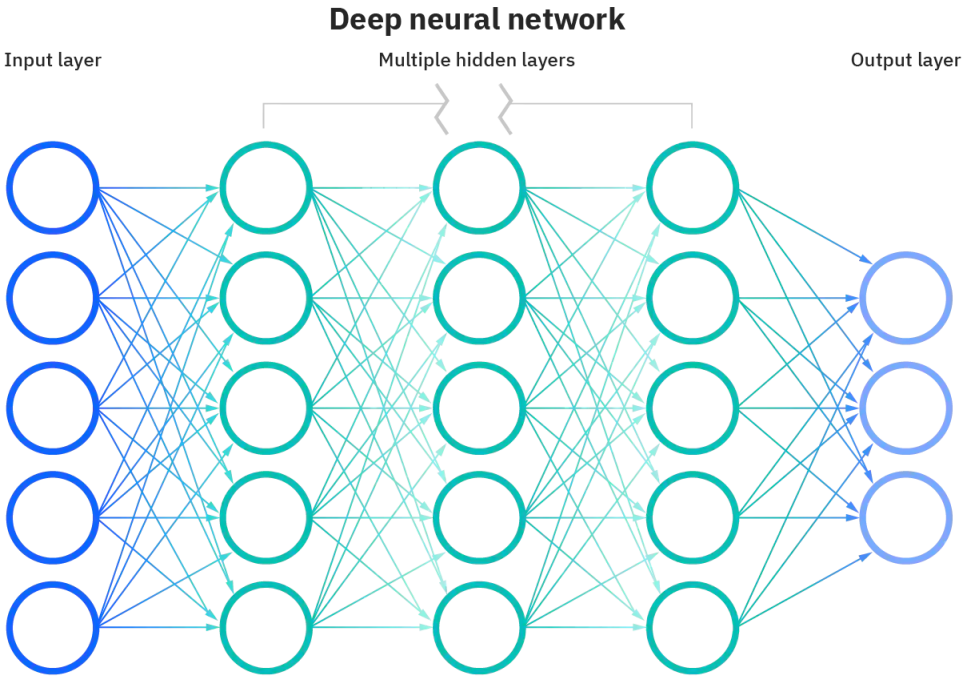
Αναγνώριση Προτύπων

2021-2022

Εργασία 7η

Artificial Neural Networks



Καρυπίδης Ευστάθιος 57556

1/1/22, Ξάνθη

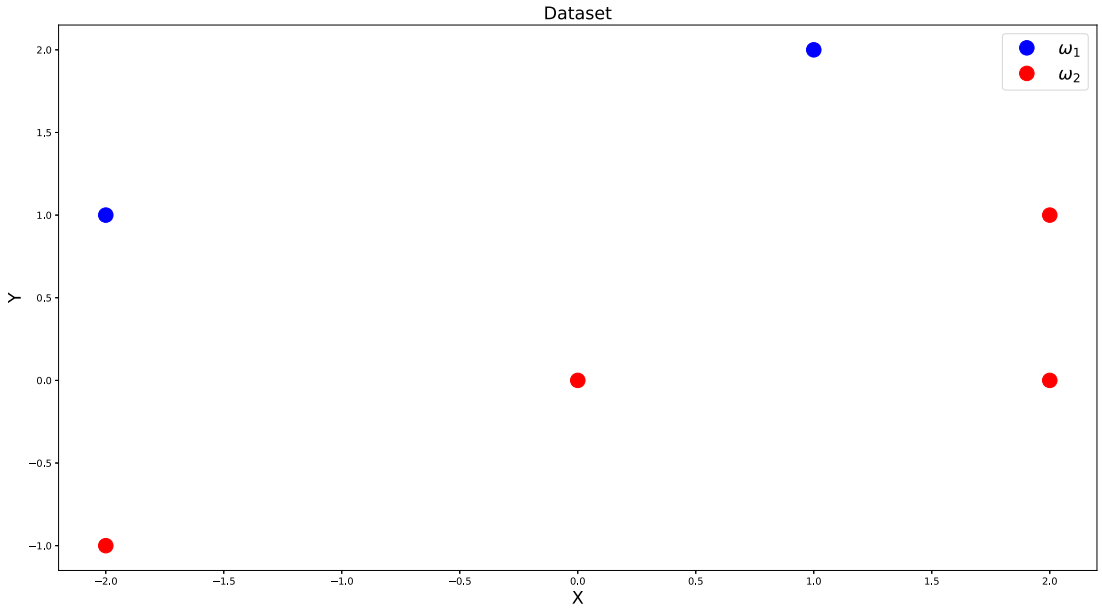
Για την υλοποίηση των ερωτημάτων της εργασίας έγινε χρήση γλώσσας python και των βιβλιοθηκών numpy,scipy και matplotlib.

Άσκηση 1

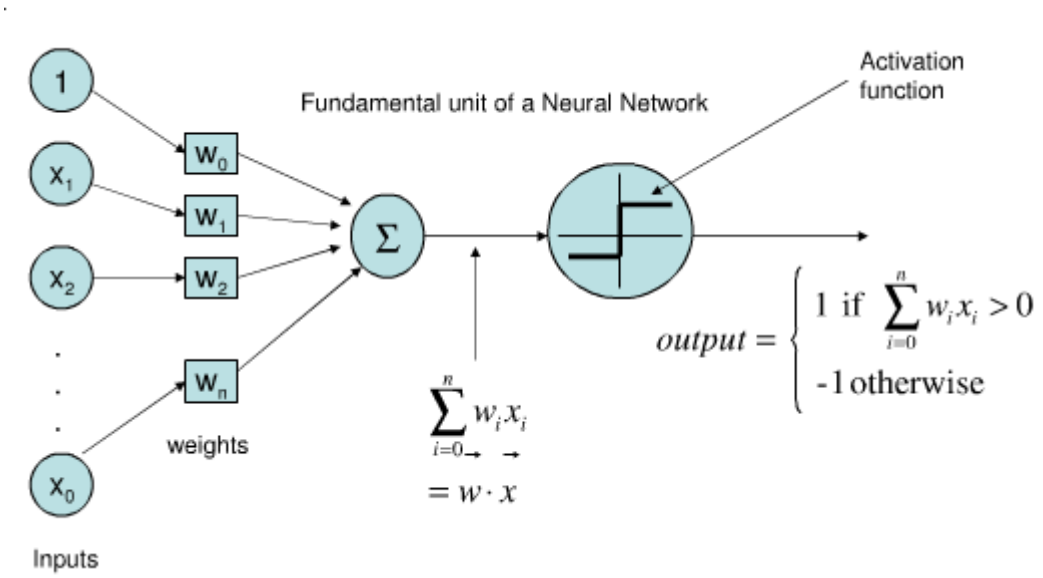
Δίνονται τα παρακάτω 6 δείγματα εκπαίδευσης:

$$S = \{\underbrace{\begin{pmatrix}-2\\1\end{pmatrix}, \begin{pmatrix}1\\2\end{pmatrix}}_{\omega_2}, \underbrace{\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}-2\\-1\end{pmatrix}, \begin{pmatrix}2\\0\end{pmatrix}, \begin{pmatrix}2\\1\end{pmatrix}}_{\omega_1}\}.$$

Χρησιμοποιώντας το κώδικα στο αρχείο **HW6_Exercise1.py** κάνουμε plot τα δεδομένα.



Όπως παρατηρούμε τα δεδομένα είναι γραμμικά διαχωρίσιμα. Συνεπώς ένα Perceptron ως γραμμικός ταξινομητής θα καταφέρει να συγκλίνει σε βάρη όπου θα ορίζουν μια ευθεία που διαχωρίζει τα δεδομένα. Το perceptron είναι η βασική μονάδα με την οποία υλοποιούνται τα νευρωνικά δίκτυα. Η λογική πίσω απο οτο perceptron περιγράφεται στο παρακάτω σχήμα. Ειδικότερα στην απλή περίπτωση η συνάρτηση ενεργοποίησης(activation function) είναι η βηματική συνάρτηση.



Αρχικά, ορίζουμε τα δεδομένα:

```
X = np.array([[-2, 1], [1, 2],
              [0, 0], [-2, -1], [2, 0], [2, 1]])
Y = np.array([0, 0, 1, 1, 1, 1])
```

Στο αρχείο κώδικα μπορούν να βρεθούν 2 υλοποιήσεις του perceptron. Η μία είναι η έτοιμη υλοποίηση που βρίσκεται στην sklearn. Ειδικότερα διαθέτει τα παρακάτω default ορίσματα:

sklearn.linear_model.Perceptron

class sklearn.linear_model.Perceptron(*, *penalty=None, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, eta0=1.0, n_jobs=None, random_state=0, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False*) [source]

Ωστόσο, από τα παραπάνω ορίσματα μας ενδιαφέρουν τα εξής:

- tol = 0.2. Ορίζουμε αυτή τη τιμή tol λίγο μεγαλύτερη του 1/6 που είναι η περίπτωση που υπάρχει 1 σφάλμα στα 6.

tol : *float, default=1e-3*

The stopping criterion. If it is not None, the iterations will stop when (loss > previous_loss - tol).

- shuffle = False. Δεν χρησιμοποιούμε την default τιμή καθώς θέλουμε να περνάει τα δεδομένα με τη σειρά που έρχονται.

shuffle : *bool, default=True*

Whether or not the training data should be shuffled after each epoch.

- verbose = 1. Δε χρησιμοποιούμε τη default τιμή ώστε να βλέπουμε τι γίνεαι σε κάθε επανάληψη.

verbose : *int, default=0*

The verbosity level.

Ακόμη, όταν καλούμε τη συνάρτηση `.fit()` έχουμε τα παρακάτω ορίσματα:

fit(*X, y, coef_init=None, intercept_init=None, sample_weight=None*)

Fit linear model with Stochastic Gradient Descent.

Parameters:

X : *{array-like, sparse matrix}, shape (n_samples, n_features)*
Training data.

y : *ndarray of shape (n_samples,)*
Target values.

coef_init : *ndarray of shape (n_classes, n_features), default=None*
The initial coefficients to warm-start the optimization.

intercept_init : *ndarray of shape (n_classes,), default=None*
The initial intercept to warm-start the optimization.

Στη περίπτωση αυτή ορίζουμε τα `coef_init`, `intercept_init` δηλαδή τις αρχικές τιμές των βαρών και της σταθεράς μεροληψίας(bias) σε 1.

Για να καλέσουμε τη συνάρτηση απο sklearn και για να πάρουμε τα βάρη και το bias έχουμε:

```
model = Perceptron(shuffle=False, verbose=1, n_iter_no_change=2, tol=0.2)
model.fit(X, Y, coef_init=[1, 1], intercept_init=1)
final_weights = model.coef_.tolist()[0]
final_bias = int(model.intercept_)
```

Παράλληλα, η δική μου συνάρτηση για το Perceptron είναι :

```
def my_perceptron(inp, out, tolerance=1e-3, max_iterations=10000):
    """
    :param inp: MxN matrix (M number of features, N number of input samples). Here 2x6
    :param out: {N,} vector
    :param tolerance: The stopping criterion. If it is not None, the iterations will stop
when (loss > previous_loss - tol).
    :param max_iterations: The maximum number of epochs
    :return: errors, weights (1xM), bias (1x1), w_list, b_list
    """
    iteration = 1
    errors, criteria = [], []
    w = np.ones([inp.shape[0], 1])
    b = 1
    w_list, b_list = [w], [b]
    while iteration < max_iterations:
        error = 0
        for i in range(inp.shape[1]):
            neuron_sum = np.dot(w.T, inp[:, i].reshape(-1, 1)) + b
            activation = 1 if int(neuron_sum) >= 0 else 0
            err = out[i] - activation
            w += err*inp[:, i].reshape(-1, 1)
            b += err
            error += abs(err)
        w_list.append(w)
        b_list.append(b)
        criterion = error/inp.shape[1]
        criteria.append(criterion)
        errors.append(error)
        if criterion < tolerance:
            break
        iteration += 1
    return errors, w, b, iteration, w_list, b_list
```

Στη περίπτωση αυτή σε κάθε εποχή υπολογίζεται για κάθε δείγμα η έξοδος του νευρώνα, συγκρίνεται με τη πραγματική και γίνεται ανανέωση των βαρών. Η εκπαίδευση ολοκληρώνεται όταν το σφάλμα ειναι μικρότερο από κάποια ανοχή(tol) όπου την ορίζουμε και εδώ default 0.001. Παράλληλα εκτός από τα βάρη και το bias επιστρέφονται και τα σφάλματα ανα εποχή, ο αριθμός των εποχών που χρειάστηκαν, και 2 λίστες με τα τελικά βάρη και bias ανα εποχή.

Αντίστοιχα, καλούμε τη συνάρτηση my_perceptron:

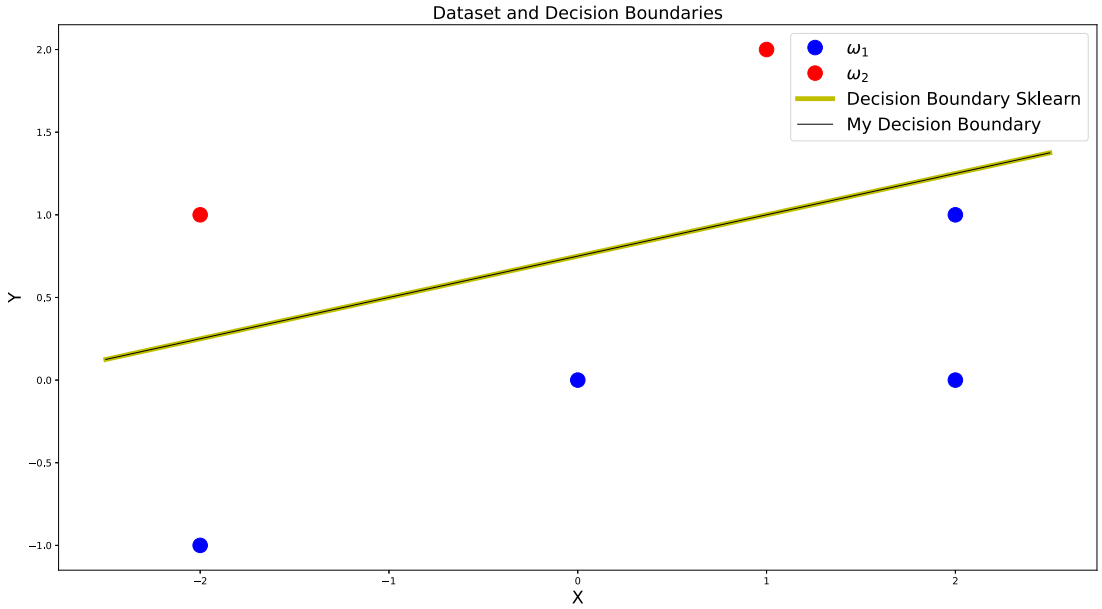
```
errs, weights, bias, iterations, weights_list, bias_list = my_perceptron(X.T, Y)
```

Τα αποτελέσματα σε κάθε περίπτωση παρουσιάζονται στην επόμενη σελίδα.

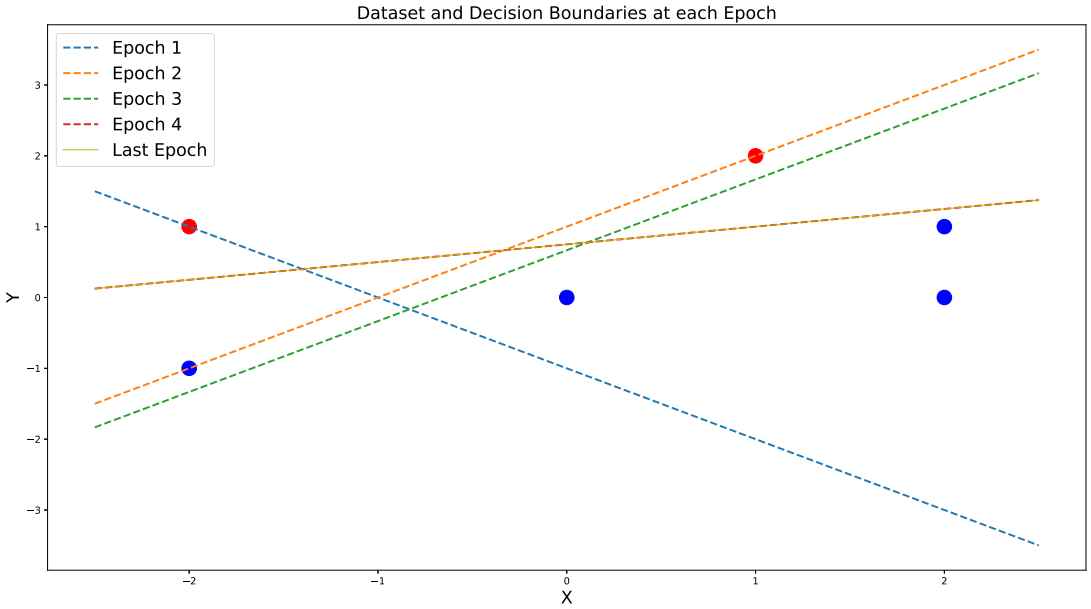
Τα αποτελέσματα είναι τα εξής:

```
-- Epoch 1
Norm: 2.83, NNZs: 2, Bias: 2.000000, T: 6, Avg. loss: 1.333333
Total training time: 0.00 seconds.
-- Epoch 2
Norm: 4.24, NNZs: 2, Bias: 2.000000, T: 12, Avg. loss: 0.166667
Total training time: 0.00 seconds.
-- Epoch 3
Norm: 4.12, NNZs: 2, Bias: 3.000000, T: 18, Avg. loss: 0.166667
Total training time: 0.00 seconds.
-- Epoch 4
Norm: 4.12, NNZs: 2, Bias: 3.000000, T: 24, Avg. loss: 0.000000
Total training time: 0.00 seconds.
Convergence after 4 epochs took 0.00 seconds
------------------------
Sklearn Perceptron Results:
weights:  [1.0, -4.0] and bias 3
Single Perceptron Accuracy 1.0
Number of iterations:  4
------------------------
My Perceptron Results:
weights:  [1.0, -4.0] and bias 3
Number of my iterations  4
Single Perceptron Accuracy:  1
Errors at each epoch:  [5, 2, 1, 0]
```

Αρχικά, παρουσιάζεται ανα εποχή κάποια δεδομένα όσον αφορά την εκπαίδευση του Perceptron από τη βιβλιοθήκη sklearn. Στη συνέχεια παρουσιάζονται αποτελέσματα από το Perceptron από τη βιβλιοθήκη sklearn και τέλος έχουμε τα αποτελέσματα απο τη δική μου υλοποίηση του Perceptron. Όπως φαίνεται τα αποτελέσματα ταυτίζονται. Παρακάτω βλέπουμε και το αντίστοιχο plot όπου οι γραμμές απόφασης ταυτίζονται.



Τέλος, εδώ παρουσιάζονται οι ευθείες απόφασης σε κάθε εποχή.



Προφανώς η ευθείας της τελευταίας εποχής ταυτίζεται με την εποχή 4. Όπως παρατηρούμε, το perceptron καταφαίρνει και συγκλίνει σε μια γραμμή όπου διαχωρίζει τα δεδομένα. Υπενθυμίζεται ότι ο κανόνας ανανέωσης των βαρών και του bias για το perceptron είναι:

$$w_{new} = w_{old} + (TrueLabel - PredictedLabel) \cdot x$$
$$b_{new} = b_{old} + (TrueLabel - PredictedLabel)$$

Δηλαδή, η τιμή του learning_rate είναι 1.

Άσκηση 2

Με βάση εκφώνηση ζητείται να χωρίζουμε το dataset σε train/validation/testing. Επιλέγουμε αυτό να γίνει σε αναλογία 60%/20%/20%. Όπως και στη προηγούμενη θεώρησα καλύτερη γενίκευση να χωρίσουμε το dataset σε train και test (με ίδιο μέγεθος το test) και στο training dataset να κάνουμε K-Folds Cross Validation και να βλέπουμε το average του validation accuracy για την κάθε υλοποίηση έτσι ώστε να βλέπουμε ποιο μοντέλο θα έκανε καλύτερο Generalization. Απομένει, λοιπόν να επιλέξουμε τη κατάλληλη τιμή του K έτσι ώστε να έχουμε το ποσοστό 20% για validation που θέλουμε. Στην εκφώνηση έχουμε 10 δείγματα ανά κάθε κατηγορία άρα 30 συνολικό validation test. Εφόσον το συνόλικό training είναι $0.8 \cdot 150 = 120$ άρα θα έχουμε 120/30=4. Συνεπώς θα επιλεχθεί τιμή K = 4.

Ερώτημα Α

Στο συγκεκριμένο ερώτημα ζητείται να ταξινομηθούν τα δεδομένα με το γραμμικό perceptron. Το νευρωνικό δίκτυο αυτό θα αποτελείται από τρεις νευρώνες (ένας για κάθε κλάση) με συνάρτηση μεταφοράς την βηματική συνάρτηση. Έτσι, ουσιαστικά, ο κάθε νευρώνας «αποφασίζει», εάν το εισερχόμενο pattern είναι ή δεν είναι στην κλάση που «διαχειρίζεται». Στην συγκεκριμένη υλοποίηση ζητείται ακόμη να γίνει αρχικοποίηση των βαρών και των biases με τυχαία συνάρτηση, ώστε να εξασφαλισθεί η ταυτόχρονη σύγκλιση τους στις τελικές τιμές τους. Η υλοποίηση σε αυτή τη περίπτωση θα γίνει μέσω της συνάρτησης Perceptron της sklearn που αναφέρθηκε πιο πάνω.

```
# Train - Test Split
X_train, X_test, y_train, y_test = train_test_split(iris_data, iris_labels, test_size=0.2,
random_state=1, stratify=iris_labels)
K_folds = 4
model = Perceptron(tol=1e-4, max_iter=1000, n_jobs=1)
model.fit(X_train, y_train)
```

Τα αποτελέσματα είναι τα εξής:

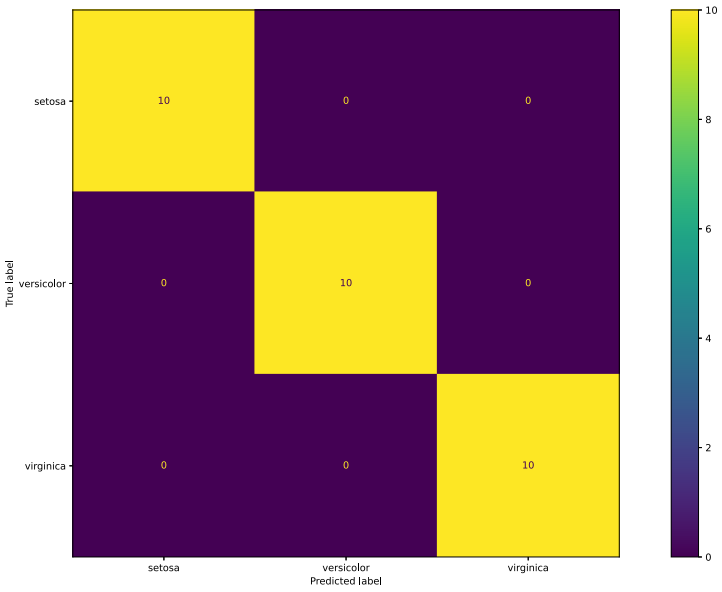
```
Train accuracy: 0.925
Validation Accuracy  : 0.75000 ± 0.12134
Classification Report on Test set:
              precision    recall  f1-score   support

      setosa    1.00000   1.00000   1.00000        10
  versicolor    1.00000   1.00000   1.00000        10
   virginica    1.00000   1.00000   1.00000        10

    accuracy                        1.00000        30
   macro avg    1.00000   1.00000   1.00000        30
weighted avg    1.00000   1.00000   1.00000        30

Model Weights:  [[  3.6   7.9 -13.2  -6.5]
 [  9.6 -29.1  13.7 -19.1]
 [-30.8 -32.2  49.2  46. ]]
Model Biases:  [  2.  10. -19.]
```

Το confusion matrix στη συγκεκριμένη περίπτωση είναι:



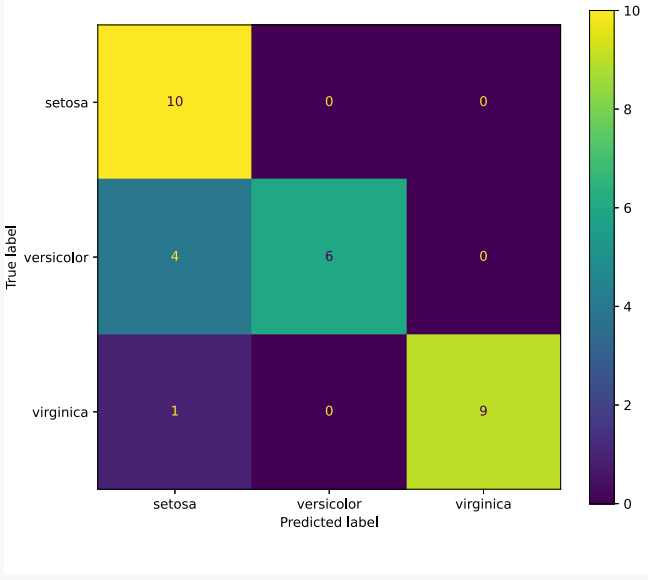
Όπως παρατηρούμε οι 3 κλάσεις διαχωρίζονται τέλεια. Ωστόσο, το αποτέλεσμα αυτό δεν είναι το αναμενόμενο με την πρώτη ματιά καθώς μόνο η κλάση 1 είναι γραμμικά διαχωρίσιμη ενώ οι κλάσεις 2 και 3 δεν είναι. Ενα perceptron δηλαδή για κάθε κλάση όπου είναι γραμμικός ταξινομητής δεν περιμέναμε να μπορεί να διαχωρίσει τα δεδομένα τέλεια. Ωστόσο, όπως παρατηρούμε ο αριθμός των δεδομένων ελέγχου είναι 30 στο σύνολο και 10 για κάθε κλάση. Επειδή ο διαχωρισμός είναι τυχαίος, υπάρχει περίπτωση να επιλεχθούν(όπως και έγινε) για έλεγχο δεδομένα που είναι γραμμικά διαχωρίσιμα. Αν αλλάξουμε το random_state στη συνάρτηση `train_test_split` σύμφωνα με το οποίο γίνεται ο διαχωρισμός των δεδομένων τότε τα αποτελέσματα μπορούν να αλλάξουν. Για παράδειγμα για `random_state=6` αντί για 1 έχουμε:

```
Train accuracy: 0.85
Validation Accuracy  : 0.76667 ± 0.06236
Classification Report on Test set:
              precision    recall  f1-score   support

      setosa    0.66667   1.00000   0.80000        10
  versicolor    1.00000   0.60000   0.75000        10
   virginica    1.00000   0.90000   0.94737        10

    accuracy                        0.83333        30
   macro avg    0.88889   0.83333   0.83246        30
weighted avg    0.88889   0.83333   0.83246        30

Model Weights:
 [[  1.1   4.7  -7.5  -3.9]
 [ 14.6 -39.    9.2 -41.2]
 [-31.8 -38.7  48.2  40.8]]
Model Biases:  [  1.  25. -21.]
```



Στη περίπτωση αυτή, όπως βλέπουμε παρατηρούμε ότι εμφανίζονται σφάλματα στις κλάσεις 2 και 3 ενώ πλέον είτε το precision είτε το recall δεν είναι 1 και το τελικο accuracy είναι στο 83,3%. Γενικά, και στις 2 περιπτώσεις παρατηρούμε ότι η τυπική απόκλιση στο validation_accuracy είναι μεγαλύτερη του 5% οπότε σε τέτοιες περιπτώσεις δε μπορούμε να βγάλουμε πολύ καλά συμπεράσματα.

Ερώτημα Β

Στο συγκεκριμένο ερώτημα να ταξινομηθούν τα δεδομένα με το multi-layer perceptron με ένα hidden layer καιτρεις νευρώνες στην έξοδο (ένας για κάθε κλάση). Ακόμη, ζητείται να χρησιμοποιηθεί ο αλγόριθμος backpropagation και να δοκιμαστούν διαφορετικό αριθμό νευρώνων στο hidden layer (2, 5 και 10 νευρώνες). Η υλοποίηση στη συγκεκριμένη περίπτωση θα γίνει μέσω της συνάρτησης `MLPClassifier` (δηλαδή Multi Layer Perceptron Classifier) της sklearn. Ακόμη, θα δοκιμαστούν οι 3 δυνατές συναρτήσεις ενεργοποίησης `logistic(sigmoid), tanh, relu` ενώ ως optimizer χρησιμοποιείται η SGD(stochastic gradient descent). Ο κώδικας βρίσκεται στο αρχείο `HW7_Exercise2_B.py` . Τα αποτελέσματα είναι τα εξής:

Για random state=1

Neurons in hidden layer	Activation function	Training Accuracy	Validation Accuracy	Test Accuracy
2	logistic	0.9666666666666667	0.966 ± 0.023	1.0
5	logistic	0.975	0.975 ± 0.014	1.0
10	logistic	0.975	0.975 ± 0.014	1.0
20	logistic	0.975	0.975 ± 0.014	1.0
50	logistic	0.9833333333333333	0.983 ± 0.016	1.0
2	tanh	0.9583333333333334	0.95 ± 0.016	1.0
5	tanh	0.9583333333333334	0.95 ± 0.016	1.0
10	tanh	0.975	0.975 ± 0.014	1.0
20	tanh	0.975	0.975 ± 0.014	1.0
50	tanh	0.975	0.975 ± 0.014	1.0
2	relu	0.975	0.966 ± 0.023	1.0
5	relu	0.975	0.958 ± 0.027	1.0
10	relu	0.975	0.966 ± 0.023	1.0
20	relu	0.975	0.958 ± 0.014	1.0
50	relu	0.975	0.975 ± 0.014	1.0

Για random_state = 123

Neurons in hidden layer	Activation function	Training Accuracy	Validation Accuracy	Test Accuracy
2	logistic	0.9916666666666667	0.983 ± 0.016	0.93333
5	logistic	0.9833333333333333	0.983 ± 0.016	0.93333
10	logistic	0.9916666666666667	0.983 ± 0.016	0.96667
20	logistic	0.9916666666666667	0.983 ± 0.016	0.96667
50	logistic	0.9916666666666667	0.983 ± 0.016	0.96667
2	tanh	0.9833333333333333	0.975 ± 0.014	0.9
5	tanh	0.9833333333333333	0.975 ± 0.014	0.9
10	tanh	0.9916666666666667	0.983 ± 0.016	0.93333
20	tanh	0.9916666666666667	0.983 ± 0.016	0.93333
50	tanh	0.9916666666666667	0.983 ± 0.016	0.93333
2	relu	0.9916666666666667	0.983 ± 0.016	0.93333
5	relu	0.9916666666666667	0.975 ± 0.014	0.96667
10	relu	0.9916666666666667	0.983 ± 0.016	0.96667
20	relu	0.9916666666666667	0.975 ± 0.027	0.93333
50	relu	0.9916666666666667	0.983 ± 0.016	0.93333

Όπως παρατηρούμε στη πρώτη περίπτωση έχουμε τέλεια ταξινόμηση στο test set για οποιαδήποτε αριθμό νευρώνων και οποιαδήποτε συνάρτηση ενεργοποίησης αλλά στο validation set έχουμε πάντα πάνω απο 95% και κάποια μικρή διακύμανση. Αντιθετα, στη δεύτερη περίπτωση παρατηρούμε ότι έχουμε σε όλες τις περιπτώσεις ενα μικρό σφάλμα ταξινόμησης στο test set ενώ στο validation set στις περισσότερεςς περιπτώσεις έχουμε 98% μια μία διακύμανση της τάξης 1.5%.

Για random_state = 100

Neurons in hidden layer	Activation function	Training Accuracy	Validation Accuracy	Test Accuracy
2	logistic	0.975	0.975 ± 0.043	0.96667
5	logistic	0.975	0.975 ± 0.043	1.0
10	logistic	0.975	0.975 ± 0.043	1.0
20	logistic	0.975	0.975 ± 0.043	1.0
50	logistic	0.975	0.975 ± 0.043	1.0
2	tanh	0.9666666666666667	0.958 ± 0.036	0.96667
5	tanh	0.9666666666666667	0.95 ± 0.037	0.96667
10	tanh	0.975	0.958 ± 0.043	1.0
20	tanh	0.975	0.958 ± 0.043	1.0
50	tanh	0.9833333333333333	0.966 ± 0.040	1.0
2	relu	0.9833333333333333	0.975 ± 0.043	0.96667
5	relu	0.9833333333333333	0.966 ± 0.040	0.96667
10	relu	0.9833333333333333	0.966 ± 0.040	1.0
20	relu	0.975	0.958 ± 0.043	1.0
50	relu	0.975	0.966 ± 0.040	1.0

Τέλος,για ένα ακόμη πιθανό random_state δηλαδή τρόπο χωρισμού των δεδομένων, βλέπουμε ότι για τα αποτελέσματα είναι βέλτιστα για 10 νευρώνες τουλάχιστον στο κρυφό στρώμα. Ωστόσο, υπενθυμιζεται και πάλι οτι αυτά τα αποτελέσματα έχουν περιέχουν κάποια αβεβαιότητα. Τέλος, μια ακόμη σημαντική παράμετρος είναι το πόσο αποδοτική είναιγ για παράδειγμα κάθε συνάρτηση ενεργοποίησης όσον αφορά τις απαιτήσεις σε υπολογιστική ισχύ. Κάνοντας μέτρηση του χρόνου εκτέλεσης που απαιτήτει για την εκπαίδευση σε κάθε περίπτωση έχουμε:

```
Training 5 models using  logistic  as activation function lasted  17.972596168518066  seconds.
Training 5 models using  tanh  as activation function lasted  10.23998498916626  seconds.
Training 5 models using  relu  as activation function lasted  5.288649320602417  seconds.
```

Αυτά τα αποτελέσματα δεν είναι απόλυτα αλλά σίγουρα μας δίνουν μια εκτίμηση. Αν κάνουμε πολλές δοκιμές θα δούμε ότι για την εκπαίδευση 5 μοντέλων με τους συγκεκριμένους αριθμούς νευρώνων στο κρυφό στρώμα τότε:

- Η χρήση της logistic(sigmoid) function ως συνάρτηση ενεργοποίησης είναι η πιο απαιτητική υπολογιστικά.
- Η συνάρτηση tanh ακολουθεί την logistic αλλά είναι πιο αργή από την relu.
- Η χρήση της relu έχει τα καλύτερα αποτελέσματα.

Παρά την απλότητα της η relu είναι πολύ αποδοτική υπολογιστικά καθώς επίσης έχει και πολύ καλά αποτελέσματα όσον αφορά την απόδοση σε ακρίβεια πράγμα που την καθιστά μια από τις καταλληλότερες και συνηθέστερες επιλογές στις μέρες μας.

Τέλος, έγιναν δοκιμές με διαφορετικές τιμές learning rate (πχ 0.0001,0.005,0.001,0.1) εκτός του 0.01 που χρησιμοποιήθηκε μέχρι τώρα. Στις περιπτώσεις όπου το learning rate ήταν μικρότερο του 0.005 οι χρόνοι εκπαίδευσης άρχισαν να αυξάνονται πολύ ενώ στη περίπτωση του 0.0001 οι 10000 επαναλήψεις δεν ήταν αρκετές για να φτάσει σε σύγκλιση με τα ίδια καλά αποτελέσματα σε σχέση με χρήση του 0.01. Από την άλλη η χρήση του 0.1 παρά το γεγονός ότι η σύγκλιση ερχόταν πιο γρήγορη, πολλές φορές είχαμε περιπτώσεις αστάθιας ενώ παράλληλα δεν μπορούσε ο αλγόριθμος να συγκλίνει σε καλό τοπικό ελάχιστο.

Ερώτημα Γ

Στη περίπτωση αυτή επαναλαμβάνουμε πειράματα αλλά με χρήση 2 ενδιάμεσων/κρυφών επιπέδων. Τα αποτελέσματα είναι τα εξής:

Για random_state = 1

Neurons in first hidden layer	Neurons in second hidden layer	Activation function	Training Accuracy	Validation Accuracy	Test Accuracy
2	2	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
2	5	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
2	10	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
2	20	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
2	50	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
5	2	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
5	5	logistic	0.3416666666666667	0.35 ± 0.028	0.33333
5	10	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
5	20	logistic	0.425	0.433 ± 0.084	0.4
5	50	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
10	2	logistic	0.31666666666666665	0.324 ± 0.036	0.4
10	5	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
10	10	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
10	20	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
10	50	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
20	2	logistic	0.30833333333333335	0.316 ± 0.016	0.23333
20	5	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
20	10	logistic	0.9833333333333333	0.975 ± 0.027	1.0
20	20	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
20	50	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
50	2	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
50	5	logistic	0.9833333333333333	0.983 ± 0.016	1.0
50	10	logistic	0.9833333333333333	0.983 ± 0.016	1.0
50	20	logistic	0.3333333333333333	0.333 ± 0.0	0.33333
50	50	logistic	0.9833333333333333	0.983 ± 0.016	1.0
2	2	tanh	0.39166666666666666	0.4 ± 0.040	0.4
2	5	tanh	0.975	0.975 ± 0.014	1.0
2	10	tanh	0.6583333333333333	0.583 ± 0.237	0.66667
2	20	tanh	0.3333333333333333	0.333 ± 0.0	0.33333
2	50	tanh	0.6583333333333333	0.741 ± 0.132	0.66667
5	2	tanh	0.975	0.966 ± 0.023	1.0
5	5	tanh	0.49166666666666664	0.875 ± 0.121	0.5
5	10	tanh	0.6416666666666667	0.858 ± 0.132	0.53333
5	20	tanh	0.3333333333333333	0.333 ± 0.0	0.33333
5	50	tanh	0.9583333333333334	0.808 ± 0.160	1.0
10	2	tanh	0.975	0.975 ± 0.014	1.0
10	5	tanh	0.9833333333333333	0.966 ± 0.023	1.0
10	10	tanh	0.975	0.966 ± 0.023	1.0
10	20	tanh	0.975	0.975 ± 0.014	1.0
10	50	tanh	0.975	0.966 ± 0.023	1.0
20	2	tanh	0.9833333333333333	0.966 ± 0.023	1.0
20	5	tanh	0.975	0.966 ± 0.023	1.0
20	10	tanh	0.975	0.966 ± 0.023	1.0
20	20	tanh	0.975	0.966 ± 0.023	1.0
20	50	tanh	0.975	0.966 ± 0.023	1.0
50	2	tanh	0.925	0.833 ± 0.102	0.9
50	5	tanh	0.9916666666666667	0.975 ± 0.014	1.0
50	10	tanh	0.9916666666666667	0.975 ± 0.014	1.0
50	20	tanh	0.9833333333333333	0.975 ± 0.014	1.0
50	50	tanh	0.9916666666666667	0.966 ± 0.023	1.0
2	2	relu	0.9583333333333334	0.958 ± 0.014	0.96667
2	5	relu	0.9833333333333333	0.958 ± 0.014	1.0
2	10	relu	0.9916666666666667	0.966 ± 0.0	1.0
2	20	relu	0.975	0.975 ± 0.027	1.0
2	50	relu	0.975	0.966 ± 0.023	1.0
5	2	relu	0.3333333333333333	0.333 ± 0.0	0.33333
5	5	relu	0.65	0.649 ± 0.016	0.66667
5	10	relu	0.3333333333333333	0.333 ± 0.0	0.33333
5	20	relu	0.975	0.966 ± 0.023	1.0
5	50	relu	0.975	0.966 ± 0.023	1.0
10	2	relu	0.3333333333333333	0.333 ± 0.0	0.33333
10	5	relu	0.9833333333333333	0.975 ± 0.014	1.0
10	10	relu	0.9833333333333333	0.975 ± 0.014	1.0
10	20	relu	0.9833333333333333	0.975 ± 0.014	1.0
10	50	relu	0.9833333333333333	0.975 ± 0.014	1.0
20	2	relu	0.975	0.975 ± 0.014	1.0
20	5	relu	0.975	0.975 ± 0.014	1.0
20	10	relu	0.975	0.95 ± 0.016	1.0
20	20	relu	0.975	0.95 ± 0.016	1.0
20	50	relu	0.975	0.941 ± 0.014	1.0
50	2	relu	0.9666666666666667	0.958 ± 0.014	0.96667
50	5	relu	0.975	0.975 ± 0.014	1.0
50	10	relu	0.975	0.958 ± 0.027	1.0
50	20	relu	0.975	0.975 ± 0.014	1.0
50	50	relu	0.975	0.966 ± 0.023	1.0

Αυτή τη φορά παρατηρούμε ότι για να φτάσουμε σε εξίσου καλά αποτελέσματα πρέπει να έχουμε αρκετούς νευρώνες ανά layer καθώς με 2 δεν έχουμε τόσο καλό αποτέλεσμα συγκριτικά με το 1 hidden layer(πέραν της περίπτωσης που έχουμε relu). Ακόμη, παρατηρούμε ότι με τη χρήση της logistic έχουμε σημαντικές διακυμάνσεις στα αποτελέσματα πράγμα που δηλώνει ότι δεν είναι ιδιαίτερα κατάλληλη για τις συγκεριμένες αρχιτεκτονικές. Τέλος υπάρχει μεγαλύτερη διαφοροποίηση μεταξύ activation functions σε σχέση με το 1 hidden layer. Αυτά τα αποτελέσματα δεν μπορούμε βέβαια να πούμε ότι μπορούμε να τα γενικέυσουμε πάντα σχετικά με την αρχιτεκτονική και τις άλλες υπερπαραμέτρους του Multi-Layer perceptron καθώς αυτό εξαρτάται από το είδος και το μέγεθος του dataset που έχουμε. Τέλος, μπορούμε να πούμε ότι η προσθήκη 2ου επιπέδου δεν είναι απαραίτητα αναγκαία καθώς το συγκεκριμένο πρόβλημα μπορούσε να λυθεί και με ένα κρυφό επίπεδο με βέλτιστη ακρίβεια. Ετσι, κάνουμε εξοικονόμηση σε υπολογιστική ισχύ καθώς και σε απαιτήσεις μνήμης. Το συμπέρασμα είναι λοιπόν πως η επιλογή των υπερπαραμέτρων όπως για παράδειγμα ο αριθμός των νευρώνων σε ένα κρυφό στρώμα, ο αριθμός των κρυφών στρωμάτων, η συνάρτηση ενεργοποίησης που θα χρησιμοποιηθεί, το learning rate, ο optimizer και πολλά άλλα δεν μπορούν να είναι στάνταρ αλλά απαιτούν ερεύνα και δοκιμές για κάθε πρόβλημα. Πλεόν υπάρχουν και τεχνικές οι οποίες βοηθάνε ακριβώς σε αυτό, δηλαδή συμβάλουν στο hyperparameter tuning.