Q1-a:

Lets put the facts down:

A single layer neuron with sigmoid activation function look like:

$$z = \sum x_i w_i = x^T w \quad \text{input}$$

$$g(z) = \frac{1}{1+e^{-z}} \Rightarrow g(x^T w) = \frac{1}{1+e^{-x^T w}} \quad (1)$$

logistic regression:

$$z = x_0\beta_0 + x_1\beta_1 + x_2\beta_2 + \cdots \overset{\text{replace } \beta \text{ with } w}{=} x_0 w_0 + x_1 w_1 + x_2 w_2 + \cdots = x^T w$$

Then hypothesis function for logistic regression is

$$\log \frac{p}{1-p} = x^T w \Rightarrow p = \frac{1}{1+e^{-x^T w}} \quad (2)$$

(1), (2) the output for logistic regression $P(y=1|x,w)$ (2)

is same as one layer sigmoid neuron network model (1)

$\Rightarrow$ one layer (no hidden layers) NN = regression model.

Q1-b:

for a simple two-layer network the cost function would look like:

$$\ell(w,\alpha,\beta) = \sum_i^n (y^i - \delta(w^T z^i))^2$$

- $z_1^i = \delta(\alpha^T x^i)$; $z_2^i = \delta(\beta^T x^i)$

layer 1 layer 2

- $\delta(x) = \frac{1}{1+e^{-x}}$

the gradient with respect to w:

$$\frac{\partial(\ell(w,\alpha,\beta))}{\partial w} = \frac{\partial(\ell(w,\alpha,\beta))}{\partial u} \times \frac{\partial u}{\partial w} \quad \text{(chain rule)}$$

$$= \frac{\partial}{\partial u} \sum_i^n (y^i - \delta(u^i))^2 \times \frac{\partial u^i}{\partial w} \quad \leftarrow \boxed{u^i = w^T z^i}$$

$$= \sum_i^n 2(y^i - \delta(u^i)) \times -1 \times \frac{\partial}{\partial u}\delta(u^i) \times \frac{\partial u^i}{\partial w}$$

$$= \sum_i^n 2(y^i - \delta(u^i)) \times -1 \times \underbrace{\delta(u^i)(1-\delta(u^i))}_{\text{derivation of sigmoid}} \times \frac{\partial w^T z^i}{\partial w}$$

$$= -\sum_i^n 2(y^i - \delta(u^i))\,\delta(u^i)(1-\delta(u^i)) \times z^i$$

$$\frac{\partial(\ell(W,\alpha,\beta))}{\partial\alpha} = \frac{\partial}{\partial u}\sum_i^n (y^i - \delta(u^i))^2 \times \frac{\delta u^i}{\partial v} \times \frac{\partial v}{\partial \alpha} \quad \leftarrow \boxed{v_i = \alpha^T x^i}$$

$$= -\sum_i^n 2(y^i - \delta(u^i))\,\delta(u^i)(1-\delta(u^i))\,\frac{\partial\, w_1^T z_1^i}{\partial v} \times \frac{\partial v}{\partial \alpha}$$

$$= -\sum_i^n 2(y^i - \delta(u^i))\,\delta(u^i)(1-\delta(u^i))\,w_1 \times \delta(v^i)(1-\delta(v^i)) \times \frac{\partial(\alpha^T x^i)}{\partial \alpha}$$

$$\boxed{z_1^i = \delta(\alpha^T x^i) = \delta(v^i)}$$

$$= -\sum_i^n 2(y^i - \delta(u^i))\,\delta(u^i)(1-\delta(u^i))\,w_1\,\delta(v^i)(1-\delta(v^i))\,x^i$$

$$\frac{\partial(\ell(W,\alpha,\beta))}{\partial\beta} = \frac{\partial}{\partial u}\sum_i^n (y^i - \delta(u^i))^2 \times \frac{\delta u}{\partial v} \times \frac{\partial v}{\partial \beta}$$

$$= \text{similar rule chain applied here}$$

$$v^i = \beta^T x^i$$

$$z_2^i = \delta(\beta^T x^i) = \delta(v^i)$$

$$= -\sum_i^n 2(y^i - \delta(u^i))\,\delta(u^i)(1-\delta(u^i))\,\underline{w_2}\,\delta(v^i)(1-\delta(v^i))\,x^i$$

Q2 – Part One (a)

In this report, I assume that performance is translated into accuracy. Higher accuracy, higher performance.

Report of testing accuracy for **support vector machine** with kernel "Radial Basis Function" (rbf) and gamma to auto. The accuracy of SVM is 97% for the **test data (20%).**

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
              precision    recall  f1-score   support

         0.0       0.93      1.00      0.97        14
         1.0       1.00      0.95      0.97        20

    accuracy                           0.97        34
   macro avg       0.97      0.97      0.97        34
weighted avg       0.97      0.97      0.97        34

[[14  0]
 [ 1 19]]
```

The accuracy for the **entire data** using the SVM is 99%

```
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99        86
         1.0       1.00      0.99      0.99        84

    accuracy                           0.99       170
   macro avg       0.99      0.99      0.99       170
weighted avg       0.99      0.99      0.99       170

[[86  0]
 [ 1 83]]
```

Report of testing accuracy for simple **neural network** with hidden layers (5, 2), activation function "Relu", is **100%**. The model managed classify all 34 data records correctly.

```
MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(5, 2), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=200,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=1, shuffle=True, solver='lbfgs',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        14
         1.0       1.00      1.00      1.00        20

    accuracy                           1.00        34
   macro avg       1.00      1.00      1.00        34
weighted avg       1.00      1.00      1.00        34

[[14  0]
 [ 0 20]]
```

The accuracy decreases to 99% when we test the model on the entire data. There was only one miss classification only.

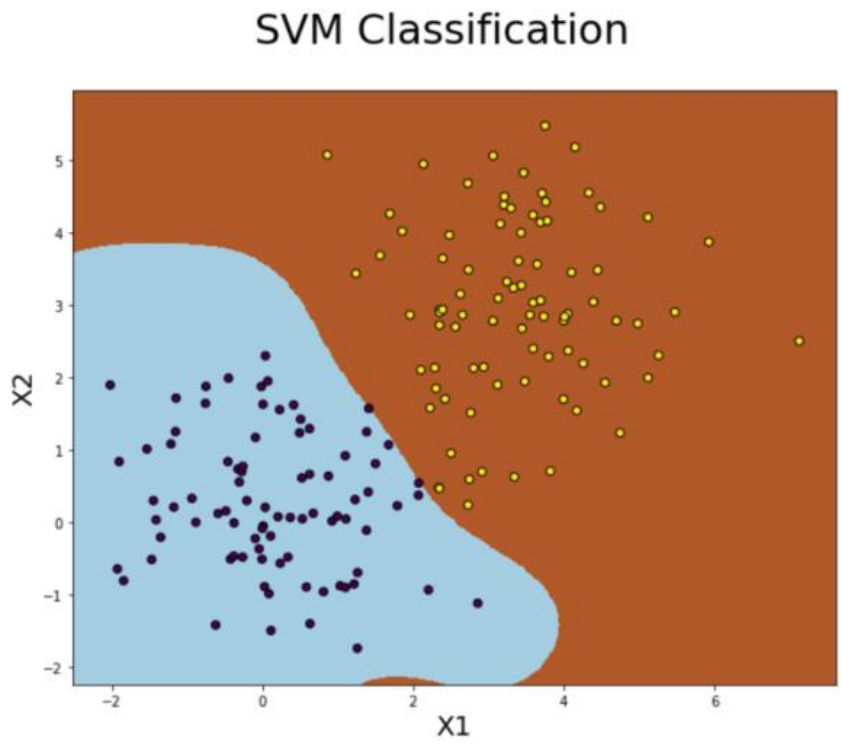
	precision	recall	f1-score	support
0.0	1.00	0.99	0.99	86
1.0	0.99	1.00	0.99	84
accuracy			0.99	170
macro avg	0.99	0.99	0.99	170
weighted avg	0.99	0.99	0.99	170

```
[[85  1]
 [ 0 84]]
```

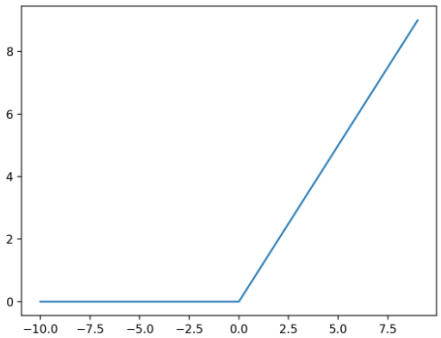
Conclusion: neural network model performed better than SVM and this result was shown by 100% accuracy on the test data. Nevertheless, when we test both on the entire data set, they get similar overall accuracy. All in all, neural network performs better on high dimensional data such as the "divorce predictors" data set which has 55 attributes. Specifically, that we can tweak the neural network model with various activation functions and hidden layers.

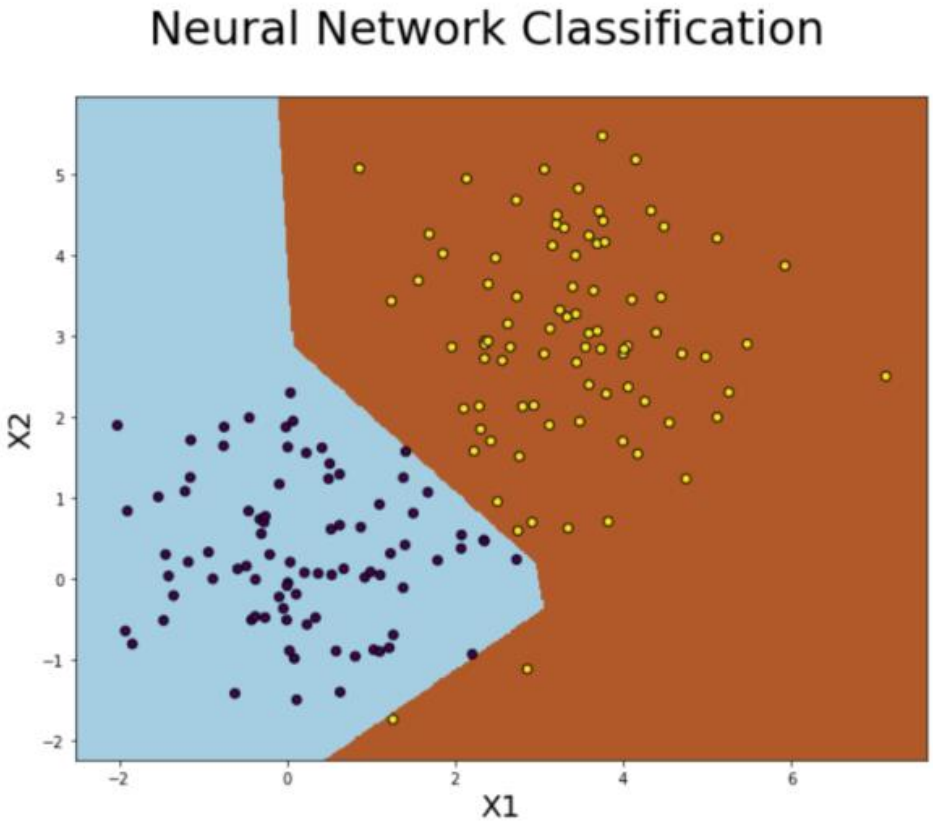
Q2 – Part one (b)

The decision boundary for SVM is as below. We are using a kernelized SVM with **Gaussian RBF** in which is a function whose value depends on the distance from the origin. And we also used **Gamma** which is a parameter of Gaussian Kernel to handle non-linear classification. Hence, we see the decision boundary is also non-linear.



On the other hand, for Neural Network model, we are using "Relu" as activation function. This is a non-linear activation function and due to its nature, Relu function has sharp boundaries. Hence, we see this non-smooth non-linear classification boundaries for our neural network model.





Conclusion: The shape of decision boundary for both SVM and Neural Network depends on the type of function we use for activation (in case of NN) and Kernel (in case of SVM). Relu, due to its nature, produces sharp decision boundaries and we RBF created smoother one. Depends on the nature of activation function we can generate decision boundaries with smoother edges. That said, even with the current state of our neural network model, the precision is very impressive.

Q2 – Part Two (a)

In this section we are going to do classification on data with more dimension compared to the one in Part One. Here the data has 784 properties.

Report of testing accuracy for **support vector machine** with kernel "Radial Basis Function" (rbf) and gamma to auto. The accuracy of SVM is 98% for the **test data (20%).**

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
              precision    recall  f1-score   support

           0       0.98      0.97      0.98       201
           1       0.97      0.98      0.98       197

    accuracy                           0.98       398
   macro avg       0.98      0.98      0.98       398
weighted avg       0.98      0.98      0.98       398

[[195   6]
 [  3 194]]
```

The accuracy for the **entire data** using the SVM is 98%. This reasonable that we are dividing our data and we have enough data in the test set that represents the entire data.

```
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      1032
           1       0.96      0.99      0.98       958

    accuracy                           0.98      1990
   macro avg       0.98      0.98      0.98      1990
weighted avg       0.98      0.98      0.98      1990

[[997  35]
 [ 12 946]]
```

Report of testing accuracy for simple **neural network** with hidden layers (5, 2), activation function "Relu", is **99%**. The model managed classify all 34 data records correctly.

```
MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(5, 2), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=200,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=1, shuffle=True, solver='lbfgs',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       201
           1       0.99      0.98      0.98       197

    accuracy                           0.98       398
   macro avg       0.99      0.98      0.98       398
weighted avg       0.98      0.98      0.98       398

[[199   2]
 [  4 193]]
```

The accuracy decreases to 100% when we test the model on the entire data. There was only 6 miss classification only. But compare to total number of data we have, the accuracy is 100%.

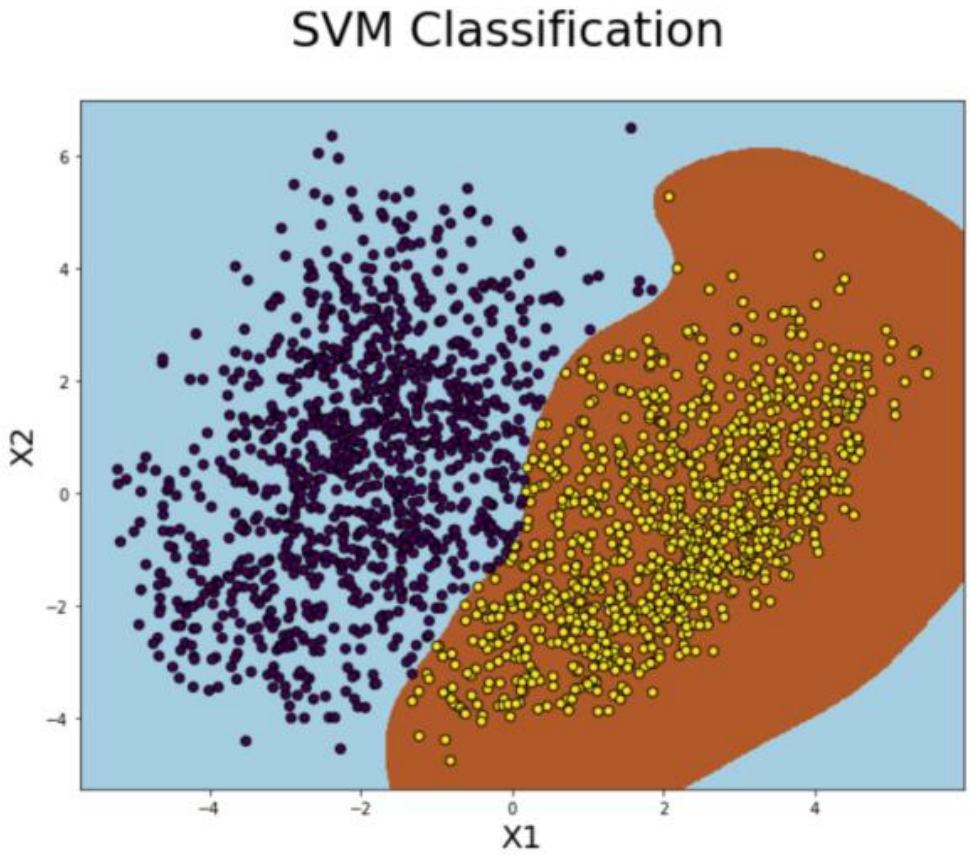
	precision	recall	f1-score	support
0	1.00	1.00	1.00	1032
1	1.00	1.00	1.00	958
accuracy			1.00	1990
macro avg	1.00	1.00	1.00	1990
weighted avg	1.00	1.00	1.00	1990

```
[[1030    2]
 [   4  954]]
```

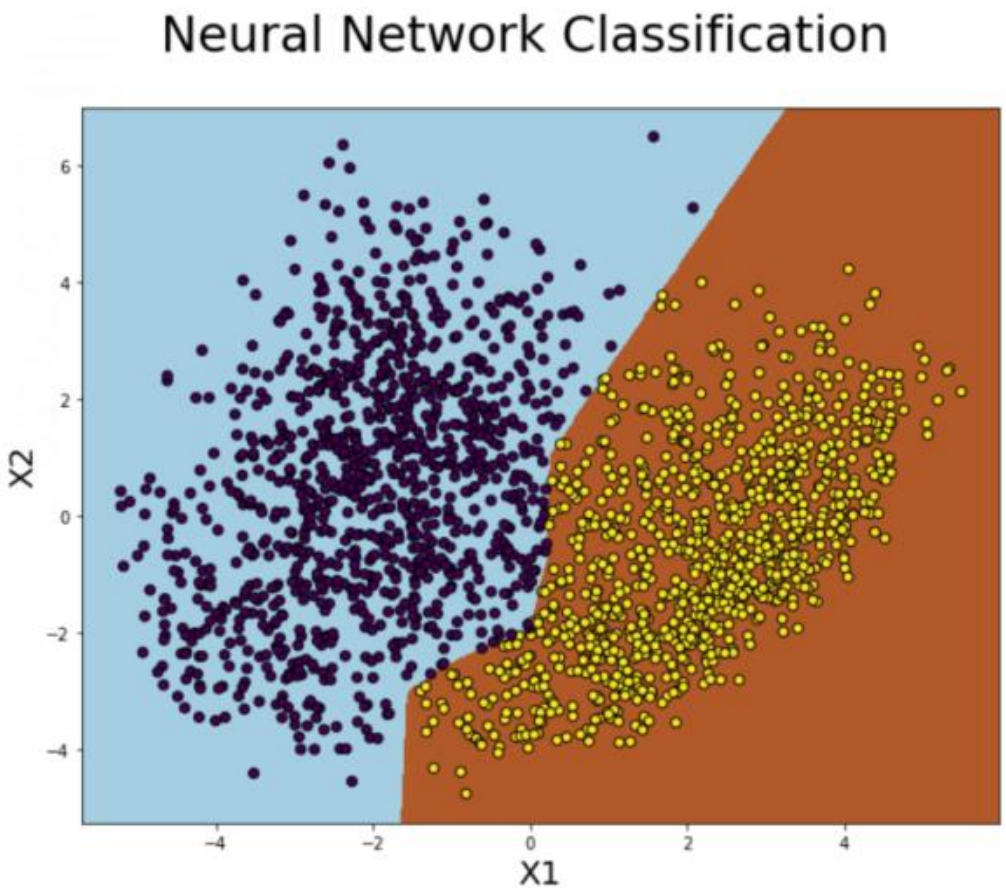
Conclusion: neural network model performed better than SVM for MNIST data set both with the test data and the entire data. Overall, neural network performs with high accuracy with respect to large dimensional data. The conclusion is like Part One.

Q2 – Part Two (b)

The decision boundary for SVM for the first two PCA dimension is as below. Here, similar to Part One, We are using a kernelized SVM with **Gaussian RBF** in which is a function whose value depends on the distance from the origin. And we also used **Gamma** which is a parameter of Gaussian Kernel to handle non-linear classification. Hence, we see the decision boundary is also non-linear.



On the other hand, for Neural Network model, again we are using “Relu” as activation function. Relu is a non-linear activation function and due to its nature that Relu function has sharp boundaries. Hence, we see this non-smooth non-linear classification boundaries for our neural network model.



Conclusion: {Similar to Part One (b)} The shape of decision boundary for both SVM and Neural Network depends on the type of function we use for activation (in case of NN) and Kernel (in case of SVM). Relu, due to its nature, produces sharp decision boundaries and we RBF created smoother one. Depends on the nature of activation function we can generate decision boundaries with smoother edges. That said, even with the current state of our neural network model, the precision is very impressive.

Q3-(a)

For this section I used Excel to do all my calculations for epsilon, alpha, Z, and the weight of each data point in each iteration. The final answers are in the table below and the excel sheet is attached to the final report (Zip file):

t	epsilon	alpha	Z	Dt(1)	Dt(2)	Dt(3)	Dt(4)	Dt(5)	Dt(6)	Dt(7)	Dt(8)
1	0.25	0.5493061	0.866025404	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
2	0.833333333	-0.804719	0.645497224	0.08333333	0.083333333	0.0833333	0.083333333	0.25	0.25	0.083333	0.083333
3	0.2	0.6931472	0.516397779	0.25	0.25	0.05	0.05	0.15	0.15	0.05	0.05

Please check the excel sheet to follow all the calculations based on AdaBoost algorithm.

Q3-b

IF T is the last iteration for AdaBoost, then we will prove that the training error for the Algorithm at most will be:

$$\text{Training error} \leq e^{-2\sum_{t=1}^{T}\gamma_t^2} = \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right) \qquad \gamma_t = \frac{1}{2} - \varepsilon_t$$

we will prove with three subproof:

1) $w_{T+1}^{(i)} = \frac{1}{N} \cdot \frac{\exp(-y_i f(x_i))}{\prod_t Z_t}$ $\qquad f(x) = \sum_t \alpha_t h_t(x)$

2) training error $= \prod_t Z_t$

3) $Z_t \leq e^{-2\gamma_t^2}$

---

1) from AdaBoost we have:

$$w_{T+1}^{(i)} = \frac{w_T^{(i)}}{Z_T} \times \begin{cases} e^{-\alpha_T} & \text{if } y_i = h_T(x_i) \\ e^{\alpha_T} & \text{if } y_i \neq h_T(x_i) \end{cases} = \frac{w_T^{(i)}}{Z_T} e^{-\alpha_T y_i h_T(x_i)}$$

unwrapping the recurrance →

$$= \frac{w_1^{(i)}}{Z_2} e^{-\alpha_1 y_i h_1(x_i)} \times e^{-\alpha_2 y_i h_2(x_i)} \times \cdots \times e^{-\alpha_T y_i h_T(x_i)}$$

$$= \frac{1}{N} \times \frac{1}{\prod_t Z_t} \times e^{-y_i \sum \alpha_t h_t(x_i)} = \frac{1}{N} \frac{e^{-y_i f(x_i)}}{\prod_t Z_t} \quad (1)$$

2) training error $= \frac{1}{N} \sum_i \begin{cases} 1 & \text{if } y_i \neq H(x_i) \\ 0 & \text{else} \end{cases}$ → Final Adboost Function

$$= \frac{1}{N} \sum_i \begin{cases} 1 & \text{if } y_i f(x_i) \leq 0 \\ 0 & \text{else} \end{cases}$$

$\leq \frac{1}{N} \sum_i \exp(-y_i f(x_i))$ ← $e^x \geq 1$ if $x \geq 0$

$= \sum_i w_{T+1}^{(i)} \prod_t Z_t = \prod_t Z_t$ (2) $\qquad \sum_i w_{T+1}^{(i)} = 1$ normalized weight

3)

3) $Z_t = \sum_i w_t(i) \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$

$= \sum_{i | h_t(x_i) = y_i} w_t(i) e^{-\alpha_t} + \sum_{i | h_t(x_i) \neq y_i} w_t(i) e^{\alpha_t}$

$= e^{-\alpha_t} \sum_{i | h_t(x_i) = y_i} w_t(i) + e^{\alpha_t} \sum_{i | h_t(x_i) \neq y_i} w_t(i)$ → definition of ε_t

$= e^{-\alpha_t}(1 - \varepsilon_t) + e^{\alpha_t} \varepsilon_t$ ← $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$

$= 2\sqrt{\varepsilon_t(1-\varepsilon_t)}$

$= \sqrt{1 - 4\gamma_t^2} \leq e^{-2\gamma_t^2}$ (3) ← $\varepsilon_t = \frac{1}{2} - \gamma_t^2$

(1), (2), (3) $\Rightarrow$ training error for $H = e^{-2\sum_{t=1}^{T} \gamma_t^2}$

- Single decision stump is a weak classifier that is slightly better than random on training data with high variance.
Ada boost is combination of weak classifiers that lead to low variance and low bias. Hence AdaBoost > Single Decision Stump