

Assignment 1

by Bowen Chen

Submission date: 30-Mar-2023 11:50PM (UTC+0100)

Submission ID: 201097591

File name: 41332_Bowen_Chen_Assignment__1_495066_1702790198.pdf (753.38K)

Word count: 5034

Character count: 28024

Classification using ML methods and Neural Networks

Bowen Chen

Student number:2200373

Abstract: This assignment revolves around two algorithms, namely, the Artificial Neural Networks (ANN) and Multiclass Logistic Regression. Both algorithms will be used to train the Fashion MNIST and CIFAR-10 datasets. The aim of the assignment is to adjusting various hyperparameters and then compare the performance of both algorithms. By analyzing the results obtained, we will have a better understanding of the performance and characteristics of each algorithm and how they can be optimized for specific datasets.

Key words: Neural Networks; Multi-class Logistic Regression; Fashion MNIST; CIFAR-10 datasets;

Introduction:

Multi-class classification is a fundamental problem in machine learning. The goal is to predict the correct label among a set of predefined classes for a given input sample. In recent years, several algorithms have been used to deal with this problem, and in this assignment Artificial Neural Networks (ANN) and Logistic Regression are used to do classification. These algorithms have been widely used in various aspects, including image classification, natural language processing, and speech recognition, and so many others^[1].

In this assignment, I will apply the ANN and Logistic Regression algorithms to two popular benchmark datasets in the image classification community. The first dataset is the Fashion MNIST dataset, which consists of 60,000 training images and 10,000 test images of 10 different fashion items, such as T-shirts, dresses, and shoes, among others. The images are grayscale and have a resolution of 28x28 pixels. The dataset was created as a replacement for the traditional MNIST dataset, which had become too easy for modern algorithms.

The second dataset is the CIFAR-10 dataset, which consists of 50,000 training images and 10,000 test images of 10 different classes of objects, such as airplanes, cars, and animals, among others. The images are RGB and have a resolution of 32x32 pixels. The dataset is considered challenging due to the small image size and the high variability within classes.

My objective is to train and optimize both algorithms on these datasets to achieve high accuracy in multi-class classification tasks. I will explore the effect of tuning various hyperparameters on the performance of these algorithms.

1.About the Datasets

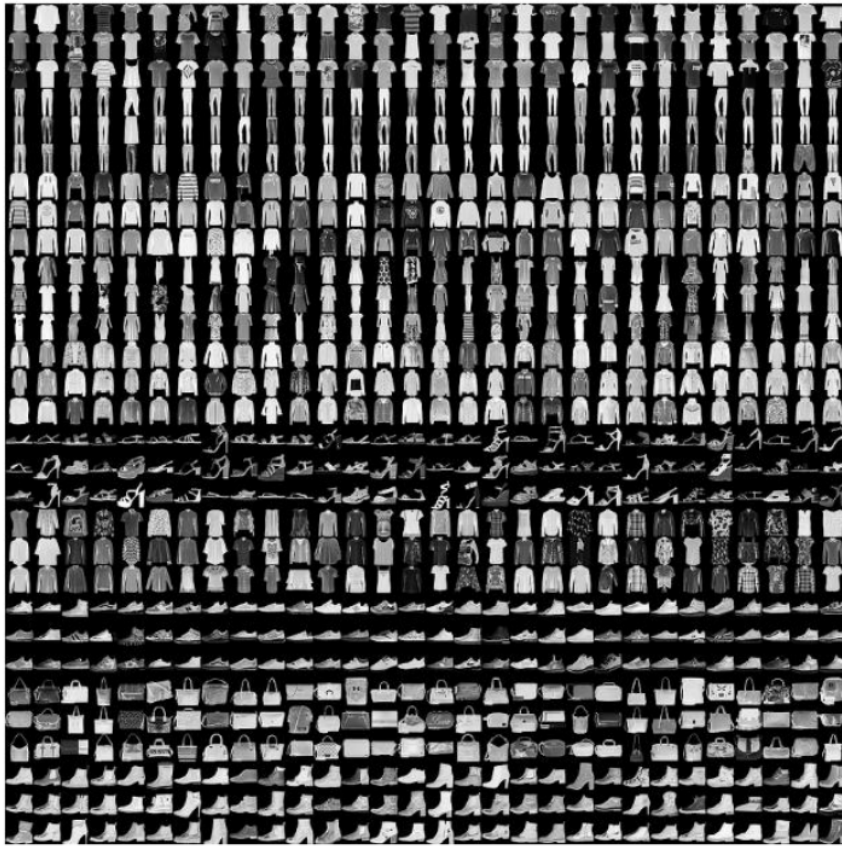
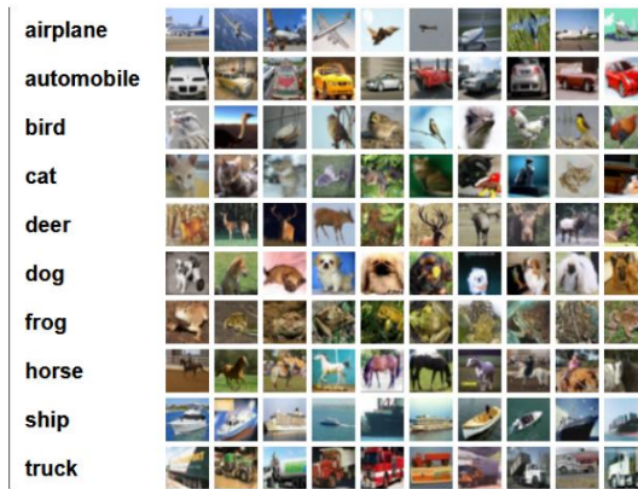


Fig1 . Fashion MNIST dataset

The Fashion MNIST dataset is a widely used benchmark dataset in the field of machine learning for image classification tasks. It consists of a collection of 70,000 grayscale images, including 60,000 training images and 10,000 test images. The images represent 10 different fashion items, including T-shirts, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots. Each image is of size 28x28 pixels and contains a single item centered in the image^[2].



variables can provide insights into the underlying structure of the data. This can be useful for identifying hidden trends, clusters, or correlations between variables. One limitation of PCA is that it assumes a linear relationship between the variables. If the relationship between the variables is nonlinear, then PCA may not capture all of the important information in the data. Additionally, PCA can be sensitive to outliers, which can skew the results. In summary, PCA can help to simplify complex data sets, identify patterns and relationships between variables, and make the data easier to visualize and analyze.

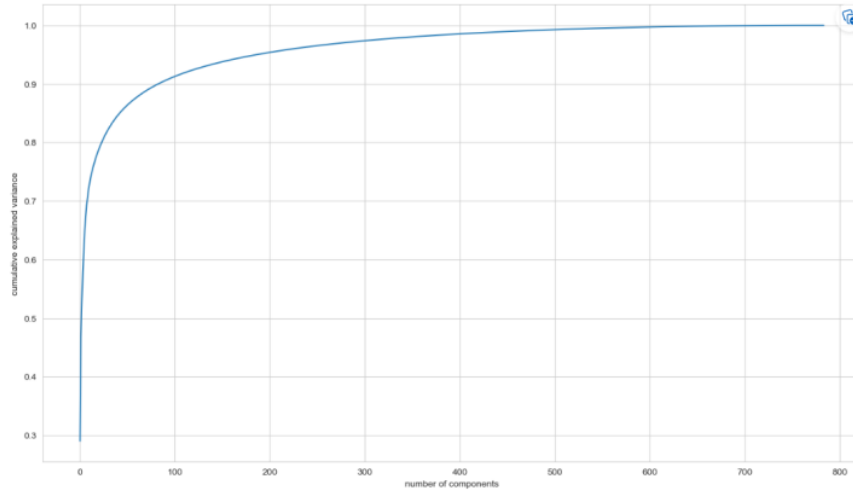


Fig3. PCA with Fashion MNIST datasets

The above figure shows PCA with Fashion mnist datasets. The number of components is plotted on the X-axis, while the cumulative explained variance is plotted on the y-axis.

In Scikit-learn^[4], `explained_variance_ratio_` is an array containing the proportion of explained data variance for each principal component. In PCA, the variance of each principal component is sorted, and the `explained_variance_ratio_` array can be used to determine the relative importance of each principal component. Specifically, each element in `explained_variance_ratio_` is a number between 0 and 1 that represents the proportion of the total variance explained by that principal component compared to the total variance explained by all the principal components. For example, if the `explained_variance_ratio_` for the first principal component is 0.6, it means that the total variance explained by the first principal component is 60% of the total variance explained by all the principal components. By examining the `explained_variance_ratio_` array, it is possible to determine how many principal components need to be retained in order to preserve enough variance. Typically, retaining a cumulative sum of explained variance ratios that reaches 80% to 90% is enough to capture most of the variability in the data.

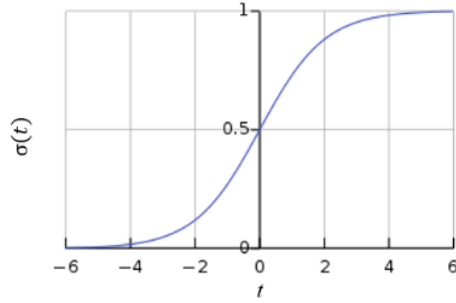
3. Multiclass Logistic Regression algorithm

logistic regression is a machine learning algorithm used for classification tasks with more than two classes. It extends binary logistic regression by using a softmax function to calculate the probabilities of each class, given a set of input features. In two class classification, the final output is the type with probability bigger than 0.5.

For example, inputs have 5 different features.

X is the input features , $X = [x_1 \ x_2 \ x_3 \ x_4 \ x_5]$

And the coefficient of X $\theta^T = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \end{bmatrix}$



The sigmod logistic function is given as:

$$\sigma_t = \frac{1}{1 + e^{-t}}$$

And finally, the probability is calculated as:

$$\hat{p} = \sigma(\theta^T X) = \frac{1}{1 + e^{-(\theta^T X)}}$$

In multiclass classification with logistic regression, the probability is calculated as

$$p(y_i = s) = \frac{e^{\beta_s X_i}}{\sum_{k=1}^K e^{\beta_k X_i}}$$

Where s is a given class, β_s is the coefficient matrix of the class's input features. And X represents the matrix of input features. This function is called "softmax". Finally, the prediction is the class type with the highest probability.

4. Multilayer Perceptron

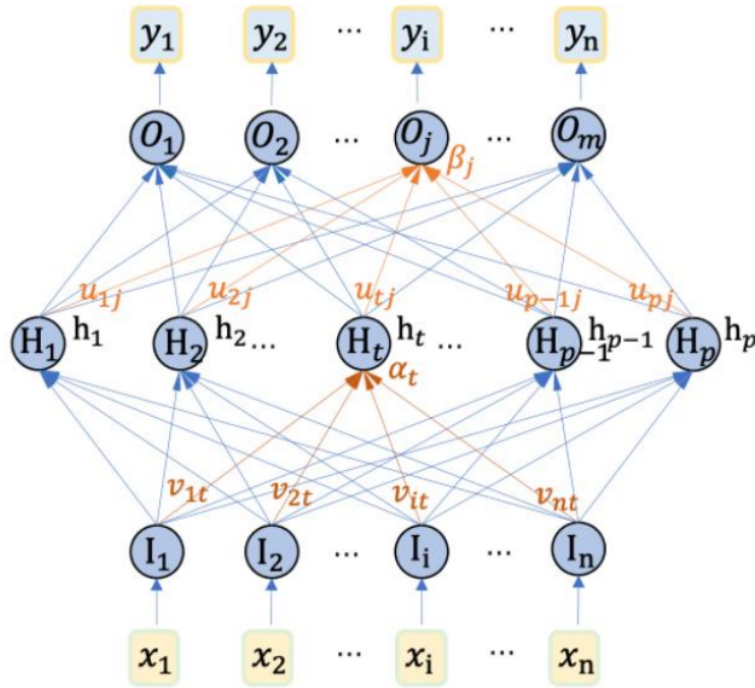


Fig4. Fully connected neural networks

Multilayer perceptron (MLP), which is also called fully-connected networks, is a basic type of artificial neural network. In a MLP neural network, each neuron in a given layer is connected to all neurons in the preceding layer, with each connection weighted by a value that is learned during training. The model typically consists of an input layer, one or more hidden layers, and an output layer, with each layer composed of multiple neurons^[5]. The neurons in the hidden and output layers apply a nonlinear activation function, such as ReLU or sigmoid, to produce a nonlinear mapping relationship.

MLP are widely used for classification, regression, and prediction tasks, and can be trained and optimized using backpropagation algorithms to minimize the difference between predicted values and actual values. Although the performance of fully connected neural networks has been surpassed by other types of neural networks, such as convolutional neural networks and recurrent neural networks, it remains a widely studied and applied neural network model due to its simplicity and versatility.

5 Experiment

5.1 Multinomial logistic regression classification on the Fashion MNIST dataset.

5.1.1 Train the dataset

```
In [5]: # Variance ratio with 100 components:
pca.explained_variance_ratio_[:100].sum()

# To saving time when training and predicting,
# we can use data set after PCA with 100 components which have presented ~90.1% of the total variance, for training model.

Out[5]: 0.9121840780529507
```

First, load the dataset, normalize the input data using MinMaxScaler. Then reduce dimensionality using PCA method. I keep the top 100 components which have presented 90.1% of the total variance to reduce the training time.

Second, use GridSearchCV to tune hyperparameters, and get the best one between listed values.

```

:
from sklearn.model_selection import GridSearchCV

# Define the hyperparameters to search over
param_grid = {
    'C': [0.1, 1, 10],
    'penalty': ['l1', 'l2'],
    'solver': ['lbfgs', 'liblinear', 'sag', 'saga']
}

# Define the grid search
grid_search = GridSearchCV(estimator=lr, param_grid=param_grid, cv=5)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters and the corresponding accuracy score
print("Best hyperparameters:", grid_search.best_params_)

C:\Users\74648\Anaconda\envs\tensorflow\lib\site-packages\sklearn\model_selection\_search.py:952: UserWarning: One or more of the test scores
are non-finite: [      nan      nan      nan      nan      nan      nan      nan
 0.84705  0.84711667      nan      nan      nan      nan      nan
 0.84773333      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan      nan]
warnings.warn(

Best hyperparameters: {'C': 1, 'penalty': 'l2', 'solver': 'sag'}

```

GridSearchCV^[6] stands for "Grid Search Cross-Validation". It is a technique used in machine learning to tune hyperparameters for a given model. The technique involves defining a grid of hyperparameter values to explore, and then evaluating each combination of hyperparameters using cross-validation, in order to identify the set of hyperparameters that produces the best results. GridSearchCV is commonly used in conjunction with k-fold cross-validation, where the data is split into k equal parts, and each part is used as the test set exactly once, with the remaining k-1 parts used for training. This allows the performance of the model to be evaluated in a more robust way, and helps to prevent overfitting.

The best hyperparameter I found is :

solver: sag;
Regularization Parameter : 1
Penalty: "l2"

5.2.2 Hyperparameter analysis.

Regularization methods, regularization strength, and optimizers can all have an impact on the results.

(1)Regularization methods: Common regularization methods include L1 regularization and L2 regularization. The goal of these methods is to prevent the model from overfitting the training data. L1 regularization tends to produce sparse solutions, where some feature coefficients are zero, thereby increasing the model's interpretability. On the other hand, L2 regularization makes feature coefficients relatively smooth, thereby increasing the model's generalization ability.

(2)Regularization strength: Regularization strength controls the degree to which the regularization term affects the model. If the regularization strength is high, the regularization term will have a

larger impact, and the model may tend to use fewer features for prediction, resulting in a simpler model. Conversely, if the regularization strength is low, the model may overfit the training data.

(3)Optimizers: Optimizers determine how the model parameters are updated during the training process. Common optimizers include stochastic gradient descent (SGD), Adam, Adagrad, and others. The optimizer selection can affect the convergence speed and stability of the model. Different optimizers may have different effects on different datasets and model architectures.

Therefore, when performing multinomial logistic regression classification, it is necessary to consider the selection of regularization methods, regularization strength, and optimizers to obtain better model performance. I list several values and used GridSearchCV and cross-validation method to try the best hyperparameters.

5.2.3 Confusion matrix.

The confusion matrix is shown below. The overall prediction accuracy on the test set is 85.57%.

accuracy on test set: 0.8557				
	precision	recall	f1-score	support
0	0.80	0.82	0.81	1000
1	0.96	0.97	0.97	1000
2	0.77	0.76	0.76	1000
3	0.85	0.88	0.87	1000
4	0.78	0.80	0.79	1000
5	0.94	0.92	0.93	1000
6	0.66	0.60	0.63	1000
7	0.91	0.92	0.91	1000
8	0.94	0.94	0.94	1000
9	0.93	0.95	0.94	1000
accuracy			0.86	10000
macro avg	0.85	0.86	0.85	10000
weighted avg	0.85	0.86	0.85	10000

These indicators are calculated as:

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN}$$

$$recall = \frac{TP}{TP + FN}$$

$$precision = \frac{TP}{TP + FP}$$

$$f1-score = \frac{2 * precision * recall}{(precision + recall)}$$

Where TP represents true positive, and FP is false positive; TN is true negative, FN is false negative.

- (1) Accuracy: Accuracy stands for the proportion of how many samples are correctly classified among all the predictions^[7].
- (2) Precision: It measures the cost of classifying the majority class as the minority class. The lower the Precision, the more majority class instances are classified as the minority class.^[7].
- (3) Recall: Recall measures the ability of a classifier to recognize positive samples. If the recall is high, it indicates that the classifier can identify positive samples well, minimizing the possibility of missed diagnoses. Conversely, if the recall is low, it indicates poor recognition ability of positive samples by the classifier, which leads to missed diagnoses^[7].
- (4) F1 score: The significance lies in its ability to comprehensively evaluate the accuracy and recall capability of a classifier, and its general applicability to different datasets and classification tasks. Compared to evaluation metrics that only consider precision or recall, F1 score can provide a more comprehensive assessment of classifier performance^[7].

5.2 MLP classification on the Fashion MNIST dataset.

First, Split the train dataset into a train set and a validation set. There are 5000 instances in the validation set and the others are in the train set. I use PCA to reduce dimensionality as usual, in order to speed up the training process. I keep the top 100 components which have presented 90.1% of the total variance.

Next, build the fully connected neural networks . Flattening the input vector from 28*28 to 1*768. My networks have 3 layers. The number of neurons in the input layer is determined by the number of features in the input data, while the number of neurons in the output layer depends on the task to be solved. And in Fashion mnist classification task, the output is 10 class. And I have 2

hidden layers, both have the non-linear “relu” activation function. It can prevent the problem of gradient vanishing: in a multi-layer neural network, using activation functions such as sigmoid or tanh is prone to the problem of gradient vanishing, that is, during the backpropagation process, the gradient will gradually become smaller, making it difficult for the network to converge. It also has fast computation speed and can better fit non-linear models.

Model: "sequential"		
Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 200)	157000
dense_1 (Dense)	(None, 50)	10050
dense_2 (Dense)	(None, 10)	510
Total params: 167,560		
Trainable params: 167,560		
Non-trainable params: 0		

The weights and biases of the network are learned during training using gradient descent. The optimizer I choose is adam. Adam (Adaptive Moment Estimation) is an optimization algorithm with adaptive learning rate, which combines the advantages of Momentum while requiring less memory. Adam calculates the first and second moments of the current gradient at each iteration and uses them to update the parameters. Specifically, for each parameter update, Adam considers the exponentially weighted moving average of both the current and past gradients, as well as the exponentially weighted moving average of both the current and past squared gradients, to adjust the learning rate. The advantages of the Adam optimizer include good convergence and generalization properties, and it is applicable to training most neural networks.

```
model.compile(loss='sparse_categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), metrics=['accuracy'])
```

```
history = model.fit(X_train, y_train, epochs=10, batch_size = 32, validation_data=(X_valid, y_valid))
```

The batch size is selected as 32, epochs choose 10 in order to speed up the training process as well as avoid overfitting. Too many training epochs can lead to overfitting because the model memorizes the noise in the training data rather than the true patterns or rules.

```
history = model.fit(X_train, y_train, epochs=10, batch_size = 32, validation_data=(X_valid, y_valid))
```

```
Epoch 1/10
1719/1719 [=====] - 7s 4ms/step - loss: 0.5039 - accuracy: 0.8205 - val_loss: 0.3960 - val_accuracy: 0.8584
Epoch 2/10
1719/1719 [=====] - 6s 3ms/step - loss: 0.3711 - accuracy: 0.8657 - val_loss: 0.3408 - val_accuracy: 0.8764
Epoch 3/10
1719/1719 [=====] - 7s 4ms/step - loss: 0.3329 - accuracy: 0.8769 - val_loss: 0.3214 - val_accuracy: 0.8808
Epoch 4/10
1719/1719 [=====] - 6s 3ms/step - loss: 0.3112 - accuracy: 0.8858 - val_loss: 0.3380 - val_accuracy: 0.8804
Epoch 5/10
1719/1719 [=====] - 7s 4ms/step - loss: 0.2957 - accuracy: 0.8899 - val_loss: 0.3121 - val_accuracy: 0.8886
Epoch 6/10
1719/1719 [=====] - 5s 3ms/step - loss: 0.2772 - accuracy: 0.8961 - val_loss: 0.3046 - val_accuracy: 0.8928
Epoch 7/10
1719/1719 [=====] - 8s 5ms/step - loss: 0.2665 - accuracy: 0.8989 - val_loss: 0.3105 - val_accuracy: 0.8896
Epoch 8/10
1719/1719 [=====] - 6s 4ms/step - loss: 0.2529 - accuracy: 0.9050 - val_loss: 0.3379 - val_accuracy: 0.8816
Epoch 9/10
1719/1719 [=====] - 6s 4ms/step - loss: 0.2457 - accuracy: 0.9083 - val_loss: 0.2954 - val_accuracy: 0.8942
Epoch 10/10
1719/1719 [=====] - 7s 4ms/step - loss: 0.2366 - accuracy: 0.9107 - val_loss: 0.3096 - val_accuracy: 0.8902
```

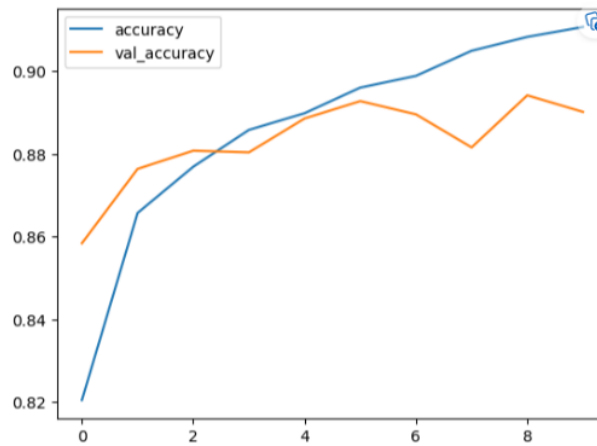


Fig5 accuracy on train and valid set during each epoch

When train the model , I record the results and play a graph to show the performance on the accuracy with train and validation set during every epochs. As we can see, there may be a little overfitting, but not very obvious.

```

313/313 [=====] - 1s 3ms/step
              precision    recall  f1-score   support

0               0.85        0.84        0.85        1000
1               0.92        0.98        0.95        1000
2               0.82        0.73        0.77        1000
3               0.94        0.77        0.85        1000
4               0.70        0.87        0.77        1000
5               0.99        0.93        0.96        1000
6               0.72        0.68        0.70        1000
7               0.94        0.95        0.95        1000
8               0.96        0.98        0.97        1000
9               0.92        0.97        0.94        1000

 accuracy                    0.87        10000
 macro avg                   0.87        0.87        0.87        10000
 weighted avg                 0.87        0.87        0.87        10000

```

As can be seen above, The prediction accuracy on the test set is 87%, the performance of the neural network I build is acceptable, although it is a little bit lower than that on the train set. The reason is that the model memorizes all the details and noise in the training set, which do not exist in the test set, resulting in the model performing worse on the test set than on the training set. To address this issue, some techniques can be used to alleviate overfitting, such as regularization, early stopping, and data augmentation. In addition, techniques such as cross-validation can be used to evaluate the performance of the model and select the best performing model for deployment. and data augmentation. What I use is the regularization method.

5.3 Switch to the Cifar10 dataset and retrain the model.

The results are shown below:

Dataset	type	Algorithm	
		Logistic Regression	MLP
Fashion mnist	train	0.847	0.9
	test	0.855	0.87
Cifar10	train	0.39	0.87
	test	0.4	0.51

Fig6 Accuracy on both train and test of the two datasets

We can see from the result that there is much overfitting when train the neural networks with Cifar10 dataset, and the performance is much lower compared with the Fashion MNIST dataset. The same algorithm performs differently on different datasets due to differences in dataset characteristics and complexity. Although Fashion MNIST and CIFAR-10 are both image classification datasets, they differ in the following ways:

- (1) Different image resolutions: Fashion MNIST has a resolution of 28 x 28, while CIFAR-10 has a resolution of 32 x 32.
- (2) Different image categories and quantities: Fashion MNIST contains 10 categories, while CIFAR-10 contains 10 more complex categories.
- (3) Different noise and variation in the dataset: CIFAR-10 contains more noise and variation in the dataset, such as different lighting conditions, angles, scales, etc.

The reason why fully connected neural networks perform poorly on CIFAR-10 is because CIFAR-10 is a complex image classification dataset containing images from 10 categories. Compared to the Fashion MNIST dataset, CIFAR-10 has higher resolution and more noisy images, which makes fully connected neural networks perform poorly on CIFAR-10. Fully connected neural networks require the input image to be flattened into a one-dimensional vector, which means it does not take into account the spatial structure and local correlation of the image. Therefore, fully connected neural networks may experience overfitting and underfitting when dealing with complex image classification problems.

In contrast, Convolutional Neural Networks (CNN)  can capture the spatial structure and local correlation of the image. CNN uses convolutional and pooling layers to extract image features, and uses fully connected layers for classification. CNN typically performs better in image classification problems, especially for complex datasets like CIFAR-10. Therefore, to achieve better performance on CIFAR-10, it is usually necessary to use convolutional neural networks or other neural network models based on CNN architecture.

6. Conclusion

Based on the results of my experiments, we can conclude that both Multinomial Logistic Regression and Fully Connected Neural Networks are not suitable for training the CIFAR-10 dataset. This is due to the fact that CIFAR-10 is a complex image classification dataset with high-resolution images and more noise compared to other datasets such as MNIST. These algorithms may experience overfitting and underfitting problems and are not capable of capturing the spatial structure and local correlation of the images. As a result, to achieve better performance on CIFAR-10, it is necessary to use more advanced neural network models such as Convolutional Neural

Networks (CNN) or other models that are designed specifically for image classification tasks. In conclusion, recommend the use of CNN or other appropriate models for training CIFAR-10 dataset to achieve better performance.

References

- [1] Gartstein, M. A. , Seamon, D. E. , Mattera, J. A. , Enlow, M. B. , Wright, R. J. , & Perez-Edgar, K. , et al. (2022). Using machine learning to understand age and gender classification based on infant temperament. PloS one, 17(4), e0266026.
- [2] Xiao, H., Rasul, K. , & Vollgraf, R. . (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.
- [3] Krizhevsky, A., & Hinton, G. . (2009). Learning multiple layers of features from tiny images. Handbook of Systemic Autoimmune Diseases, 1(4).
- [4] <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>
- [5] https://blog.csdn.net/weixin_44052055/article/details
- [6] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
- [7] <https://www.sciencedirect.com/topics/engineering/confusion-matrix>

Appendix

1. Fashion MNIST using ML method

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)

import warnings
from sklearn.exceptions import ConvergenceWarning
warnings.filterwarnings(action='ignore', category=ConvergenceWarning)

import warnings
from sklearn.exceptions import FitFailedWarning
warnings.filterwarnings('ignore', category=FitFailedWarning)

df_train=pd.read_csv('G:/data_csv/fashion-mnist_train.csv')
df_test=pd.read_csv('G:/data_csv/fashion-mnist_test.csv')

X_train=df_train.iloc[:,1:]
y_train=df_train.iloc[:,0]
X_test=df_test.iloc[:,1:]
y_test=df_test.iloc[:,0]

# Nomalize data by using MinmaxScaler
from sklearn.preprocessing import MinMaxScaler

X_train = MinMaxScaler().fit_transform(X_train)
X_test = MinMaxScaler().fit_transform(X_test)

# Check min_max after normalize:
(X_train.min(), X_train.max()),(X_test.min(), X_test.max())

from sklearn.decomposition import PCA
```

```

# Plot PCA
sns.set_style('whitegrid')

plt.figure(figsize=(16, 9))
pca = PCA().fit(X_train)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')

pca.explained_variance_ratio_[100].sum()

def pca_set(X_train, X_test, n_components):
    pca_used = PCA(n_components=n_components)
    # Transform data set:
    X_train_pca = pca_used.fit_transform(X_train)
    X_test_pca = pca_used.transform(X_test)
    print(f'DONE === with n_components: {pca_used.n_components_}')
    return X_train_pca, X_test_pca

X_train, X_test = pca_set(X_train, X_test, 100)

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import accuracy_score, classification_report

from sklearn.model_selection import GridSearchCV

# Define the hyperparameters to search over
param_grid = {
    'C': [0.1, 1, 10],
    'penalty': ['l1', 'l2'],
    'solver': ['lbfgs', 'liblinear', 'sag', 'saga']
}

# Define the grid search
grid_search = GridSearchCV(estimator=lr, param_grid=param_grid, cv=5 )

```

```

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters and the corresponding accuracy score
print("Best hyperparameters:", grid_search.best_params_)

lr.fit(X_train, y_train)

# Evaluate function:
def evaluate(estimator, X, y, cross_val_score=None, on='train'):
    y_pred = estimator.predict(X)

    if not cross_val_score is None:
        print(f'mean accuracy on {on} set: ', cross_val_score.mean(), '\n')
    else:
        print(f'accuracy on {on} set: ', accuracy_score(y, y_pred), '\n')

    print(classification_report(y, y_pred))

# Confusion_matrix:
con_matrix = pd.crosstab(pd.Series(y, name='Actual'), pd.Series(y_pred, name='Predicted'))
plt.figure(figsize = (9,6))
plt.title("Confusion Matrix")
sns.heatmap(con_matrix, cmap="Blues", annot=True, fmt='g')

return y_pred

evaluate(lr, X_train, y_train, cvs, on='train')
evaluate(lr, X_test, y_test, on='test')

```

2. Fashion MNIST using NN.

```

import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"

print(tf.__version__)

(train_image,train_label),(test_image,test_label)= tf.keras.datasets.fashion_mnist.load_data()
train_image.shape
plt.imshow(train_image[0])

X_valid = train_image[:5000]/255
X_train=train_image[5000:]/255.0
#similarly for the y target labels
y_valid=train_label[:5000]
y_train=train_label[5000:]

model = tf.keras.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(28,28))) # 28*28
model.add(tf.keras.layers.Dense(200,activation="relu"))
model.add(tf.keras.layers.Dense(50,activation="relu"))
model.add(tf.keras.layers.Dense(10,activation="softmax",))

model.summary()

model.compile(loss='sparse_categorical_crossentropy',optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),metrics=["accuracy"])
history = model.fit(X_train,y_train,epochs=10,batch_size = 32,validation_data=(X_valid,y_valid))

plt.plot(history.epoch, history.history.get("accuracy"), label="accuracy")

```

```

plt.plot(history.epoch, history.history.get("val_accuracy"), label="val_accuracy")

plt.legend()
model.evaluate(test_image, test_label)

import seaborn as sns
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# Evaluate function:
def evaluate(estimator, X, y):

    # Use the trained model to predict the classes for the test images
    predictions = estimator.predict(X)

    # Convert the probability distribution for each sample to a single predicted class label
    predicted_labels = np.argmax(predictions, axis=1)

    # Calculate the classification accuracy of the model on the test set
    accuracy = np.mean(predicted_labels == y)

    y_pred = predicted_labels
    print(classification_report(y, y_pred))

    # Confusion matrix:
    con_matrix = pd.crosstab(pd.Series(y, name='Actual'), pd.Series(y_pred, name='Predicted'))
    plt.figure(figsize=(9,6))
    plt.title("Confusion Matrix")
    sns.heatmap(con_matrix, cmap="Blues", annot=True, fmt='g')

    return y_pred

evaluate(model, train_image, train_label);
evaluate(model, test_image, test_label);

```

3.Cifar10 using ML.

```

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import accuracy_score, classification_report

# load and reshape data

from tensorflow.keras.datasets import cifar10

(X_train, y_train), (X_test, y_test) = cifar10.load_data()

#reshape data
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)

y_train = y_train.ravel()
y_test = y_test.ravel()
y_train.shape

# Nomalize data by using MinmaxScaler
from sklearn.preprocessing import MinMaxScaler

X_train = MinMaxScaler().fit_transform(X_train)
X_test = MinMaxScaler().fit_transform(X_test)

# Check min_max after normalize:
(X_train.min(), X_train.max()), (X_test.min(), X_test.max())

```

```

## PCA

from sklearn.decomposition import PCA
# Plot PCA
sns.set_style('whitegrid')

plt.figure(figsize=(16, 9))
pca = PCA().fit(X_train)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')

# Variance ratio with 100 components:
pca.explained_variance_ratio_[:100].sum()

# To saving time when training and predicting,
# we can use data set after PCA with 100 components which have presented ~90.1% of the total
variance, for training model.

def pca_set(X_train, X_test, n_components):
    pca_used = PCA(n_components=n_components)
    # Transform data set:
    X_train_pca = pca_used.fit_transform(X_train)
    X_test_pca = pca_used.transform(X_test)
    print(f'DONE === with n_components: {pca_used.n_components_}')
    return X_train_pca, X_test_pca

X_train, X_test = pca_set(X_train, X_test, 100)

# logistic regression on Cifar10image

```

```

lr = LogisticRegression(max_iter=100,solver='saga')

cvs = cross_val_score(lr, X_train, y_train,
                      scoring='accuracy',
                      cv=None,
                      n_jobs=-1, verbose=1)

# Mean score of accuracy:
cvs.mean()

lr.fit(X_train, y_train)

# Evaluate function:
def evaluate(estimator, X, y, cross_val_score=None, on='train'):
    y_pred = estimator.predict(X)

    if not cross_val_score is None:
        print(f'mean accuracy on {on} set: ', cross_val_score.mean(), '\n')
    else:
        print(f'accuracy on {on} set: ', accuracy_score(y, y_pred), '\n')

    print(classification_report(y, y_pred))

# Confusion_matrix:
con_matrix = pd.crosstab(pd.Series(y, name='Actual'), pd.Series(y_pred, name='Predicted'))
plt.figure(figsize = (9,6))
plt.title("Confusion Matrix")
sns.heatmap(con_matrix, cmap="Blues", annot=True, fmt='g')

return y_pred

evaluate(lr, X_train, y_train, cvs, on='train')

evaluate(lr, X_test, y_test, on='test')

```

4.Cifar10 using NN

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)

import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import accuracy_score, classification_report

# load and reshape data

from tensorflow.keras.datasets import cifar10

(X_train,y_train),(X_test,y_test) = cifar10.load_data()

plt.imshow(X_train[0])

#reshape data
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)
```

```

y_train = y_train.ravel()
y_test = y_test.ravel()
y_train.shape

# Nomalize data by using MinmaxScaler
from sklearn.preprocessing import MinMaxScaler

X_train = MinMaxScaler().fit_transform(X_train)
X_test = MinMaxScaler().fit_transform(X_test)

# Check min_max after normalize:
(X_train.min(), X_train.max()),(X_test.min(), X_test.max())

## PCA

from sklearn.decomposition import PCA
# Plot PCA
sns.set_style('whitegrid')

plt.figure(figsize=(16, 9))
pca = PCA().fit(X_train)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')

# Variance ratio with 100 components:
pca.explained_variance_ratio_[0:100].sum()

# To saving time when training and predicting,
# we can use data set after PCA with 100 components which have presented ~90.1% of the total
variance, for training model.

def pca_set(X_train, X_test, n_components):
    pca_used = PCA(n_components=n_components)
    # Trainsform data set:
    X_train_pca = pca_used.fit_transform(X_train)

```

```

X_test_pca = pca_used.transform(X_test)
print(f'DONE === with n_components: {pca_used.n_components_}')
return X_train_pca, X_test_pca

X_train, X_test = pca_set(X_train, X_test, 100)

X_train.shape

X_valid = X_train[:5000]
X_train=X_train[5000:]
#similarly for the y target labels
y_valid=y_train[:5000]
y_train=y_train[5000:]

import tensorflow as tf
import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"

model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(300,activation="relu",input_shape=(100,)))
model.add(tf.keras.layers.Dense(100,activation="relu"))
model.add(tf.keras.layers.Dense(50,activation="relu"))
model.add(tf.keras.layers.Dense(10,activation="softmax",))

model.summary()

model.compile(loss='sparse_categorical_crossentropy',optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),metrics=["accuracy"])

history = model.fit(X_train,y_train,epochs=20,batch_size = 32,validation_data=(X_valid,y_valid))

```

```

history.history.keys()

plt.plot(history.epoch, history.history.get("accuracy"), label="accuracy")

plt.plot(history.epoch, history.history.get("val_accuracy"), label="val_accuracy")

plt.legend()

model.evaluate(X_test,y_test)

import seaborn as sns
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# Evaluate function:
def evaluate(estimator, X, y ):

    # Use the trained model to predict the classes for the test images
    predictions = estimator.predict(X)
    # Convert the probability distribution for each sample to a single predicted class label
    predicted_labels= np.argmax(predictions, axis=1)
    # Calculate the classification accuracy of the model on the test set
    accuracy = np.mean(predicted_labels == y)

    y_pred = predicted_labels
    print(classification_report(y, y_pred))

    # Confusion_matrix:
    con_matrix = pd.crosstab(pd.Series(y, name='Actual' ),pd.Series(y_pred, name='Predicted'))
    plt.figure(figsize = (9,6))
    plt.title("Confusion Matrix")
    sns.heatmap(con_matrix, cmap="Blues", annot=True, fmt='g')

    return y_pred

# Display result on test set

```

```
evaluate(model, X_test, y_test);
```

```
evaluate(model, X_train, y_train);
```


Assignment 1

GRADEMARK REPORT

FINAL GRADE

68 / 100

GENERAL COMMENTS

Instructor

Avoid use of first person - use 3rd person. Also refer to school ref std. Don't use superfix. See handbook and ref doc on Moodle.

Your lit review and related work covers required background theory but does not cover other related work. Eg it would be nice to see a summary of other researchers results when working with these datasets applied to ML classifiers. You can then compare your results with their efforts. Further discussion of early stopping and application please.

Overall a well structured report, well written (but don't use first person!). You have clearly researched the area but please cite your reference when asserting facts or to support arguments. Avoid too many electronic refs.

You could have explored changes to the models to improve results on CIFAR10.

Well done- good effort.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

 **Comment 1**
ref please

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

GRADING FORM: SMU GENERIC_GJ

BOWEN CHEN

0

RESEARCH-INFORMED LITERATURE

KNOWLEDGE AND UNDERSTANDING

ANALYSIS

PRACTICAL APPLICATION AND DEP.

SKILLS FOR PROFESSIONAL PRAC.