CS/ECE/ME532 Assignment 10

1. Neural net functions

a) Sketch the function generated by the following 3-neuron ReLU neural network.

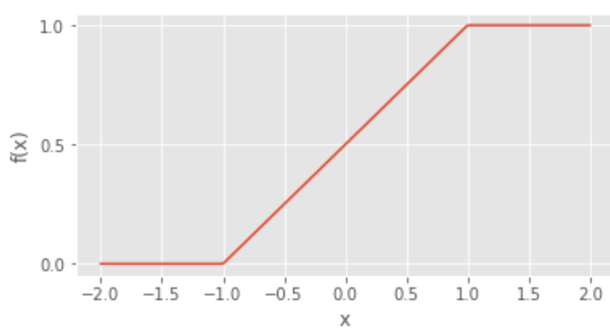
$$f(x) = 2(x - 0.5)_+ - 2(2x - 1)_+ + 4(0.5x - 2)+$$

where $x \in \mathbb{R}$ and where $(z)_+ = max(0, z)$ for any $z \in \mathbb{R}$. Note that this is a single-input, single-output function. Plot $f(x)$ vs x by hand.

b) Consider the continuous function depicted below. Approximate this function with ReLU neural network with 2 neurons. The function should be in the form

$$f(x) = \sum_{j=1}^{2} v_j(w_j x + b_j)_+$$

Indicate the weights and biases of each neuron and sketch the neural network function.



c) A neural network f_w can be used for binary classification by predicting the label as $\hat{y} = \text{sign}(f_w(\mathbf{x}))$. Consider a setting where $\mathbf{x} \in \mathbb{R}^2$ and the desired classifier is -1 if both elements of $\mathbf{x}$ are less than or equal to zero and $+1$ otherwise. Sketch the desired classification regions in the two-dimensional plane, and provide a formula for a ReLU network with 2-neurons that can produce the desired classification. For simplicity, assume in this questions that $\text{sign}(0) = -1$.

2. Gradients of a neural net. Consider a 2 layer neural network of the form $f(\boldsymbol{x}) = \sum_{j=1}^{J} v_j(\mathbf{w}_j^T \boldsymbol{x})_+$. Suppose we want to train our network on a dataset of N samples $\mathbf{x}_i$ with corresponding labels y_i, using a least squares loss function $\mathcal{L} = \sum_{i=1}^{n}(f(\boldsymbol{x}_i) - y_i)^2$. Derive the gradient descent update steps for the input weights $\mathbf{w}_j$ and output weights v_j.

3. **Compressing neural nets.** Large neural network models can be approximated by considering low rank approximations to weight matrices. The neural network $f(\boldsymbol{x}) = \sum_{j=1}^{J} \boldsymbol{v}_j(\mathbf{w}_j^T \boldsymbol{x})_+$ can be written as

$$f(\boldsymbol{x}) = \boldsymbol{v}^T(\mathbf{W}\boldsymbol{x})_+.$$

where $\mathbf{v}$ is a $J \times 1$ vector of the output weights and $\mathbf{W}$ is a $J \times d$ matrix with ith row $\mathbf{w}_j^T$. Let $\sigma_1, \sigma_2, \ldots$ denote the singular values of $\mathbf{W}$ and assume that $\sigma_i \leq \epsilon$ for $i > r$. Let f_r denote the neural network obtained by replacing $\mathbf{W}$ with its best rank r approximation $\hat{\mathbf{W}}_r$. Assuming that $\boldsymbol{x}$ has unit norm, find an upper bound to the difference $\max_x |f(\boldsymbol{x}) - f_r(\boldsymbol{x})|$. (Hint: for any pair of vectors $\mathbf{a}$ and $\mathbf{b}$, the following inequality holds $\|\mathbf{a}_+ - \mathbf{b}_+\|_2 \leq \|\mathbf{a} - \mathbf{b}\|_2$).

4. **Face Emotion Classification with a three layer neural network**. In this problem we return to the face emotion data studied previously. You may find it very helpful to use code from an activity (or libraries such as Keras and Tensorflow).

   a) Build a classifier using a full connected three layer neural network with logistic activation functions. Your network should

   - take a vector $\boldsymbol{x} \in \mathbb{R}^{10}$ as input (nine features plus a constant offset),
   - have a single, fully connected hidden layer with 32 neurons
   - output a scalar $\hat{y}$.

   Note that since the logistic activation function is always positive, your decision should be as follows: $\hat{y} > 0.5$ corresponds to a 'happy' face, while $\hat{y} \leq 0.5$ is not happy.

   b) Train your classifier using stochastic gradient descent (start with a step size of $\alpha = 0.05$) and create a plot with the number of epochs on the horizontal axis, and training accuracy on the vertical axis. Does your classifier achieve 0% training error? If so, how many epoch does it take for your classifier to achieve perfect classification on the training set?

   c) Find a more realistic estimate of the accuracy of your classifier by using 8-fold cross validation. Can you achieve perfect test accuracy?

S32
Assignment 10
DEVIN BRESSER

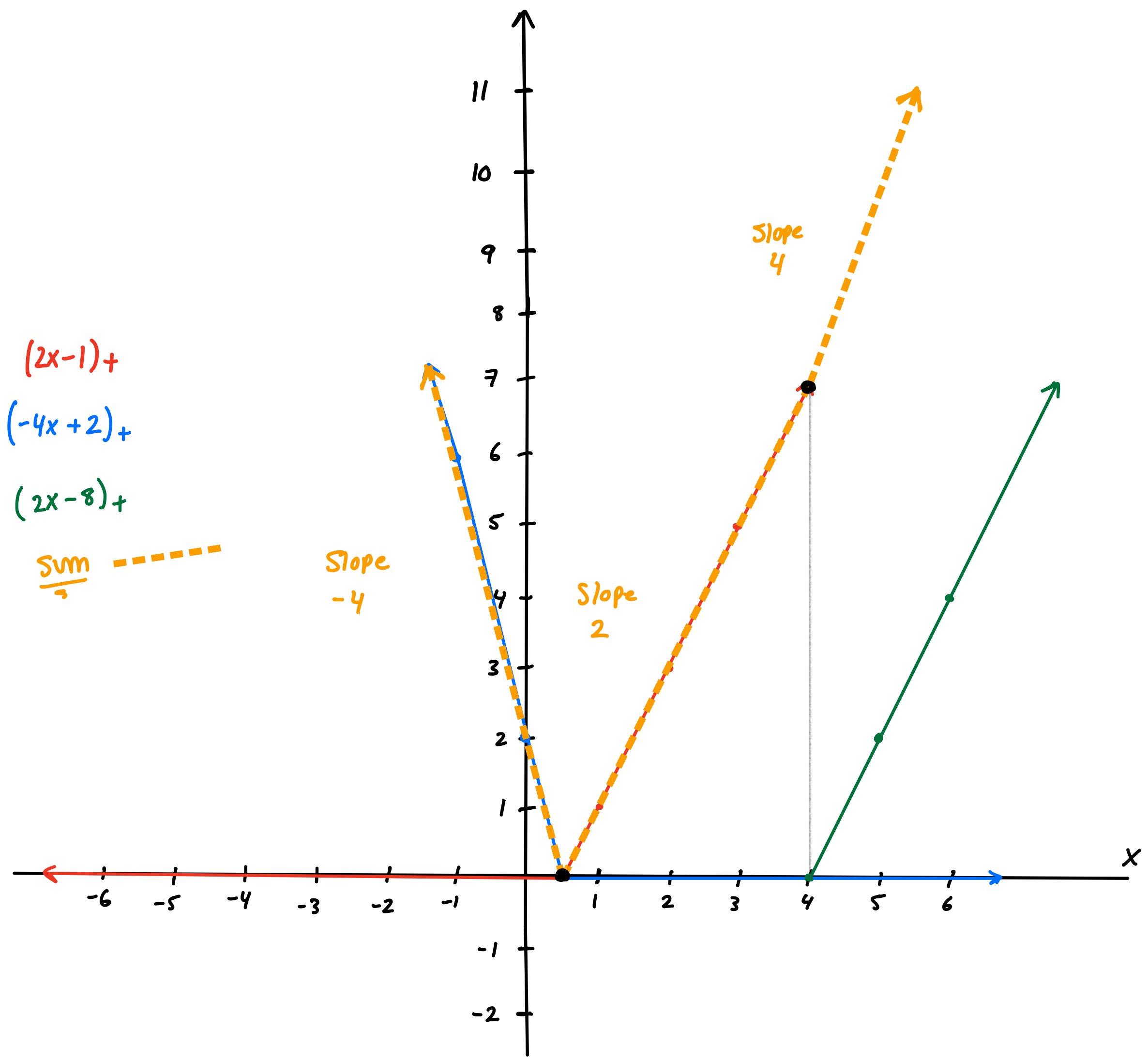
1. Neural net functions

a) Sketch the function generated by the following 3-neuron ReLU neural network.

$$f(x) = 2(x - 0.5)_+ - 2(2x - 1)_+ + 4(0.5x - 2)+$$

where $x \in \mathbb{R}$ and where $(z)_+ = max(0, z)$ for any $z \in \mathbb{R}$. Note that this is a single-input, single-output function. Plot $f(x)$ vs x by hand.

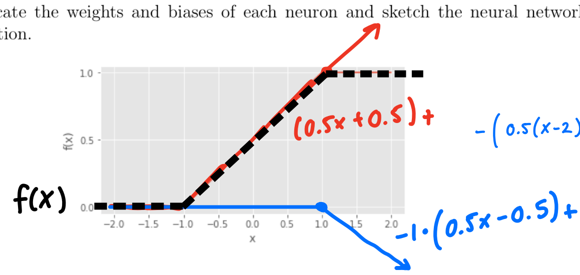
1a)



b) Consider the continuous function depicted below. Approximate this function with ReLU neural network with 2 neurons. The function should be in the form

$$f(x) = \sum_{j=1}^{2} v_j (w_j x + b_j)_+$$

Indicate the weights and biases of each neuron and sketch the neural network function.



$-(0.5(x-2)+0.5)_+ = -(0.5x-0.5)_+$

1b.)

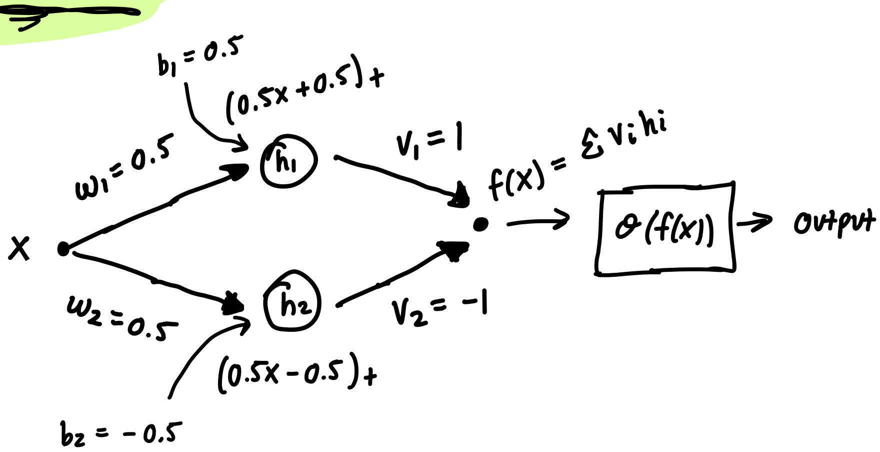
$$f(x) = (0.5x + 0.5)_+ - (0.5x - 0.5)_+$$

$$= \sum_{j=1}^{2} v_j (w_j x + b_j)$$

$$\underline{v} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad \underline{w} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

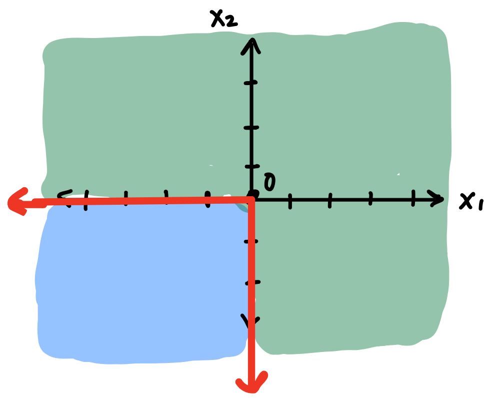
$$\underline{b} = \begin{bmatrix} 0.5 \\ -0.5 \end{bmatrix}$$

Neural Network:



c) A neural network f_w can be used for binary classification by predicting the label as $\hat{y} = \text{sign}(f_w(\mathbf{x}))$. Consider a setting where $\mathbf{x} \in \mathbb{R}^2$ and the desired classifier is -1 if both elements of $\mathbf{x}$ are less than or equal to zero and $+1$ otherwise. Sketch the desired classification regions in the two-dimensional plane, and provide a formula for a ReLU network with 2-neurons that can produce the desired classification. For simplicity, assume in this questions that $\text{sign}(0) = -1$.

$\hat{y} = \text{sign}(f_w(\underline{x}))$. $\quad \hat{y} = \begin{cases} -1, & x_1 \le 0 \ \& \ x_2 \le 0 \\ 1, & \text{otherwise} \end{cases}$



→ $\hat{y} = -1$

→ $\hat{y} = 1$

→ ReLU function

$$f_w(\underline{x}) = (x_1)_+ + (x_2)_+$$

$$\hat{y} = \text{sign}(f_w(\underline{x})) = \begin{cases} -1, & x_1 \le 0 \ \& \ x_2 \le 0 \\ 1, & \text{otherwise} \end{cases}$$

(assuming $\text{sign}(0) = -1$)

2. Gradients of a neural net. Consider a 2 layer neural network of the form $f(\boldsymbol{x}) = \sum_{j=1}^{J} v_j(\mathbf{w}_j^T \boldsymbol{x})_+$. Suppose we want to train our network on a dataset of N samples $\mathbf{x}_i$ with corresponding labels y_i, using a least squares loss function $\mathcal{L} = \sum_{i=1}^{n}(f(\boldsymbol{x}_i) - y_i)^2$. Derive the gradient descent update steps for the input weights $\mathbf{w}_j$ and output weights v_j.

v_j

$$\frac{\partial L}{\partial v_j} = \sum_{i=1}^{N} 2(f(\underline{x}_i) - y_i) \cdot \frac{\partial f(x_i)}{\partial v_j}$$

$$= \sum_{i=1}^{N} (2(f(\underline{x}_i) - y_i) \cdot \frac{\partial \sum_{j=1}^{J} v_j(\underline{w}_j^T \underline{x})_+}{\partial v_j}$$

$$= \sum_{j=1}^{N} (2(f(\underline{x}_i) - y_i) \cdot (\underline{w}_i^T \underline{x})_+$$

$\underline{w_j}$

$$\frac{\partial L}{\partial w_j} = \sum_{i=1}^{N} 2\left(f(\underline{x}_i) - y_i\right) \cdot \frac{\partial f(\underline{x}_i)}{\partial w_j}$$

$$= \sum_{i=1}^{N} 2\left(f(\underline{x}_i) - y_i\right) \cdot \frac{\partial \sum_{j=1}^{J} v_j \left(\underline{w}_j^T \underline{x}_i\right)}{\partial w_j}$$

$\frac{d}{dx} ReLU(x) = \mathbb{1}\{x > 0\}$

$$= \sum_{i=1}^{N} 2\left(f(\underline{x}_i) - y_i\right) \cdot v_j \cdot \mathbb{1}\{\underline{w}_j^T \underline{x}_i > 0\} \cdot x_i$$

$\underline{\text{Gradient Descent}}$

1.) Initialize $w_{m,j}^{(0)}$, $v_{k,\ell}^{(0)}$

2.) For $t = 0, 1, 2 \ldots$

- Choose $i_t \in \{0, 1, \ldots N\}$ at random
- Compute h_m^{it}, $\hat{d}_q^{it}$ from x_ℓ^{it}, $w_{m,j}^{t}$, $v_{k,\ell}^{t}$

V update step $\longrightarrow$ • $v_{k,\ell}^{(t+1)} = v_{k,\ell} - \alpha_t \cdot \sum_{j=1}^{N} \left(2\left(f(\underline{x}_i) - y_i\right) \cdot \left(\underline{w}_i^T \underline{x}\right)_+\right.$

W update step $\longrightarrow$ • $w_{m,j}^{(t+1)} = w_{m,j} - \alpha_t \cdot \sum_{i=1}^{N} 2\left(f(\underline{x}_i) - y_i\right) \cdot v_j \cdot \mathbb{1}\{\underline{w}_j^T \underline{x}_i > 0\} \cdot x_i$

3. Compressing neural nets. Large neural network models can be approximated by considering low rank approximations to weight matrices. The neural network $f(\boldsymbol{x}) = \sum_{j=1}^{J} \boldsymbol{v}_j(\mathbf{w}_j^T \boldsymbol{x})_+$ can be written as

$$f(\boldsymbol{x}) = \boldsymbol{v}^T(\mathbf{W}\boldsymbol{x})_+.$$

where $\mathbf{v}$ is a $J \times 1$ vector of the output weights and $\mathbf{W}$ is a $J \times d$ matrix with ith row $\mathbf{w}_j^T$. Let $\sigma_1, \sigma_2, \ldots$ denote the singular values of $\mathbf{W}$ and assume that $\sigma_i \leq \epsilon$ for $i > r$. Let f_r denote the neural network obtained by replacing $\mathbf{W}$ with its best rank r approximation $\hat{\mathbf{W}}_r$. Assuming that $\boldsymbol{x}$ has unit norm, find an upper bound to the difference $\max_x |f(\boldsymbol{x}) - f_r(\boldsymbol{x})|$. (Hint: for any pair of vectors $\mathbf{a}$ and $\mathbf{b}$, the following inequality holds $\|\mathbf{a}_+ - \mathbf{b}_+\|_2 \leq \|\mathbf{a} - \mathbf{b}\|_2$).

$$f(\underline{x}) = \sum_{j=1}^{J} \underline{v}_j(\underline{w}_j^T\underline{x})_+ = \underline{v}^T(\underline{W}\underline{x})_+$$

$$f_r(\underline{x}) = \underline{v}^T(\hat{\underline{W}}\underline{x})_+$$

$$\rightarrow |f(\underline{x}) - f_r(\underline{x})| = \|(\underline{v}^T\underline{W}\underline{x})_+ - (\underline{v}^T\hat{\underline{W}}_r\underline{x})_+\|_2 = \|\underline{v}^T\| \, \|(\underline{W}_r\underline{x})_+ - (\hat{\underline{W}}_r\underline{x})_+\|$$

$$\rightarrow \|\underline{v}\|_2 \, \|(\underline{W}\underline{x})_+ - (\hat{\underline{W}}_r\underline{x})_+\|_2 \leq \|\underline{v}\|_2 \underbrace{\|\underline{W}\underline{x} - \hat{\underline{W}}_r\underline{x}\|_2}$$

all s_i's $< \varepsilon$ for $i > r$

Thus this error can be at most ε since $\underline{x}$ is unit norm

$$\rightarrow \|\underline{v}\|_2 \, \|(\underline{W}\underline{x})_+ - (\hat{\underline{W}}_r\underline{x})_+\|_2 \leq \|\underline{v}\|_2 \cdot \varepsilon$$

$$\rightarrow |f(\underline{x}) - f_r(\underline{x})| \leq \|\underline{v}\|_2 \cdot \varepsilon$$

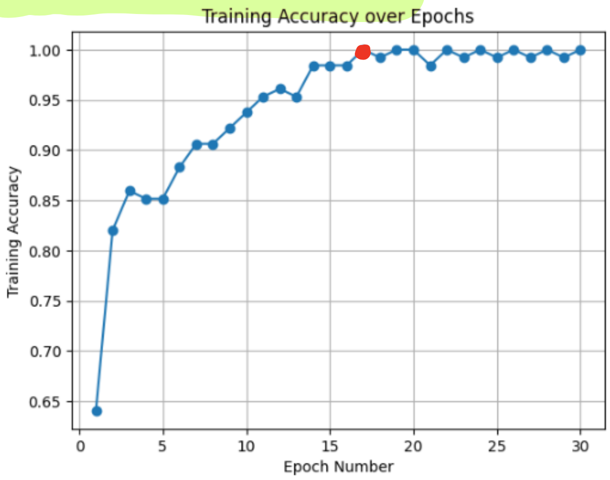
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

dataset = loadmat('face_emotion_data.mat')

X, y = dataset['X'], dataset['y']
n, p = np.shape(X)

y[y==-1] = 0  # use 0/1 for labels instead of -1/+1
X = np.hstack((np.ones((n,1)), X))  # append a column of ones
```

4a 3/3

```python
# Problem 4a - Devin Bresser

# define the model as follows:
# takes in a vector x E R^10
# one hidden layer with 32 neurons
# outputs a scalar y_hat E R

model = Sequential([
    Dense(32, activation='sigmoid', input_shape=(10,)), # hidden layer 32 neurons
    Dense(1, activation='sigmoid') # output layer
])

# compile the model with squared error loss function and SGD optimizer with learning rate 0.05
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.05), loss='mean_squared_error', metrics=['accuracy'])

# Problem 4b - training accuracy

# try with 30 epochs
max_epochs = 30
train_accuracies = []

for i in range(0, max_epochs):

    fit = model.fit(X, y, epochs=1, batch_size=1, verbose=0)

    # obtain the accuracy for this epoch
    train_accuracy = fit.history['accuracy'][0]
    train_accuracies.append(train_accuracy)

    #print(f"Completed Epochs: {i}, Accuracy: {train_accuracy:.4f}")

first_perfect_accuracy = next((i, acc) for i, acc in enumerate(train_accuracies, start=1) if acc == 1)

print(f"training accuracy of 1.0 first occurs at epoch # {first_perfect_accuracy[0]}")

# plotting:
epochs = list(range(1,max_epochs+1))

plt.plot(epochs, train_accuracies, marker='o', linestyle='-')
plt.title('Training Accuracy over Epochs')
plt.xlabel('Epoch Number')
plt.ylabel('Training Accuracy')
plt.grid(True)
plt.show()
```

4b 3/3

```
training accuracy of 1.0 first occurs at epoch # 17
```



Yes, 100% training accuracy is achievable with this model!

4c

```python
# Problem 4c - CV
# This code essentially runs the cross validation process with a number of epochs
# ranging from 1 to 100. We should see a convergence after a certain point,
# and from there be able to tell if it's possible to attain perfect test accuracy.

from sklearn.model_selection import KFold

max_epochs = 100
epoch_accuracies = []

# for each number of epochs, do the CV
for i in range(1, max_epochs):
    kf = KFold(n_splits=8, shuffle=True, random_state=42)
    fold_accuracies = []

    # do CV using KFold module
    for train_index, test_index in kf.split(X):

        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        # re-define the model each time
        model = Sequential([
            Dense(32, activation='sigmoid', input_shape=(10,)),
            Dense(1, activation='sigmoid')
        ])
        model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.05),
                      loss='mean_squared_error',
                      metrics=['accuracy'])

        # fit the model to the training part
        model.fit(X_train, y_train, epochs=i, batch_size=1, verbose=0) # picked 20 epochs based on training results

        # get the accuracy on the testing part and append to fold_accuracies
        _, accuracy = model.evaluate(X_test, y_test, verbose=0)
        fold_accuracies.append(accuracy)

    # compute the average accuracy across all folds
    average_accuracy = np.mean(fold_accuracies)

    # store average test accuracy for that epoch count
    print(f"average test accuracy at epoch {i}: {average_accuracy}")
    epoch_accuracies.append(average_accuracy)
```

```
average test accuracy at epoch 1: 0.7578125
average test accuracy at epoch 2: 0.8046875
average test accuracy at epoch 3: 0.8515625
average test accuracy at epoch 4: 0.84375
average test accuracy at epoch 5: 0.8828125
average test accuracy at epoch 6: 0.8828125
average test accuracy at epoch 7: 0.8984375
average test accuracy at epoch 8: 0.8828125
average test accuracy at epoch 9: 0.9140625
average test accuracy at epoch 10: 0.9375
average test accuracy at epoch 11: 0.9296875
average test accuracy at epoch 12: 0.9375
average test accuracy at epoch 13: 0.9296875
average test accuracy at epoch 14: 0.9375
average test accuracy at epoch 15: 0.9453125
average test accuracy at epoch 16: 0.953125
average test accuracy at epoch 17: 0.96875
average test accuracy at epoch 18: 0.9765625
average test accuracy at epoch 19: 0.9765625
average test accuracy at epoch 20: 0.9765625
average test accuracy at epoch 21: 0.9609375
average test accuracy at epoch 22: 0.9609375
average test accuracy at epoch 23: 0.96875
average test accuracy at epoch 24: 0.96875
average test accuracy at epoch 25: 0.9765625
average test accuracy at epoch 26: 0.9765625
average test accuracy at epoch 27: 0.9765625
average test accuracy at epoch 28: 0.96875
average test accuracy at epoch 29: 0.984375
average test accuracy at epoch 30: 0.984375
average test accuracy at epoch 31: 0.96875
average test accuracy at epoch 32: 0.96875
average test accuracy at epoch 33: 0.9765625
average test accuracy at epoch 34: 0.9921875
average test accuracy at epoch 35: 0.9765625
average test accuracy at epoch 36: 0.984375
average test accuracy at epoch 37: 0.984375
average test accuracy at epoch 38: 0.96875
average test accuracy at epoch 39: 0.9921875
average test accuracy at epoch 40: 0.984375
average test accuracy at epoch 41: 0.9765625
average test accuracy at epoch 42: 0.9765625
average test accuracy at epoch 43: 0.9921875
average test accuracy at epoch 44: 0.9765625
average test accuracy at epoch 45: 0.9765625
average test accuracy at epoch 46: 0.9765625
average test accuracy at epoch 47: 0.9765625
average test accuracy at epoch 48: 0.984375
average test accuracy at epoch 49: 0.984375
average test accuracy at epoch 50: 0.9765625
average test accuracy at epoch 51: 0.984375
average test accuracy at epoch 52: 0.96875
```

```
average test accuracy at epoch 48: 0.984375
average test accuracy at epoch 49: 0.984375
average test accuracy at epoch 50: 0.9765625
average test accuracy at epoch 51: 0.984375
average test accuracy at epoch 52: 0.96875
average test accuracy at epoch 53: 0.9765625
average test accuracy at epoch 54: 0.984375
average test accuracy at epoch 55: 0.9765625
average test accuracy at epoch 56: 0.984375
average test accuracy at epoch 57: 0.984375
average test accuracy at epoch 58: 0.984375
average test accuracy at epoch 59: 0.984375
average test accuracy at epoch 60: 0.9765625
average test accuracy at epoch 61: 0.984375
average test accuracy at epoch 62: 0.984375
average test accuracy at epoch 63: 0.9765625
average test accuracy at epoch 64: 0.9765625
average test accuracy at epoch 65: 0.984375
average test accuracy at epoch 66: 0.984375
average test accuracy at epoch 67: 0.984375
average test accuracy at epoch 68: 0.9765625
average test accuracy at epoch 69: 0.9765625
average test accuracy at epoch 70: 0.9765625
average test accuracy at epoch 71: 0.984375
average test accuracy at epoch 72: 0.9765625
average test accuracy at epoch 73: 0.9765625
average test accuracy at epoch 74: 0.984375
average test accuracy at epoch 75: 0.984375
average test accuracy at epoch 76: 0.984375
average test accuracy at epoch 77: 0.984375
average test accuracy at epoch 78: 0.9765625
average test accuracy at epoch 79: 0.9765625
average test accuracy at epoch 80: 0.9765625
average test accuracy at epoch 81: 0.9765625
average test accuracy at epoch 82: 0.9765625
average test accuracy at epoch 83: 0.984375
average test accuracy at epoch 84: 0.9765625
average test accuracy at epoch 85: 0.984375
average test accuracy at epoch 86: 0.9765625
average test accuracy at epoch 87: 0.984375
average test accuracy at epoch 88: 0.9765625
average test accuracy at epoch 89: 0.984375
average test accuracy at epoch 90: 0.9765625
average test accuracy at epoch 91: 0.984375
average test accuracy at epoch 92: 0.984375
average test accuracy at epoch 93: 0.984375
average test accuracy at epoch 94: 0.9765625
average test accuracy at epoch 95: 0.984375
average test accuracy at epoch 96: 0.984375
average test accuracy at epoch 97: 0.9765625
average test accuracy at epoch 98: 0.984375
average test accuracy at epoch 99: 0.984375
```

... conclusion:

100% avg. test accuracy isn't possible with this model.