NYCU Pattern Recognition, Homework 2

311553010, 陳姿羽

Part. 1, Coding (70%):

1. (0%) Show the learning rate, epoch, and batch size that you used.
   learning rate: _ 0.005 __
   batch size: _ 50__
   epoch: _ 5000__

```
# For Q1
lr = 0.005
batch_size = 50
epoch = 5000

logistic_reg = MultiClassLogisticRegression()
logistic_reg.fit(X_train, y_train, lr=lr, batch_size=batch_size, epoch=epoch)
```

2. (5%) What's your training accuracy?
   training accuracy: _ 0.896__

```
# For Q2
print('Training acc: ', logistic_reg.evaluate(X_train, y_train))
```

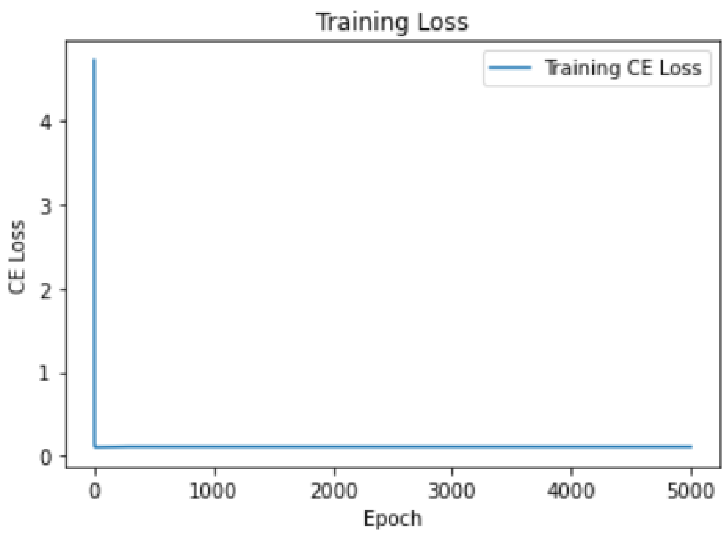
```
Training acc:  0.896
```

3. (5%) What's your testing accuracy?
   testing accuracy: _ 0.885__

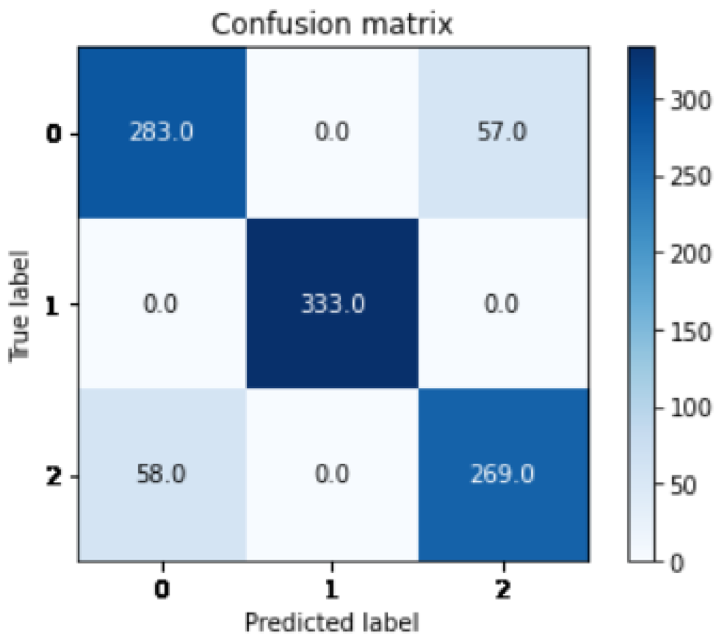
```
# For Q3
print('Testing acc: ', logistic_reg.evaluate(X_test, y_test))
```

```
Testing acc:  0.885
```

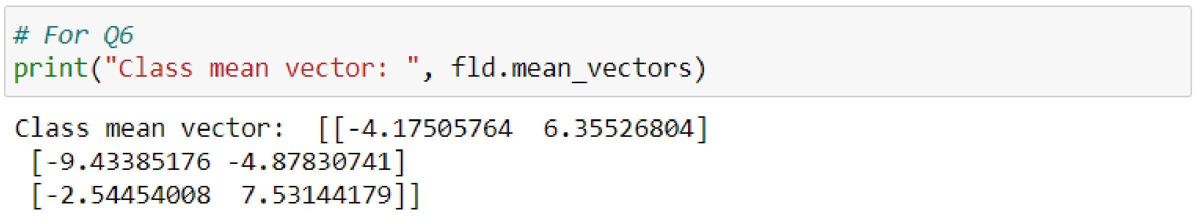
4. (5%) Plot the learning curve of the training. (x-axis=epoch, y-axis=loss)



5. (5%) Show the confusion matrix on testing data.



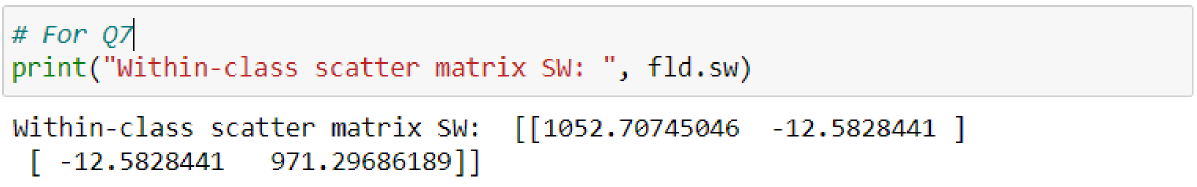
6. (2%) Compute the mean vectors mi (i=1, 2, 3) of each class on training data.
mean vectors m1: _ [-4.17505764 6.35526804] __
mean vectors m2: _ [-9.43385176 -4.87830741] __
mean vectors m3: _ [-2.54454008 7.53144179] __



```
# For Q6
print("Class mean vector: ", fld.mean_vectors)
```

```
Class mean vector:  [[-4.17505764  6.35526804]
 [-9.43385176 -4.87830741]
 [-2.54454008  7.53144179]]
```

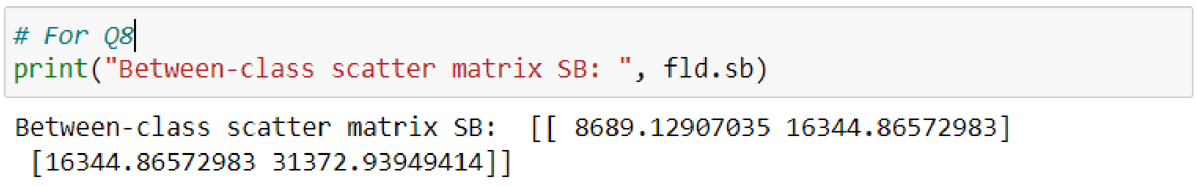
7. (2%) Compute the within-class scatter matrix SW on training data.
within-class scatter matrix SW: _ [[1052.70745046 -12.5828441]
[-12.5828441   971.29686189]] __



```
# For Q7
print("Within-class scatter matrix SW: ", fld.sw)
```

```
Within-class scatter matrix SW:  [[1052.70745046  -12.5828441 ]
 [ -12.5828441   971.29686189]]
```

8. (2%) Compute the between-class scatter matrix SB on training data.
between-class scatter matrix SB: _ [[8689.12907035 16344.86572983]
[16344.86572983 31372.93949414]] __



```
# For Q8
print("Between-class scatter matrix SB: ", fld.sb)
```

```
Between-class scatter matrix SB:  [[ 8689.12907035 16344.86572983]
 [16344.86572983 31372.93949414]]
```

9. (4%) Compute the Fisher's linear discriminant w on training data.
   Fisher's linear discriminant w: _ [-0.44115384 -0.8974315] __

```
# For Q9
print("W: ", fld.w)
```

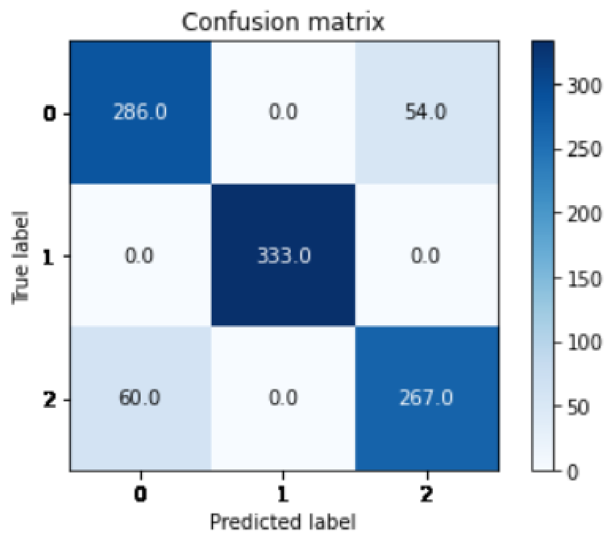
```
W:  [-0.44115384 -0.8974315 ]
```

10. (8%) Project the testing data to get the prediction using the shortest distance to the class mean. Report the accuracy score and draw the confusion matrix on testing data.
    FLD using class mean, accuracy: _ 0.886__

```
# For Q10
y_pred = fld.predict_using_class_mean(X_train, y_train, X_test)
print("FLD using class mean, accuracy: ", fld.accuracy_score(y_test, y_pred))

fld.show_confusion_matrix(y_test, y_pred)
```

```
FLD using class mean, accuracy:  0.886
```



11. (8%) Project the testing data to get the prediction using K-Nearest-Neighbor.
    Compare the accuracy score on the testing data with K values from 1 to 5.
    FLD using knn (k=1), accuracy: _0.839_
    FLD using knn (k=2), accuracy: _0.846_
    FLD using knn (k=3), accuracy: _0.878_
    FLD using knn (k=4), accuracy: _0.876_
    FLD using knn (k=5), accuracy: _0.878_

```
# For Q11
y_pred_k1 = fld.predict_using_knn(X_train, y_train, X_test, k=1)
print("FLD using knn (k=1), accuracy: ", fld.accuracy_score(y_test, y_pred_k1))

y_pred_k2 = fld.predict_using_knn(X_train, y_train, X_test, k=2)
print("FLD using knn (k=2), accuracy: ", fld.accuracy_score(y_test, y_pred_k2))

y_pred_k3 = fld.predict_using_knn(X_train, y_train, X_test, k=3)
print("FLD using knn (k=3), accuracy: ", fld.accuracy_score(y_test, y_pred_k3))

y_pred_k4 = fld.predict_using_knn(X_train, y_train, X_test, k=4)
print("FLD using knn (k=4), accuracy: ", fld.accuracy_score(y_test, y_pred_k4))

y_pred_k5 = fld.predict_using_knn(X_train, y_train, X_test, k=5)
print("FLD using knn (k=5), accuracy: ", fld.accuracy_score(y_test, y_pred_k5))
```

```
FLD using knn (k=1), accuracy:  0.839
FLD using knn (k=2), accuracy:  0.846
FLD using knn (k=3), accuracy:  0.878
FLD using knn (k=4), accuracy:  0.876
FLD using knn (k=5), accuracy:  0.878
```

12. (4%)

1) Plot the best projection line on the training data and show the slope and intercept on the title

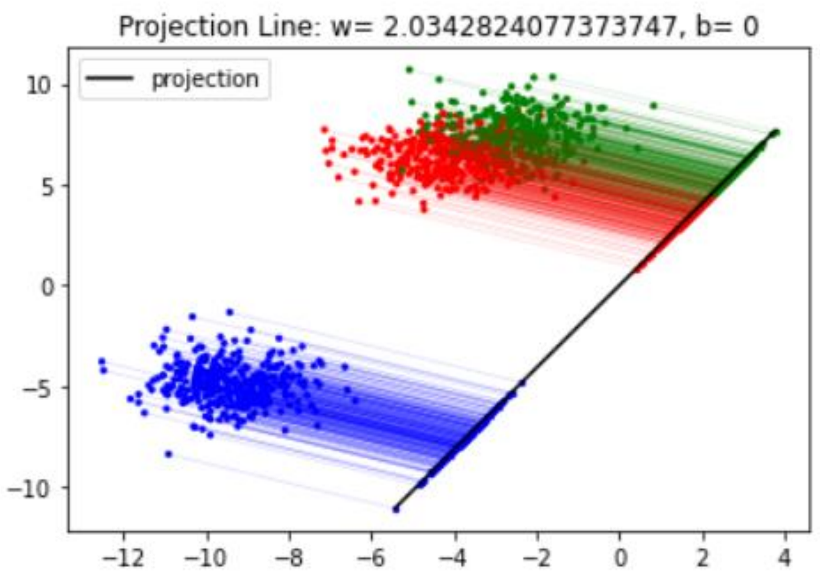
2) colorize the training data with each class

3) project all training data points on your projection line.

slope: _ 2.0342824077373747_

intercept: _0_

```
# For Q12, using only training data
fld.plot_projection(X_train, y_train)
```



13. Explain how you chose your model and what feature processing you have done in detail. Otherwise, no points will be given.
learning rate: _0.03__
epoch: _ 1000 __
batch_size: _50__
Used features: _Feature1, Feature2, Feature3, Feature4__
Training accuracy: _ 0.9201409277745156__
Validation accuracy: _ 0.9123287671232877__

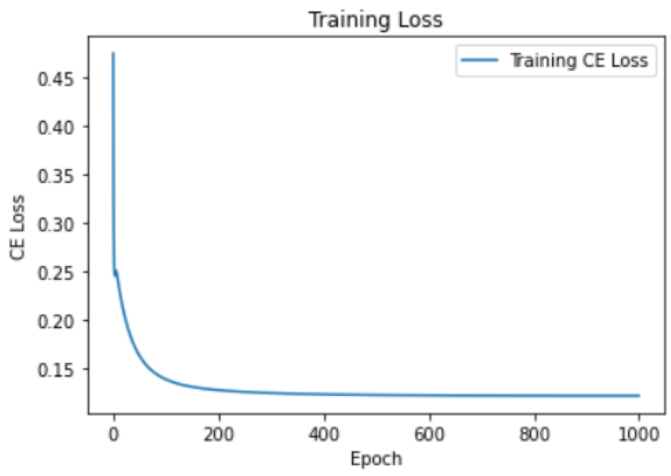
First, I used all features to choose the model. I found that the accuracy of Fisher's Linear Discriminant couldn't achieve above 0.82, so I decided to use Logistic Regression. Next, I tried different combinations of features to find the best set, and I found that using all features resulted in the highest accuracy. Then, I started modified the learning rate, batch size, and epochs, but the accuracy still couldn't exceed 0.79. After referring to the TA's suggestion, I used the data pre-processing by subtracting the mean value and dividing by the standard deviation, and the accuracy exceeded 0.8. I also found that the accuracy can be exceeded 0.9 after multiplying the data by 10. Finally, I modified the learning rate, batch size, and epochs to achieve an accuracy above 0.91.

```python
# Train your model here
lr = 0.030
batch_size = 50
epoch = 1000

your_model = MultiClassLogisticRegression()
your_model.fit(X_train_my, y_train_my, lr=lr, batch_size=batch_size, epoch=epoch)
```

```python
print('Training acc: ', your_model.evaluate(X_train_my, y_train_my))
print('Validation acc: ', your_model.evaluate(X_val_my, y_val_my))
your_model.plot_curve()
```

```
Training acc:  0.9201409277745156
Validation acc:  0.9123287671232877
```



Part. 2, Questions (30%):

(6%) 1. Discuss and analyze the performance

a) between Q10 and Q11, which approach is more suitable for this dataset. Why?
   "predict_using_class_mean" directly uses the class closest to the mean for classification. And "predict_using_knn" uses the k-nearest neighbors with the closest distance to predict the data.
   However, when two classes are very close, it is easy to find neighbors of the other class, which can lead to prediction errors. For example: The prediction errors in "predict_using_class_mean" may occur when data of target = 0 is closer to the mean of target = 1, and the prediction result is target=1.The prediction errors in "predict_using_knn" may occur when most of the data is target = 1 and the k-nearest neighbors is target = 0, and the prediction result is target=1.
   In this dataset, there are two classes of data that are close to each other. And the prediction errors in "predict_using_knn" are more likely to occur in these two classes, resulting in lower accuracy. Therefore, "predict_using_class_mean" is more suitable for this dataset.

b) between different values of k in Q11. (Which is better, a larger or smaller k? Does this always hold?)
   We assume there are 3 data points(target = 0) and 4 data points(target = 1) around a data(target = 0), and the 3 data points(target = 0) are closer to the data(target = 0). When k <= 7, the data will be predicted as target = 0. When k > 7, the data will be predicted as target = 1.
   On the other hand, we assume there are 5 data points(target = 0) and 1 data point(target = 1) around a data(target = 0), and the 1 data point(target = 1) is closer to the data(target = 0). When k <= 2, the data will be predicted as target = 1. When k > 2, the data will be predicted as target = 0.
   For the first assumption, when k > 7, the model may be underfitting. For the second assumption, when k <= 2, the model may be overfitting.
   Therefore, if the k value is too large or too small, it may not perform well. In the assignment, the accuracy of k = 5 is higher than k = 1, so k = 5 is a suitable value for this model.

(6%) 2. Compare the sigmoid function and softmax function.

The Sigmoid function is often used for binary classification problems.
The advantage of the Sigmoid function is that its output values are all between 0 and 1, and it can convert into a probability value. However, its disadvantage is that when the output values are very large or very small, the gradient can be very small, leading to the problem of vanishing gradients. Additionally, the Sigmoid function cannot be used when multiple outputs need to be converted into probability distributions.
The Softmax function is often used for multiclass classification problems.

The advantage of the Softmax function is that it can convert multiple outputs into probability distributions. However, the disadvantage of the Softmax function is that it is sensitive to changes in the input values. Additionally, when some values in the input are very large, the Softmax function can become very unstable.
The Sigmoid function is useful in binary classification problems, while the Softmax function is an ideal activation function in multiclass classification problems. However, if the values in the input vector are very large or very small, the both functions can have problems.

(6%) 3. Why do we use cross entropy for classification tasks and mean square error for regression tasks?

Cross entropy is a loss function that measures how well the predicted probabilities match the actual probabilities of the output classes. It is commonly used in classification tasks because the goal of classification tasks is to predict the correct class label for a given input. And the probability distribution provides a measure of how confident the model is in its predictions.
Mean square error is a loss function that measures the average squared difference between the predicted values and the true values of the output variables. It is commonly used in regression tasks because the goal of regression tasks is to predict a continuous value. And the mean square error provides a measure of how well the model is able to predict that value.
In classification tasks, cross entropy is preferred than mean square error because it is more suitable for measuring the similarity between probability distributions. The goal of classification tasks is to predict the correct class label with high confidence.
In regression tasks, mean square error is preferred than cross entropy because it is more suitable for measuring the similarity between continuous values. The goal of regression tasks is to predict a value as close as possible to the true value.

(6%) 4. In Q13, we provide an imbalanced dataset. Are there any methods to improve Fisher Linear Discriminant's performance in handling such datasets?

There are some methods to improve FLDA's performance in imbalanced datasets:

a、 Assigns different weights: Assigns different weights to different classes to make the classifier pay more attention to the minority class and balance the importance of each class in the classification. And the weights are usually determined by the inverse proportion of the class frequencies.

b、 Oversampling: Increase the number of samples in the minority class to make the dataset more balanced. We can duplicate samples or synthesize new samples to increase the number of samples.

c、 Undersampling: Reducing the number of samples in the majority class to balance the dataset. We can randomly drop samples or select subsets to reduce the number of samples.

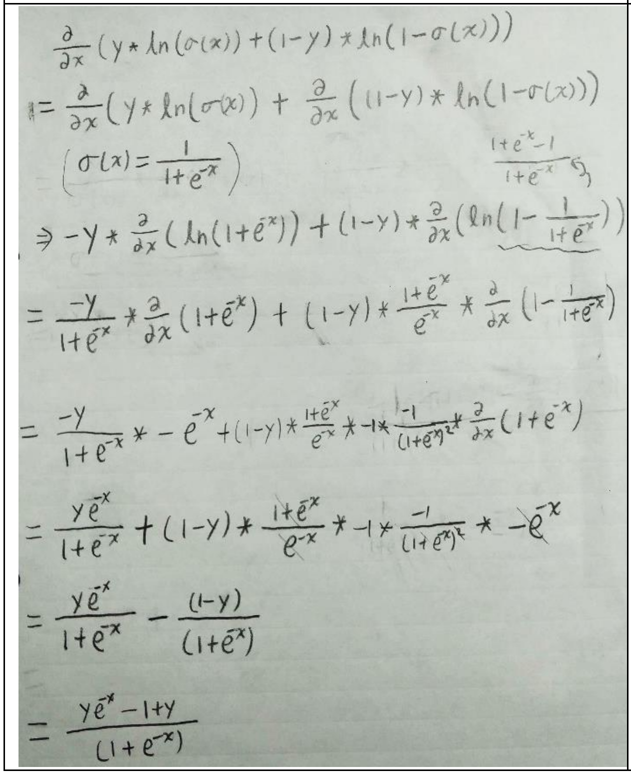
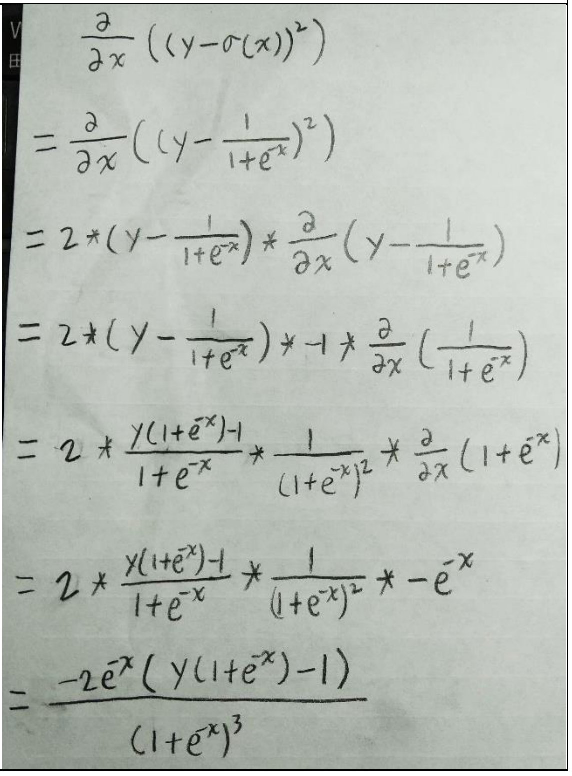
d、 Hybrid methods: Combine over-sampling and under-sampling techniques to create a balanced dataset.

e、 Dimensionality reduction: Reduce the dimensionality of the dataset by selecting only the most discriminative features. We can use feature selection techniques such as mutual information, correlation-based feature selection, or recursive feature elimination.

(6%) 5. Calculate the results of the partial derivatives for the following equations. (The first one is binary cross-entropy loss, and the second one is mean square error loss followed by a sigmoid function.)

$$\frac{\partial}{\partial x}\left(\mathrm{y} * \ln(\sigma(x)) + (1-y) * \ln(1-\sigma(x))\right)$$

$$\frac{\partial}{\partial x}\left((\mathrm{y}-\sigma(x))^2\right)$$

$\frac{\partial}{\partial x}\left(\mathrm{y} * \ln(\sigma(x)) + (1-y) * \ln(1-\sigma(x))\right)$	$\frac{\partial}{\partial x}\left((\mathrm{y}-\sigma(x))^2\right)$
 $\frac{\partial}{\partial x}(y * \ln(\sigma(x)) + (1-y) * \ln(1-\sigma(x)))$ $= \frac{\partial}{\partial x}(y * \ln(\sigma(x))) + \frac{\partial}{\partial x}((1-y) * \ln(1-\sigma(x)))$ $\left(\sigma(x) = \frac{1}{1+e^{-x}}\right)$ $\frac{1+e^{-x}-1}{1+e^{-x}}$ $\Rightarrow -y * \frac{\partial}{\partial x}(\ln(1+e^{-x})) + (1-y) * \frac{\partial}{\partial x}(\ln(1-\frac{1}{1+e^{-x}}))$ $= \frac{-y}{1+e^{-x}} * \frac{\partial}{\partial x}(1+e^{-x}) + (1-y) * \frac{1+e^{-x}}{e^{-x}} * \frac{\partial}{\partial x}(1-\frac{1}{1+e^{-x}})$ $= \frac{-y}{1+e^{-x}} * -e^{-x} + (1-y) * \frac{1+e^{-x}}{e^{-x}} * -1 * \frac{-1}{(1+e^{-x})^2} * \frac{\partial}{\partial x}(1+e^{-x})$ $= \frac{ye^{-x}}{1+e^{-x}} + (1-y) * \frac{\cancel{1+e^{-x}}}{\cancel{e^{-x}}} * -1 * \frac{-1}{(1+e^{-x})^2} * -\cancel{e^{-x}}$ $= \frac{ye^{-x}}{1+e^{-x}} - \frac{(1-y)}{(1+e^{-x})}$ $= \frac{ye^{-x}-1+y}{(1+e^{-x})}$	 $\frac{\partial}{\partial x}((y-\sigma(x))^2)$ $= \frac{\partial}{\partial x}((y-\frac{1}{1+e^{-x}})^2)$ $= 2 * (y-\frac{1}{1+e^{-x}}) * \frac{\partial}{\partial x}(y-\frac{1}{1+e^{-x}})$ $= 2 * (y-\frac{1}{1+e^{-x}}) * -1 * \frac{\partial}{\partial x}(\frac{1}{1+e^{-x}})$ $= 2 * \frac{y(1+e^{-x})-1}{1+e^{-x}} * \frac{1}{(1+e^{-x})^2} * \frac{\partial}{\partial x}(1+e^{-x})$ $= 2 * \frac{y(1+e^{-x})-1}{1+e^{-x}} * \frac{1}{(1+e^{-x})^2} * -e^{-x}$ $= \frac{-2e^{-x}(y(1+e^{-x})-1)}{(1+e^{-x})^3}$