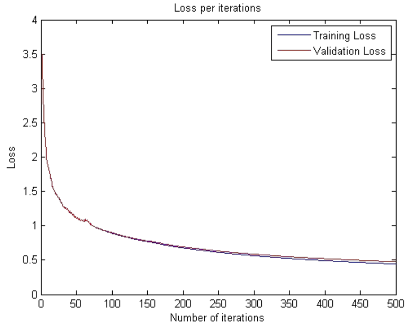
Neural Network

Model	Total Parameters	Training Set	Validation Set	Test Set
NN_relu_softmax_256	**207,642**	**85.57 %**	**85.18 %**	**84.60 %**
NN_leakyrelu_softmax_256	**207,642**	**85.37 %**	**84.70 %**	**84.15 %**
NN_swish_softmax_256	**207,642**	**84.77 %**	**83.85 %**	**83.55 %**
NN_mish_softmax_256	**207,642**	**85.16 %**	**84.43 %**	**84.30 %**

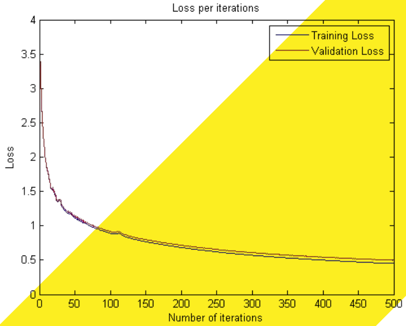
Neural Network

Model	Total Parameters	Training Set	Validation Set	Test Set
NN_relu_softmax_512	**415,258**	**87.03 %**	**86.02 %**	**85.51 %**
NN_leakyrelu_softmax_512	**415,258**	**86.79 %**	**85.00 %**	**85.30 %**
NN_swish_softmax_512	**415,258**	**85.20 %**	**83.89 %**	**84.13 %**
NN_mish_softmax_512	**415,258**	**86.21 %**	**85.65 %**	**85.01 %**

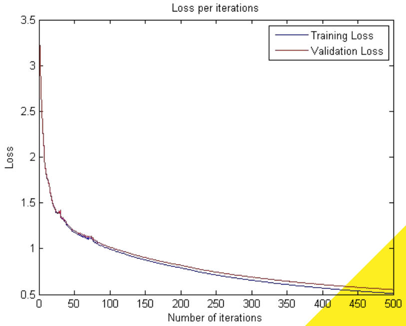
Neural Network



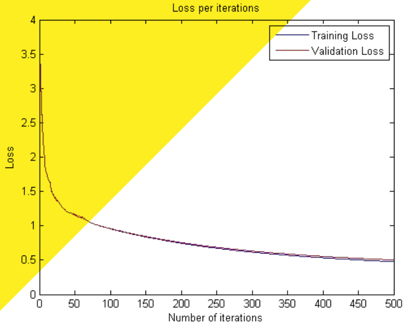
NN_relu_softmax_512



NN_leakyrelu_softmax_512



NN_swish_softmax_512



NN_mish_softmax_512

NN_relu_softmax_512

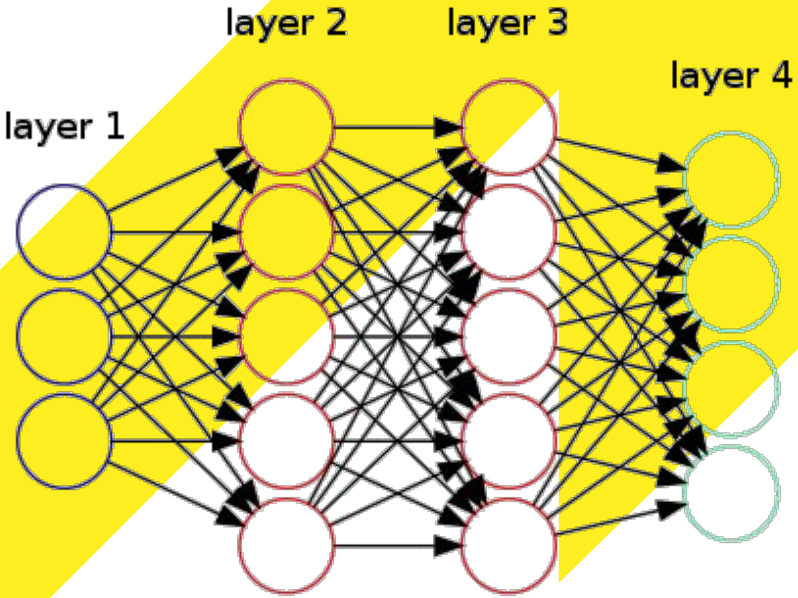
Confusion Matrix

True label	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Z	6	0	3	2	13	1	6	2	4	3	2	6	1	2	0	2	3	1	1	2	0	0	0	9	0	731
Y	1	5	0	2	0	2	15	1	1	14	3	5	2	1	0	1	3	3	2	12	1	23	2	10	691	0
X	8	0	0	3	0	2	1	3	3	1	27	2	1	2	0	1	3	6	0	3	0	8	0	703	21	2
W	0	0	0	4	0	1	2	2	0	1	4	0	3	19	0	0	0	0	0	2	15	5	740	2	0	0
V	0	0	0	2	0	0	0	1	1	1	5	2	0	6	0	1	0	9	0	2	32	705	4	3	25	1
U	6	1	2	9	0	0	2	4	0	3	3	2	1	3	4	0	1	0	1	0	717	31	8	0	2	0
T	2	3	2	2	7	7	2	3	4	4	5	6	1	1	1	0	0	7	3	725	0	0	0	3	12	0
S	5	2	1	0	2	0	23	1	5	18	1	0	1	1	1	0	2	2	734	0	1	0	0	0	0	0
R	24	3	4	1	10	18	1	1	3	0	18	2	2	3	0	10	2	649	1	17	0	2	6	8	10	5
Q	40	4	4	7	5	11	133	1	3	2	2	4	0	2	12	7	539	4	3	5	3	0	1	0	6	2
P	2	0	0	4	1	12	2	0	0	0	0	0	0	3	0	754	4	5	0	4	0	1	0	0	8	0
O	3	1	4	11	3	1	3	0	0	0	0	0	2	3	760	1	2	2	1	0	3	0	0	0	0	0
N	24	1	0	8	0	0	1	23	0	0	12	0	25	666	1	3	2	6	0	3	4	6	9	5	1	0
M	2	0	0	0	0	0	0	4	0	0	2	0	753	12	0	2	2	1	0	8	2	0	5	3	4	0
L	0	1	6	1	0	2	0	4	94	2	1	676	0	1	0	0	2	2	0	1	1	0	0	0	3	3
K	2	6	5	2	1	2	0	31	2	1	674	6	4	3	0	0	0	16	1	6	4	6	3	20	5	0
J	1	1	0	15	1	2	6	1	25	698	0	9	0	0	0	1	2	0	8	19	0	2	0	1	6	2
I	1	2	1	2	1	4	2	0	469	19	2	271	0	0	0	0	0	5	3	3	0	1	1	2	0	11
H	9	11	0	6	0	5	1	669	2	0	24	13	12	24	1	0	0	3	0	0	9	0	3	4	4	0
G	27	20	6	1	6	6	591	0	0	11	1	3	1	1	4	4	90	0	12	1	2	0	2	0	11	0
F	1	1	1	3	3	700	4	0	2	1	0	5	1	0	0	34	3	11	2	24	0	1	0	1	1	1
E	11	3	38	0	696	2	6	0	1	0	2	1	2	0	3	4	3	14	4	6	1	0	0	0	0	3
D	17	20	3	656	0	1	1	6	2	9	4	5	0	3	42	8	3	1	5	1	5	1	2	0	2	3
C	6	1	725	2	22	1	4	0	2	0	3	5	0	0	10	2	2	7	0	2	2	0	1	1	0	2
B	8	706	0	7	5	0	11	27	2	3	3	5	2	2	4	3	2	1	1	0	1	0	0	0	0	7
A	660	3	4	12	9	2	15	19	0	1	1	0	8	8	9	4	17	5	2	3	9	0	1	1	0	7

Predicted label

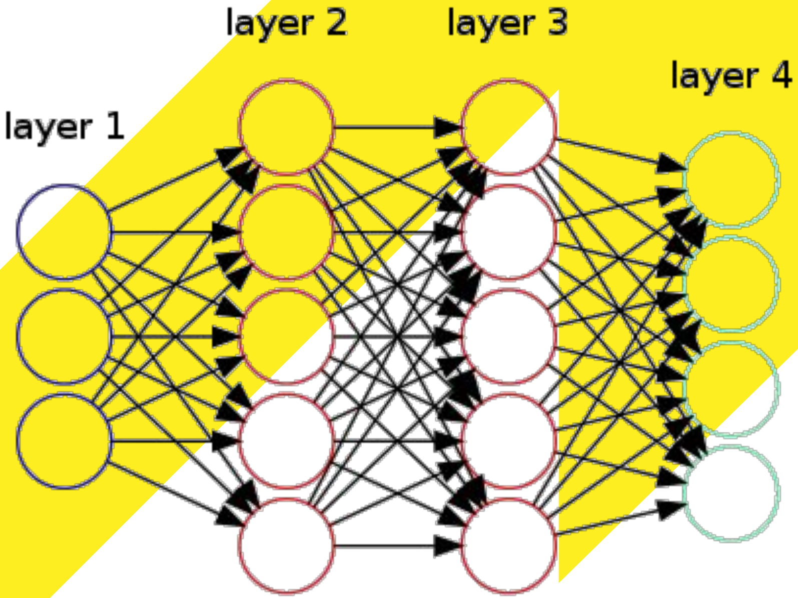
Deeper Neural Network

- Architecture:
  - 1 input layer (784 nodes)
  - 2 hidden layers
  - 1 output layer (26 nodes)



Deeper Neural Network

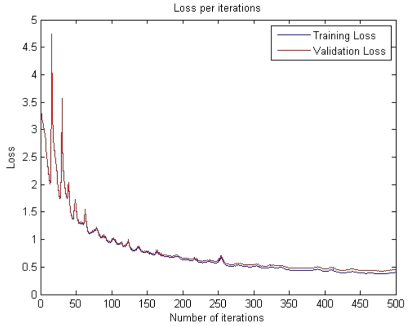
- Hidden layer 1:
  - 512 nodes
- Hidden layer 2:
  - 64 nodes
  - 128 nodes
  - 256 nodes



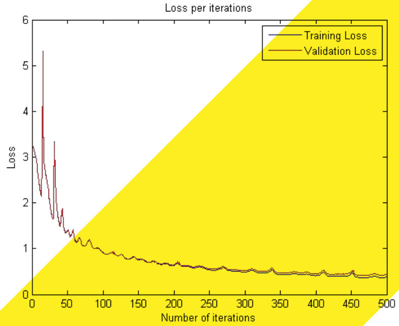
Deeper Neural Network

Model	Total Parameters	Training Set	Validation Set	Test Set
NN_relu_softmax_512_64	**436,442**	**88.46 %**	**86.85 %**	**86.71 %**
NN_leakyrelu_softmax_512_64	**436,442**	**88.84 %**	**87.31 %**	**87.28 %**
NN_swish_softmax_512_64	**436,442**	**87.58 %**	**86.68 %**	**86.42 %**
NN_mish_softmax_512_64	**436,442**	**87.94 %**	**87.17 %**	**87.01 %**

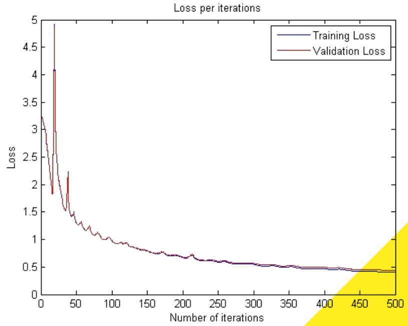
Deeper Neural Network



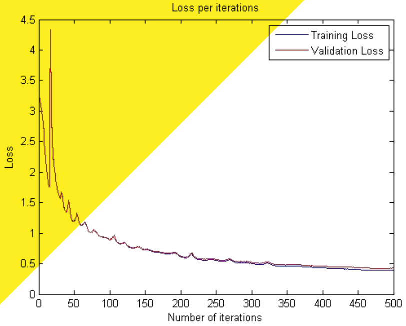
NN_relu_softmax_512_64



NN_leakyrelu_softmax_512_64



NN_swish_softmax_512_64



NN_mish_softmax_512_64

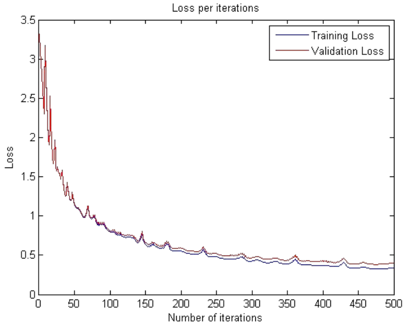
Deeper Neural Network

Model	Total Parameters	Training Set	Validation Set	Test Set
NN_relu_softmax_512_128	**470,938**	**89.40 %**	**88.05 %**	**87.56 %**
NN_leakyrelu_softmax_512_128	**470,938**	**89.25 %**	**87.36 %**	**87.65 %**
NN_swish_softmax_512_128	**470,938**	**87.86 %**	**87.02 %**	**86.81 %**
NN_mish_softmax_512_128	**470,938**	**88.34 %**	**87.37 %**	**87.24 %**

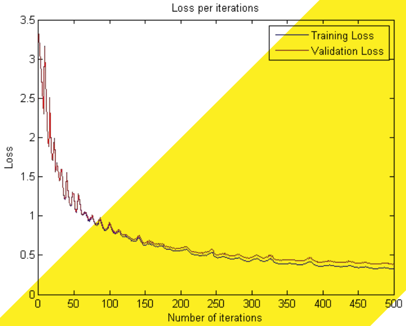
Deeper Neural Network

Model	Total Parameters	Training Set	Validation Set	Test Set
NN_relu_softmax_512_256	**539,930**	**89.98 %**	**88.10 %**	**88.29 %**
NN_leakyrelu_softmax_512_256	**539,930**	**89.99 %**	**88.34 %**	**88.31 %**
NN_swish_softmax_512_256	**539,930**	**87.56 %**	**86.71 %**	**86.44 %**
NN_mish_softmax_512_256	**539,930**	**88.74 %**	**87.69 %**	**87.50 %**

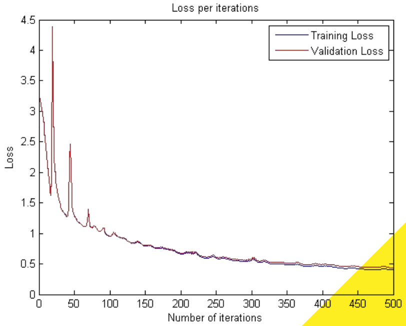
Deeper Neural Network



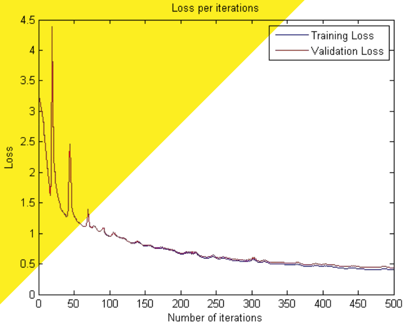
NN_relu_softmax_512_256



NN_leakyrelu_softmax_512_256



NN_swish_softmax_512_256



NN_mish_softmax_512_256

NN_leakyrelu_softmax_512_256

Confusion Matrix

True label \ Predicted label	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Z	2	2	3	2	12	0	7	0	5	1	2	3	1	1	0	0	3	3	1	0	1	0	0	7	1	743
Y	0	4	0	1	0	0	12	2	0	3	1	2	1	0	0	1	4	6	3	4	0	17	2	12	725	0
X	3	0	0	2	0	1	3	0	0	0	29	0	0	2	0	1	1	5	0	0	0	5	0	728	18	2
W	0	1	0	0	0	0	1	1	0	1	3	1	5	9	0	0	0	0	0	1	12	3	761	1	0	0
V	0	0	0	2	0	0	1	0	0	0	5	1	0	3	1	1	2	8	0	1	32	715	4	1	23	0
U	2	1	1	2	0	0	2	4	0	2	4	1	1	1	3	0	4	0	1	0	737	26	7	0	1	0
T	1	6	0	2	6	5	2	1	3	4	8	2	0	0	2	2	1	11	1	727	0	0	0	3	12	1
S	4	0	0	1	1	1	23	1	3	6	1	0	0	1	3	0	3	2	749	0	1	0	0	0	0	0
R	4	5	2	1	10	5	0	0	1	0	16	1	1	1	0	6	1	720	1	5	0	5	0	8	4	3
Q	22	4	3	1	8	6	116	0	2	1	0	0	0	1	5	6	609	2	2	3	2	0	0	0	7	0
P	0	0	0	5	1	6	5	0	0	0	1	1	1	1	0	761	6	8	0	1	0	1	0	0	2	0
O	2	3	2	9	0	0	5	0	0	0	0	0	0	1	768	1	3	2	0	0	4	0	0	0	0	0
N	13	2	0	5	0	1	1	17	0	0	12	0	29	693	1	0	1	5	0	1	3	2	9	5	0	0
M	0	0	0	0	0	0	0	4	0	0	2	0	768	10	0	2	0	2	0	3	2	0	3	2	2	0
L	0	4	5	0	1	3	0	3	92	1	3	673	0	0	1	2	1	3	0	1	1	0	0	1	4	1
K	1	6	3	0	1	4	1	13	0	0	726	5	2	1	0	0	0	9	1	3	1	1	2	18	2	0
J	1	5	0	11	1	0	7	0	24	706	0	9	0	0	0	0	2	0	6	12	1	2	0	1	7	5
I	0	2	1	0	1	1	1	0	471	16	3	276	0	0	0	0	0	7	3	2	0	1	1	4	0	10
H	14	17	0	3	0	2	4	685	0	0	29	8	7	16	0	0	0	3	0	0	6	0	3	1	2	0
G	11	18	3	1	4	3	637	0	0	5	0	1	0	1	3	0	98	1	5	0	0	0	2	0	7	0
F	1	1	0	2	4	714	6	0	1	1	0	2	1	0	0	36	0	10	2	15	0	0	0	1	2	1
E	6	4	15	0	736	0	5	0	1	0	1	1	2	0	2	3	3	13	2	3	2	0	0	1	0	0
D	11	22	1	689	0	1	3	2	1	5	1	0	0	2	41	8	1	1	1	1	3	0	2	1	3	0
C	2	3	728	3	30	0	3	0	1	0	2	4	0	0	8	2	3	6	0	2	2	0	1	0	0	0
B	4	735	0	4	2	0	9	19	0	1	1	2	2	1	4	1	4	2	1	0	1	0	0	1	0	6
A	665	6	4	11	9	2	21	11	0	1	7	0	5	4	7	5	26	3	1	0	3	0	2	1	0	6

True label

Predicted label

Even deeper NN, but wait...

OOP-style Python realization.

Naive implementation → exploding gradient, NaN loss, network learning nothing.

Solution:

- Stable softmax (max subtraction)
- Good initialization (Xavier/He)

Optimization technique

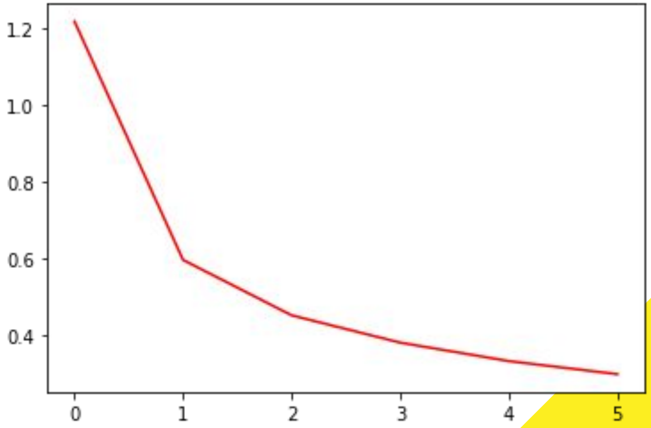
Gradient descent with momentum → faster convergence, avoid bad local minima.

Crucial: mini-batch GD. Much faster convergence.

For the rest of training process, we use lr=0.1, momentum=0.9, batch size 256.

New results

Every run easily reached 88-89% accuracy on test set and almost converges within **1-2 minutes** of training time.

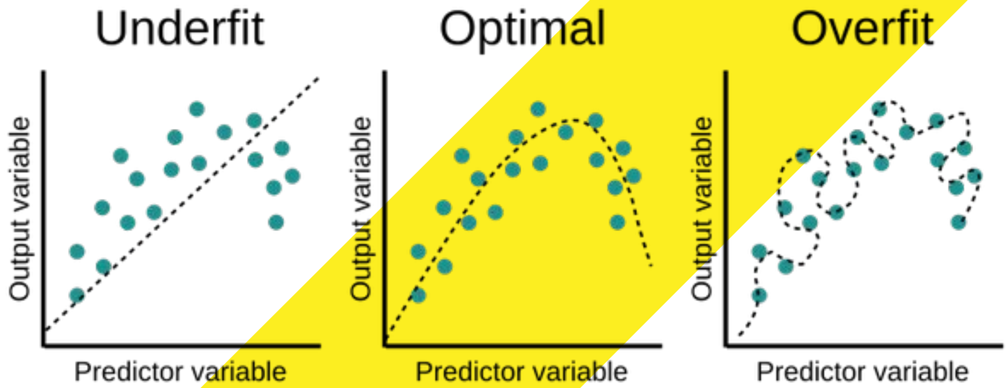


After 6 epochs with training time of 1m47s, in this run a 512-256-128 ReLU model achieved 0.299 training loss and 88.33% accuracy on test set.

About regularization

Simple regularization prevents overfitting, but:

- ➔ It is essentially limiting the model's complexity.
- ➔ Cherry-picking lambda can be hard.



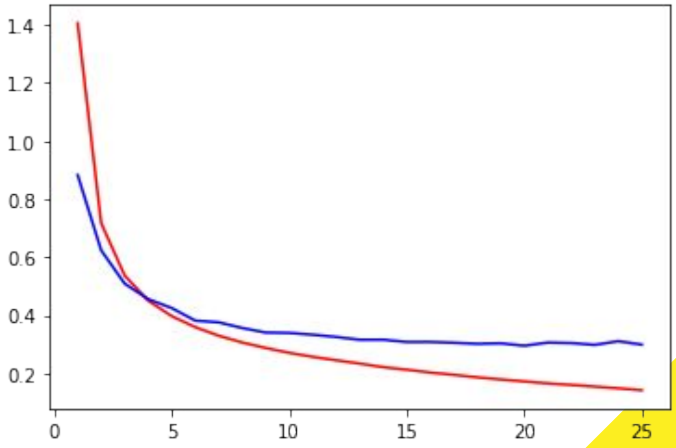
Data shuffling

Serving the data in a meaningful order or the same order every epoch might introduce some bias.

Thus the training data is shuffled every epoch.

New results

Every run reached ~90% accuracy on test set.

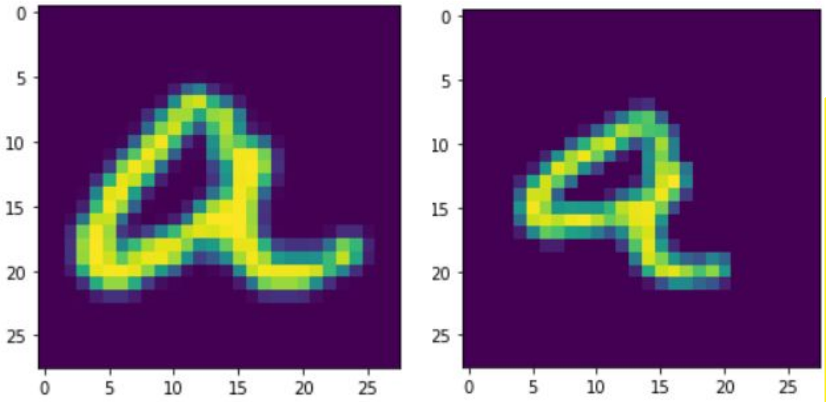


Loss graph of a run with 90.92% accuracy on test set (512-256-128 Swish). The run was automatically stopped after 26 epochs.

On data augmentation

Low image resolution + simple model (NN only) → Applying augmentation is **generally bad**.

However, small rotations might be useful.



20 degree right rotation of an ‘a’ letter. In training we applied on-the-fly 50% chance 10-15 degree rotation for each image.

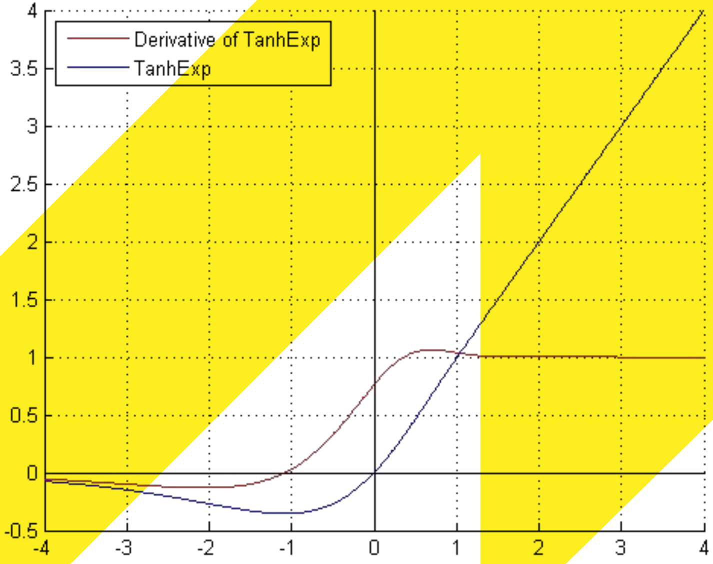
Tanh Exponential activation

➔ Formula:

$$g(z) = z * tanh(e^z)$$

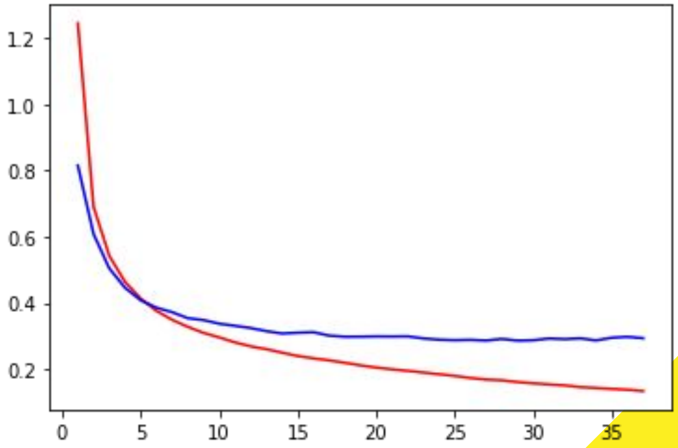
➔ Derivative:

$$g'(z) = \tanh(e^z) - ze^z(\tanh^2(e^z) - 1)$$



The result

We reached ~91% test accuracy on the most runs.

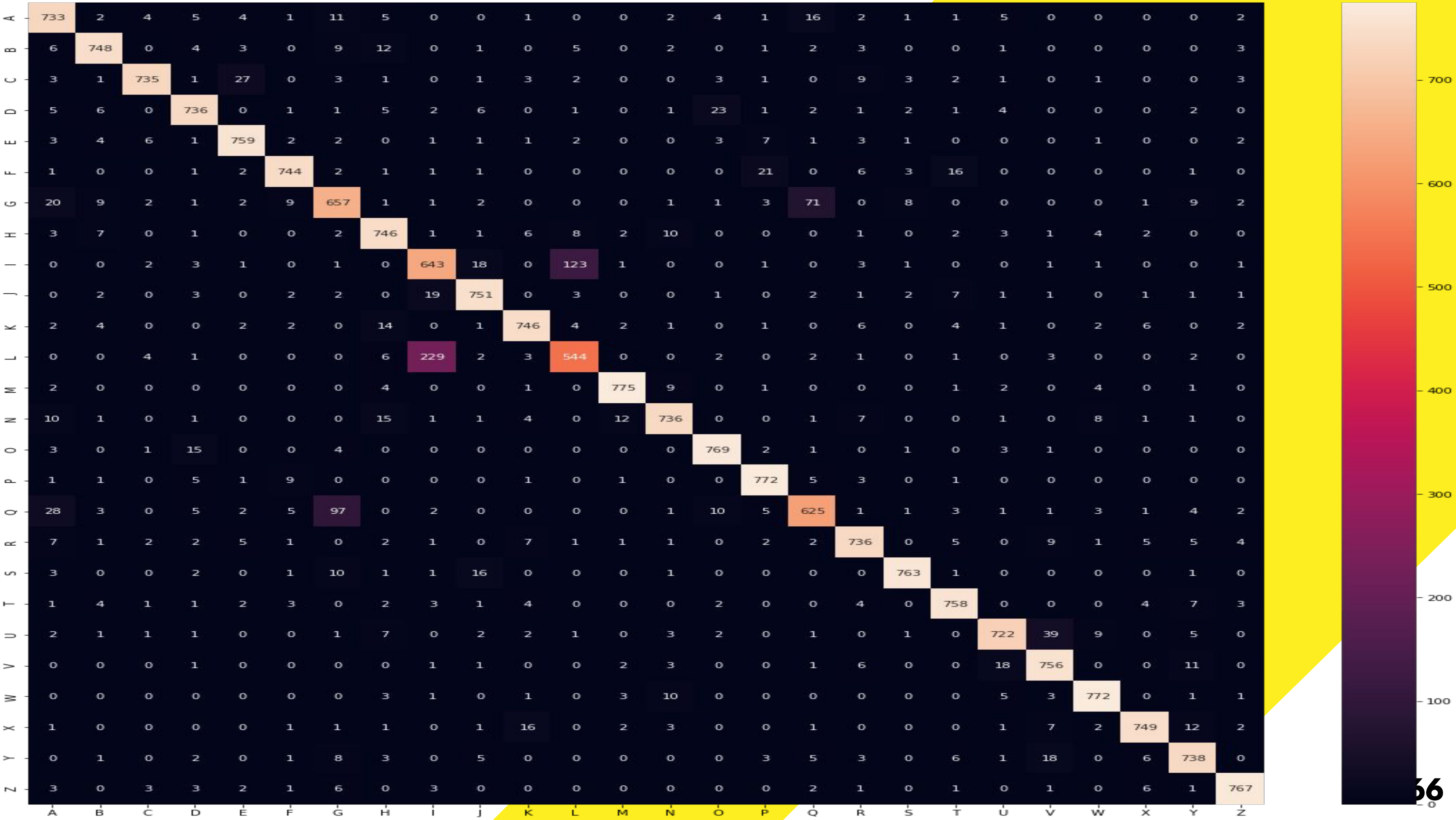


Loss graph of a run with 91.25% accuracy on test set (512-256-128 TanhExp). The run was automatically stopped after 38 epochs.

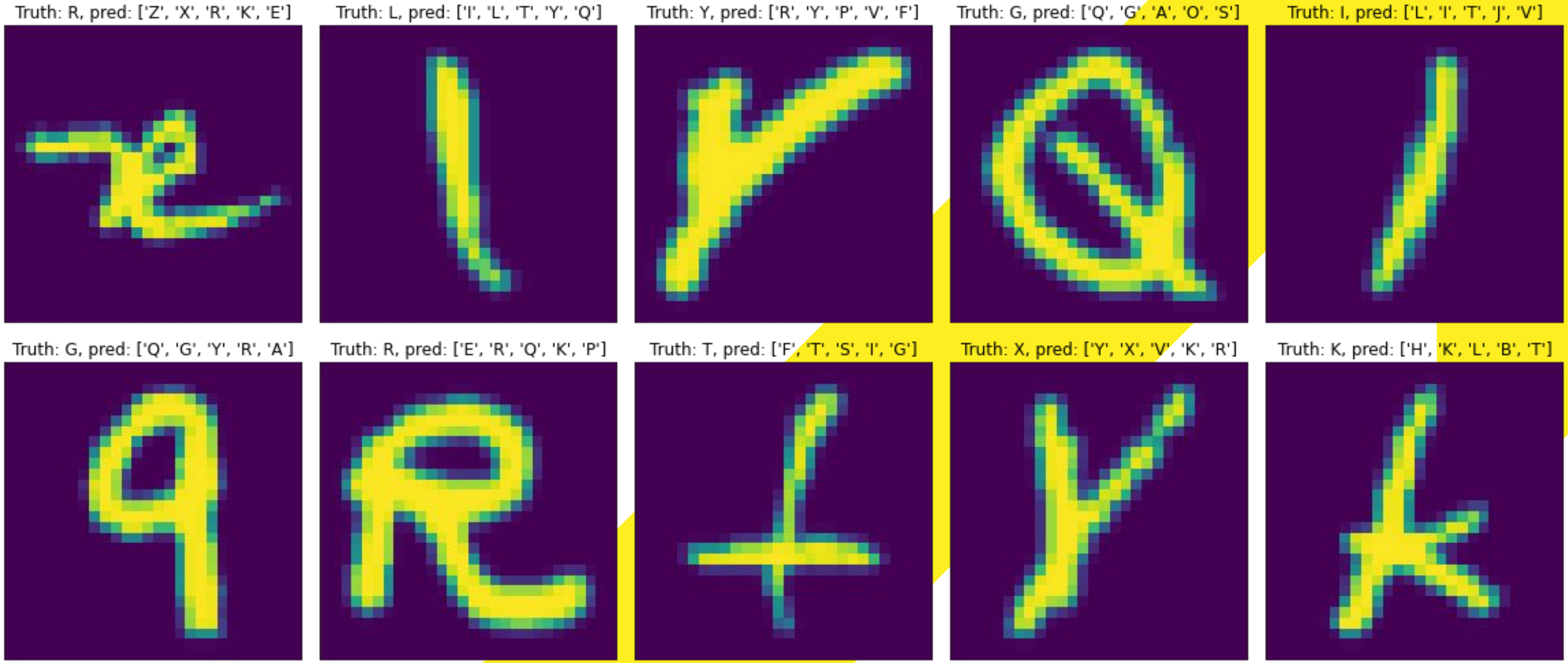
We tried, didn't work

- Deeper/wider network.
- Multi-angle prediction.
- Learning rate decay.

5.
Evaluation


A 733 2 4 5 4 1 11 5 0 0 1 0 0 2 4 1 16 2 1 1 5 0 0 0 0 2
B 6 748 0 4 3 0 9 12 0 1 0 5 0 2 0 1 2 3 0 0 1 0 0 0 0 3
C 3 1 735 1 27 0 3 1 0 1 3 2 0 0 3 1 0 9 3 2 1 0 1 0 0 3
D 5 6 0 736 0 1 1 5 2 6 0 1 0 1 23 1 2 1 2 1 4 0 0 0 2 0
E 3 4 6 1 759 2 2 0 1 1 1 2 0 0 3 7 1 3 1 0 0 0 1 0 0 2
F 1 0 0 1 2 744 2 1 1 1 0 0 0 0 0 21 0 6 3 16 0 0 0 0 1 0
G 20 9 2 1 2 9 657 1 1 2 0 0 0 1 1 3 71 0 8 0 0 0 0 1 9 2
H 3 7 0 1 0 0 2 746 1 1 6 8 2 10 0 0 0 1 0 2 3 1 4 2 0 0
I 0 0 2 3 1 0 1 0 643 18 0 123 1 0 0 1 0 3 1 0 0 1 1 0 0 1
J 0 2 0 3 0 2 2 0 19 751 0 3 0 0 1 0 2 1 2 7 1 1 0 1 1 1
K 2 4 0 0 2 2 0 14 0 1 746 4 2 1 0 1 0 6 0 4 1 0 2 6 0 2
L 0 0 4 1 0 0 0 6 229 2 3 544 0 0 2 0 2 1 0 1 0 3 0 0 2 0
M 2 0 0 0 0 0 0 4 0 0 1 0 775 9 0 1 0 0 0 1 2 0 4 0 1 0
N 10 1 0 1 0 0 0 15 1 1 4 0 12 736 0 0 1 7 0 0 1 0 8 1 1 0
O 3 0 1 15 0 0 4 0 0 0 0 0 0 0 769 2 1 0 1 0 3 1 0 0 0 0
P 1 1 0 5 1 9 0 0 0 0 1 0 1 0 0 772 5 3 0 1 0 0 0 0 0 0
Q 28 3 0 5 2 5 97 0 2 0 0 0 0 1 10 5 625 1 1 3 1 1 3 1 4 2
R 7 1 2 2 5 1 0 2 1 0 7 1 1 1 0 2 2 736 0 5 0 9 1 5 5 4
S 3 0 0 2 0 1 10 1 1 16 0 0 0 1 0 0 0 0 763 1 0 0 0 0 1 0
T 1 4 1 1 2 3 0 2 3 1 4 0 0 0 2 0 0 4 0 758 0 0 0 4 7 3
U 2 1 1 1 0 0 1 7 0 2 2 1 0 3 2 0 1 0 1 0 722 39 9 0 5 0
V 0 0 0 1 0 0 0 0 1 1 0 0 2 3 0 0 1 6 0 0 18 756 0 0 11 0
W 0 0 0 0 0 0 0 3 1 0 1 0 3 10 0 0 0 0 0 0 5 3 772 0 1 1
X 1 0 0 0 0 1 1 1 0 1 16 0 2 3 0 0 1 0 0 0 1 7 2 749 12 2
Y 0 1 0 2 0 1 8 3 0 5 0 0 0 0 0 3 5 3 0 6 1 18 0 6 738 0
Z 3 0 3 3 2 1 6 0 3 0 0 0 0 0 0 0 2 1 0 1 0 1 0 6 1 767
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
700
600
500
400
300
200
100
0

Some misclassified examples



Top-K accuracy

A correct prediction have the ground truth value in top K most confident classes of the prediction.

Our best model:

- ➔ Top-1: 91.25%
- ➔ Top-2: 97.63%
- ➔ Top-5: 99.25%

6.
Application

Predicting digital images of letters

Problems:

- The model only takes grayscale inputs.
- The image size must be 28x28.
- Train/test images have a common standard: black background and letter foreground.

Predicting digital images of letters

Straight conversation to grayscale and resizing just doesn't work!

The 28x28 'S' letter image above was predicted to be [Q, G, A, M, W].

Our workaround

1. Resize the image to a fixed size (300x300).
2. Apply K-mean segmentation on the image with K=2.
3. Find the ‘background’ and ‘foreground’ classes, assuming the background has more pixels.
4. Set all the pixels in background class to 0 and foreground class to 1.
5. Rescale to 28x28.
6. Apply Gaussian blur.

Our workaround

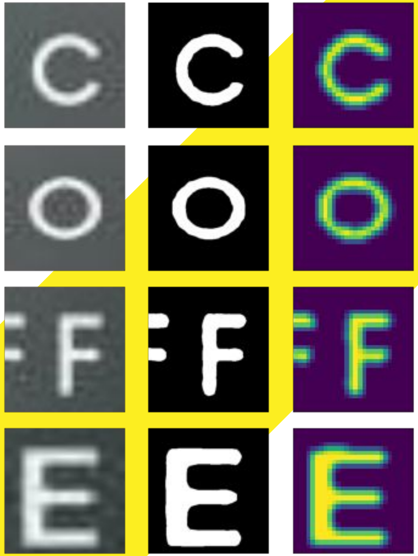
In most cases, the example becomes closer to what had been seen in the training set, thus allowing more precise prediction.

The ‘S’ letter is now correctly predicted.

Some correctly predicted samples

Better use with letter segmentation

In the example below, all 4 letters once segmented were correctly predicted.



7.
Final words

Summary

We implemented and trained NNs without any high-end libraries on the EMNIST letters dataset. Top result achieved 91.25% accuracy on the test set.

We showed that the simple model can be used for some real-world images application.

Takeaway

We gained a better understanding of neural networks and related concepts.

We now have more experience in actually debugging and improving a model.

We learned to put our ML model to potential real-world applications.

Thanks!

Any questions?