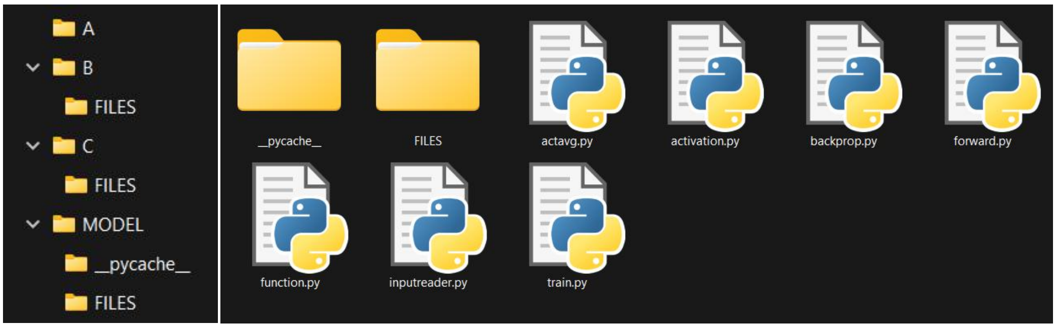
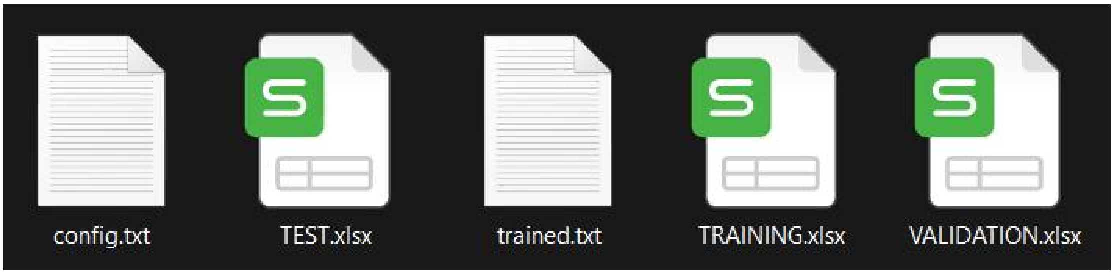
How This ANN works

'train.py' for training the ANN and **'function.py'** for using trained ANN
>> all other files are modules made to be used

All usage related files are to be placed in **'//FILES'** sub-folder



FILES contains:



config.txt: contains info on the ANN structure and Functions

TEST, TRAINING, VALIDATION: contain sample data

trained.txt: contains the trained model saved by the training process

---

Prerequisites: numpy, matplotlib, openpyxl

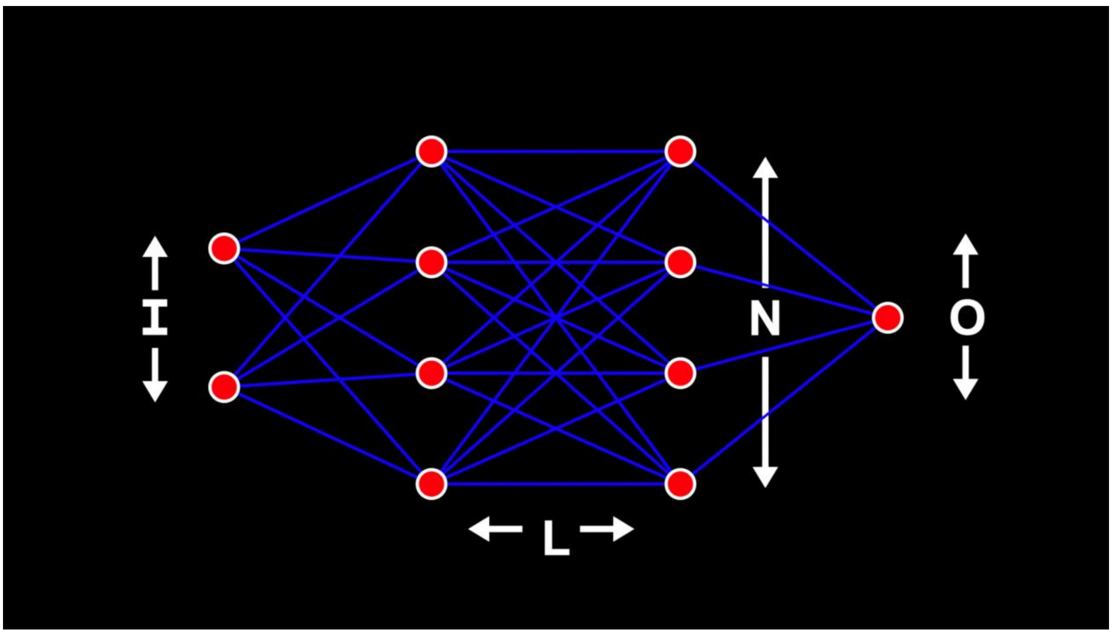
CONFIG FILE:

```
2
4
3
1
1
1
180
0.001
0.0001
0
```

```python
f = open('files\\config.txt', "r")

L = int(f.readline())
N = int(f.readline())
Af = int(f.readline())
I = int(f.readline())
O = int(f.readline())

B = int(f.readline())
EP = int(f.readline())
lrate = float(f.readline())
mom = float(f.readline())
reg = float(f.readline())
```

L - Number of Hidden Layers
N - Neurons per Hidden Layer
Af - Activation Function (1: tanh, 2: logistic, 3: ReLU)
I - Number of inputs
O - Number of Outputs

B - Batch size
EP - Number of Epochs
lrate - learning rate parameter
mom - momentum
reg - regularization constant



TRAINING (DATA) FILEs:

	A	B	C	D	E
1	14.96	41.76	1024.07	73.17	463.26
2	25.18	62.96	1020.04	59.08	444.37
3	5.11	39.4	1012.16	92.14	488.56
4	20.86	57.32	1010.24	76.64	446.48
5	10.82	37.5	1009.23	96.62	473.9
6	26.27	59.44	1012.23	58.77	443.67
7	15.89	43.96	1014.02	75.24	467.35

> The training data needs to be in ***xlsx*** files without any labels

> The first **I** (from config file) elements must be inputs

> The last **O** (from config file) elements must be outputs

e.g. data sample from CCPP dataset

14.96	41.76	1024.07	73.17	463.26

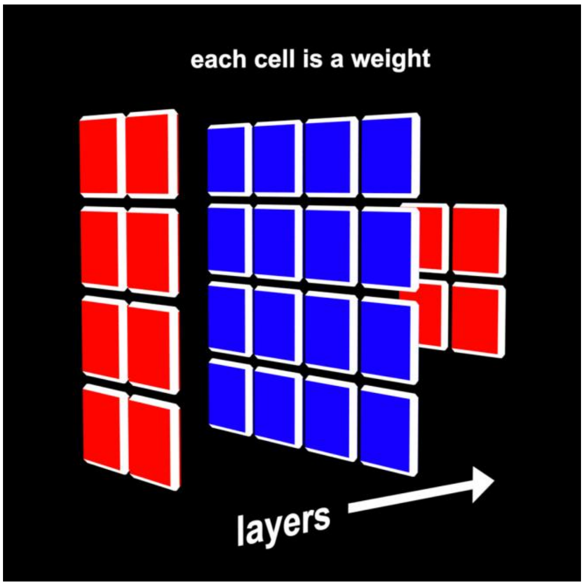
First 4 value(s) are input
Last 1 value(s) is output

How ‘train.py’ works:

Initialization:

Initialize Neurons, Weights, Biases, Delta Weights and Delta Biases

```
#_weights_random_initialization
Wf = None
W = []
W.append( np.random.rand(I, N))

for i in range(1, L):
    W.append( np.random.rand(N,N))

W.append( np.random.rand(N, O))

#_neurons_zero_initialization
An = []
An.append( np.zeros(I))

for i in range(1, L+1):
    An.append( np.zeros(N))

An.append( np.zeros(O))
```



Weights are stored as such:
A list of matrices (2D arrays)

Biases/Neuron activations are stored as such:
A list of 1D arrays

Loading samples:

Training and Validation data is loaded via a inputreader module
The samples are min-max mapped between (0,1)

```
##########################################################################
#_processing_training_data

train_i = ip.inparse(1, I, Af, "TRAINING")
train_o = ip.inparse(2, O, Af, "TRAINING")

##########################################################################
#_processing_validation_data

vali_i = ip.inparse(1, I, Af, "VALIDATION")
vali_o = ip.inparse(2, O, Af, "VALIDATION")
```

*name*_I contains inputs, and *name*_o contains outputs

Training:

```
for ep in range(EP):
```

Runs for as many epochs as specified

```
seed = random.random()

random.Random(seed).shuffle(train_i)
random.Random(seed).shuffle(train_o)
```

Shuffles data at every epoch

```
BAn[i%B] = fwd.output(W, An, O, Af, v, L, Bias)
BErr[i%B] = fwd.erroratout(train_o[i], BAn[i%B][L+1])
```

Performs ***function calculations*** and then finds error for a batch

```
if(i%B == (B-1)):
    An = aavg.batchaverage(BAn, An, B)
    Err = aavg.batcherr(BErr, Err, B)

    (dW, dB)  = backp.backprop(dW, dB, W, Bias, L, An, Err, lrate, mom, Af, I, N, O, reg)

    (W, Bias) = backp.upW(W, Bias, dW, dB, L)
```

At the end of each batch

>Average activations and Average errors are calculated

>Change in Weights and Biases is calculated (***back-prop***)

>Weights and Biases are updated

>Weights corresponding to the **Lowest validation error** are saved

```
##########################################################################
#plotting_error_percentage
plt.title("Err Percent vs Epochs")
plt.xlabel("epoch")
plt.ylabel("error")
plt.plot(cfv,label = "validation" , color = 'red')
plt.plot(cft, label = "training" , color = 'blue')
plt.axvline(x = np.argmin(cfv), label = "validation lowest error", color = 'red',linestyle='dashed')
plt.axvline(x = np.argmin(cft), label = "training lowest error", color = 'blue',linestyle='dashed')
leg = plt.legend(loc='center right')
plt.show()
```

Error% vs Epochs is plotted

Batch Average:

Activation:
For all batches:
    nodeAvg = sum(nodes)/batchSize

Error:
For all batches:
    errorAvg = sum(errors)/batchSize

Activation Functions:

Activations are calculated by the activation module
Which has 3 main features:

> actfunc	activation function
> iactfunc	inverse activation function
> dactfunc	derivative of activation function

All three take 2 parameters:

1. Value	value to be calculated upon
2. Func	which function to use

Function Calculations:
Function calculations are done in the **'forward.py'** module
    >in **output** function as follow

```
For layers in [0, L+2):
    If(layer = input layer):
        Activations = Input
    Else:
        Activations = act(WxA[l-1] + B)  #act is activation function
```

Error is calculated by vector subtraction
    >in **erroratout** function

BackProp Calculations:

Local Gradient is calculated for all nodes using equations written in (A)

$$\delta_L = e_L \, \phi'_L(v_L(n))$$

if (L is outlayer)

$$\{ e_L = d_L - y_L \}$$

else

$$\{ e_L = \sum_{L+1} \delta_{L+1} \cdot w_{(L+1)(L)} \}$$

Once local gradient is calculated, change In Weights and Biases is calculated

$$\Delta w_{(L)(L-1)} = \eta \cdot \delta_{(L)} \, y(n)_{(L-1)}$$

(Momentum and Regularization implemented in code)

```
M = np.size(dW[i])
dW[i][j][k] = mom*dW[i][j][k] + (1-mom)*lrate*An[i][j]*LG[i+1][k] - (reg*dW[i][j][k]/M)
```

```
M = np.size(dB[i])
dB[i][j] = mom*dB[i][j] + (1-mom)*lrate*LG[i][j] - (reg*dB[i][j]/M)
```

Saving Data:

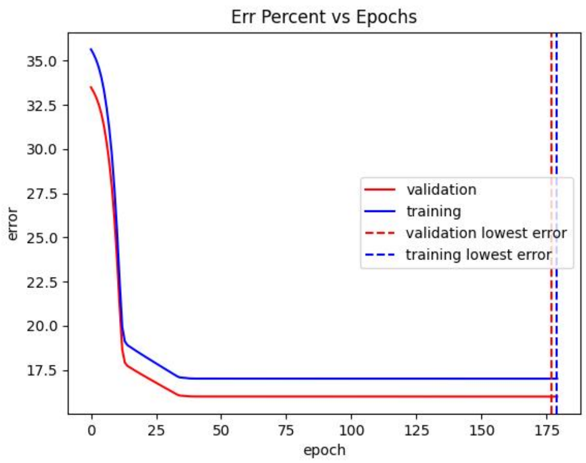
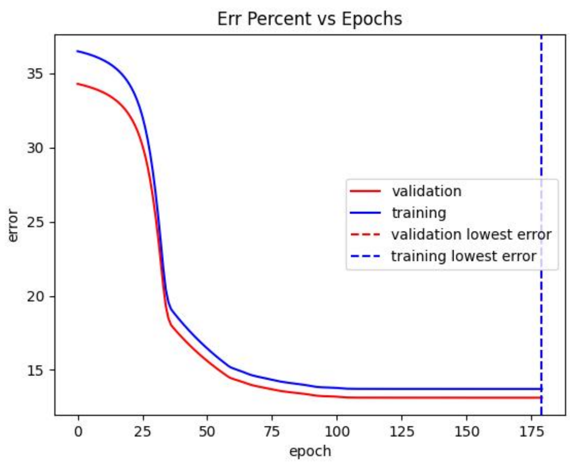
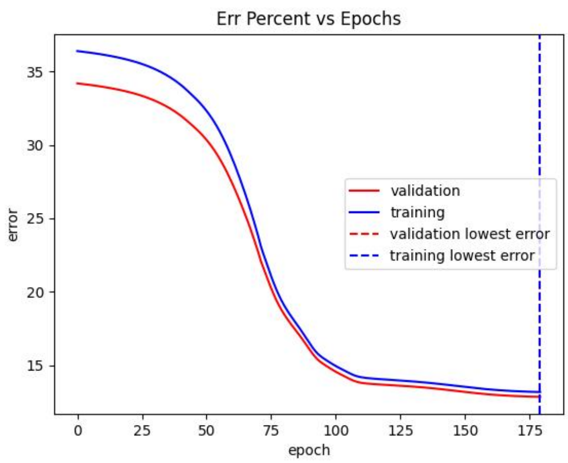
Data is separated by new line and is labelled:

Data in same layer is placed between
[ ]

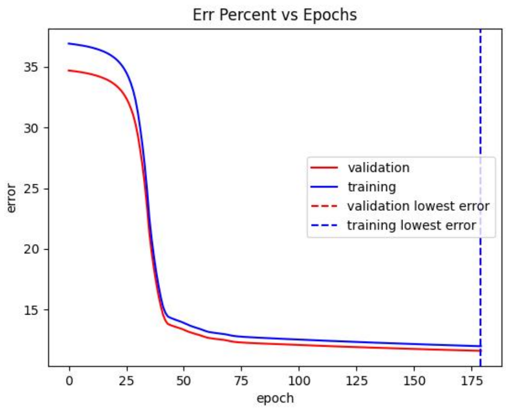
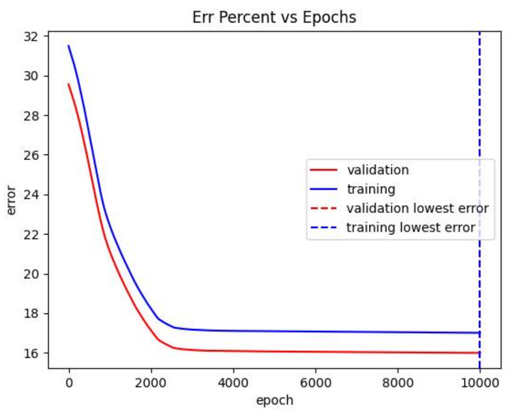
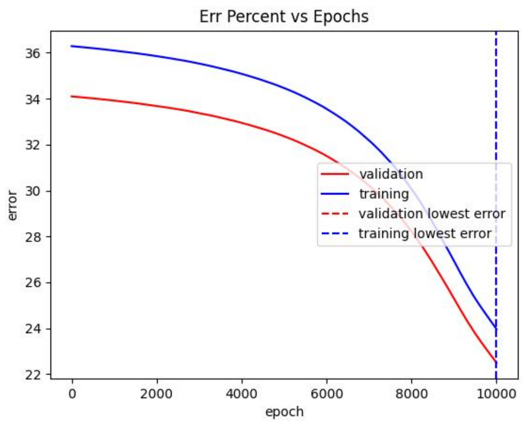
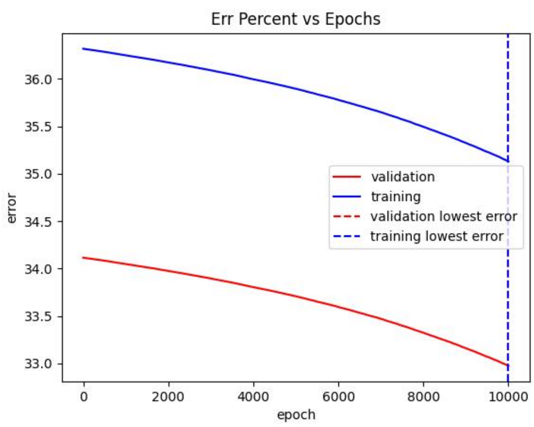
Connection every neuron has with next layer neurons is between
(only for weights)
{}

Impact of parameters:

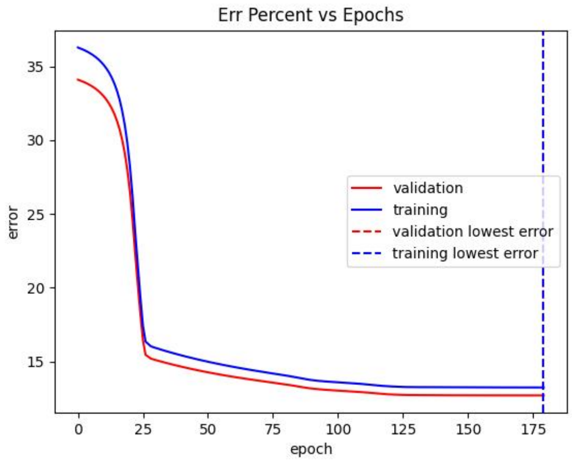
Structure:

	• 3 hidden layers • 3 neurons each • ReLU
	• 2 hidden layers • 4 neurons each • ReLU
	• 1 hidden layers • 15 neurons each • ReLU

Batch Size:

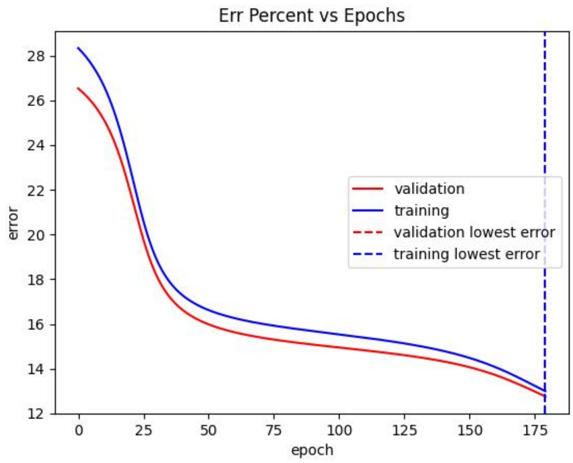
	● 2 hidden layers ● 4 neurons each ● ReLU ● Batch Size 1
	● 2 hidden layers ● 4 neurons each ● ReLU ● Batch Size 64
	● 2 hidden layers ● 4 neurons each ● ReLU ● Batch Size 256
	● 2 hidden layers ● 4 neurons each ● ReLU ● Batch Size FULL

Activation function:

<table>
<tr>
<td>


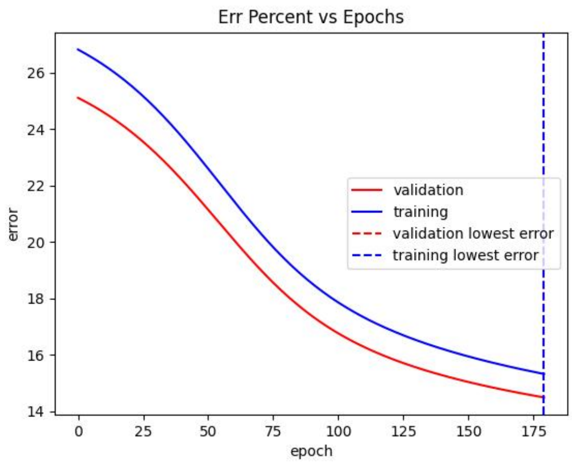
</td>
<td>

- 2 hidden layers
- 4 neurons each
- ReLU

</td>
</tr>
<tr>
<td>


</td>
<td>

- 2 hidden layers
- 4 neurons each
- TanH

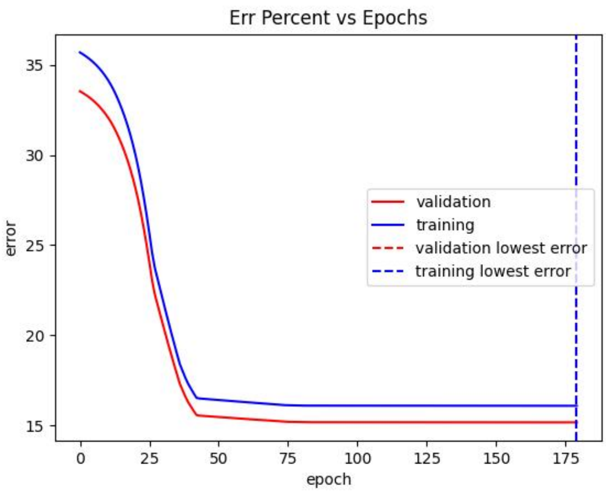
</td>
</tr>
<tr>
<td>


</td>
<td>

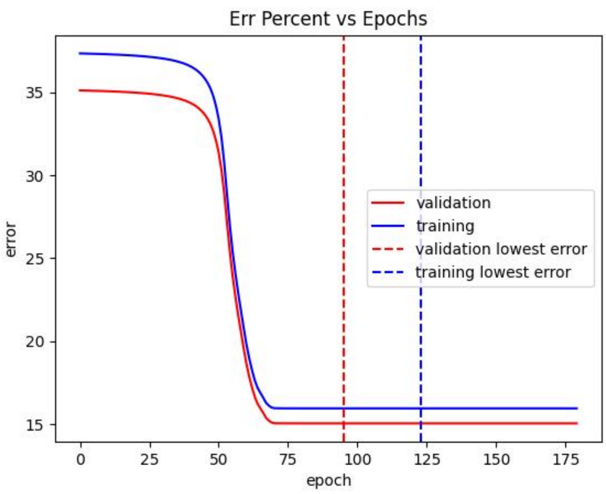
- 2 hidden layers
- 4 neurons each
- Logistic

</td>
</tr>
</table>

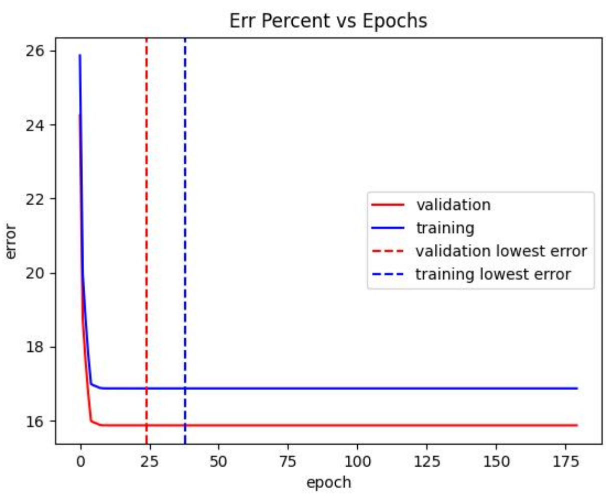
Learning Rate:



- 2 hidden layers
- 4 neurons each
- ReLU
- LRate 0.001

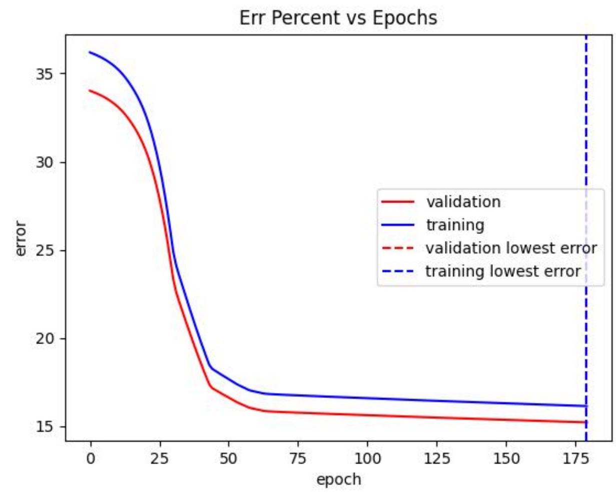
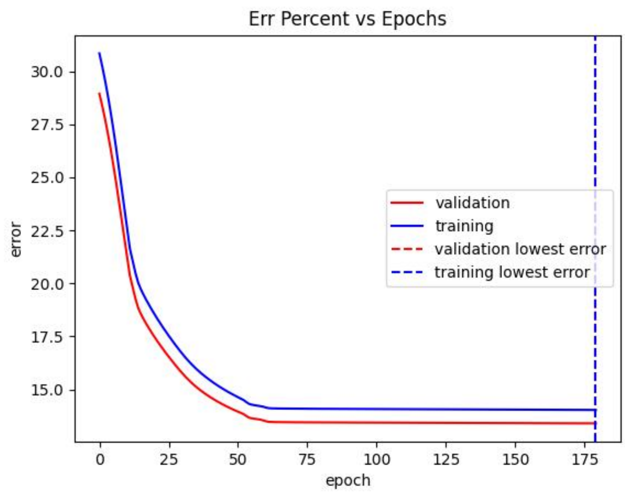
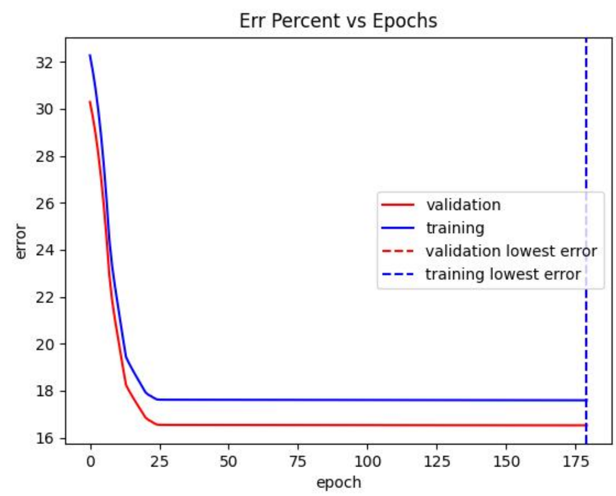


- 2 hidden layers
- 4 neurons each
- ReLU
- LRate 0.002



- 2 hidden layers
- 4 neurons each
- ReLU
- LRate 0.01

Momentum:

 	● 2 hidden layers ● 4 neurons each ● ReLU ● LRate 0.001 ● Mom 0.001
 	● 2 hidden layers ● 4 neurons each ● ReLU ● LRate 0.001 ● Mom 0.002
 	● 2 hidden layers ● 4 neurons each ● ReLU ● LRate 0.001 ● Mom 0.01