1. Basics:

a. A hypothesis set is a set of potential functions that a learning algorithm will choose from a single hypothesis that best approximates the target function.

b. For a linear model it is the set of all functions in the form f(x | w, b) = transpose(w) * x - b where w is our weight vector.

c. Overfitting occurs when we fit the data more than warranted meaning that we are fitting the training data too well hence making our learned hypothesis bad for generalizations as we are fitting the noise too. Overfitting results in low in sample or training error and high out of sample or testing error as our hypothesis is bad for generalization.

d. The ways to prevent overfitting is by having a bigger training set, using regularization, or using validation.

e. Training data is the data set we use apply a learning algorithm on to identify a pattern by picking a hypothesis that best approximates our target function and test data is the data we apply our final hypothesis on to see if the our hypothesis is a good generalization of this data by measuring the out of sample error to see how well our model learned the distribution as both the training and test data are from the same distribution. You should never change your learning model based on information from test data because otherwise you will be fitting the noise which results in a hypothesis that is bad for generalization.

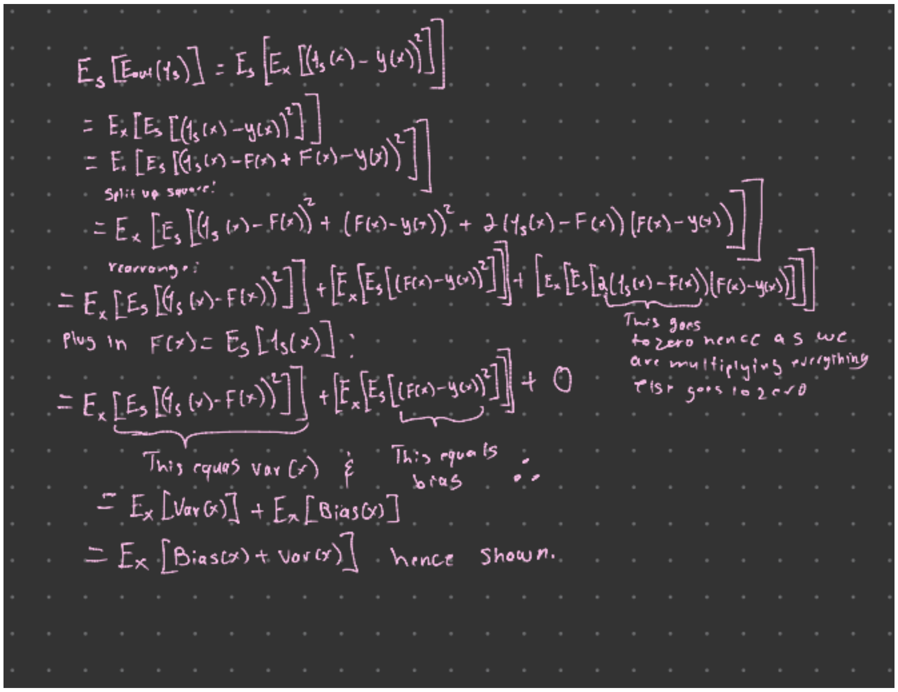
f. We assume that the dataset is a random sample and that it is representative and of the population we are trying to generalize.

g. Our input space could be a Bag Of Words vector representation of email titles with zero meaning the word is absent and one meaning the word is present. The output space would be a classification of whether that email is a spam, represented by 1, or not a spam, represented by -1.

h. The k-fold cross-validation procedure is the process of splitting our dataset into k equal clusters (most times putting them into random clusters for better results) and then setting one cluster aside as a test set and train the learning model on the remaining clusters and testing the learning model on the test cluster. Then repeat the process until each cluster has served as a test set and each has served as a part of the training set thus the entire dataset is used for validation; finish by averaging the in sample and out of sample error.

2. Bias-Variance Tradeoff:

a.



$$E_S\left[E_{out}(g_S)\right] = E_S\left[E_x\left[(g_S(x) - y(x))^2\right]\right]$$

$$= E_x\left[E_S\left[(g_S(x) - y(x))^2\right]\right]$$

$$= E_x\left[E_S\left[(g_S(x) - F(x) + F(x) - y(x))^2\right]\right]$$

Split up square:

$$= E_x\left[E_S\left[(g_S(x) - F(x))^2 + (F(x) - y(x))^2 + 2(g_S(x) - F(x))(F(x) - y(x))\right]\right]$$

rearrange:

$$= E_x\left[E_S\left[(g_S(x) - F(x))^2\right]\right] + \left[E_x\left[E_S\left[(F(x) - y(x))^2\right]\right]\right] + \left[E_x\left[E_S\left[2\underbrace{(g_S(x) - F(x))}_{\text{This goes to zero hence as we are multiplying everything else goes to zero}}(F(x) - y(x))\right]\right]\right]$$

Plug in $F(x) = E_S[g_S(x)]$:

$$= E_x\underbrace{\left[E_S\left[(g_S(x) - F(x))^2\right]\right]}_{\text{This equas var}(x)} + \left[E_x\left[E_S\left[\underbrace{(F(x) - y(x))^2}_{\text{This equals bias}}\right]\right]\right] + 0$$

This equas var (x) & This equals bias $\therefore$

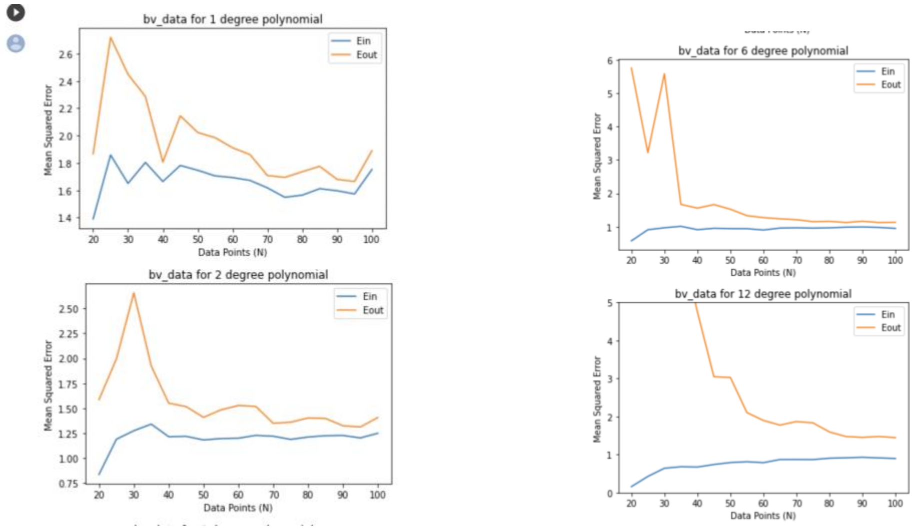
$$= E_x[Var(x)] + E_x[Bias(x)]$$

$$= E_x[Bias(x) + Var(x)]$$ hence shown.

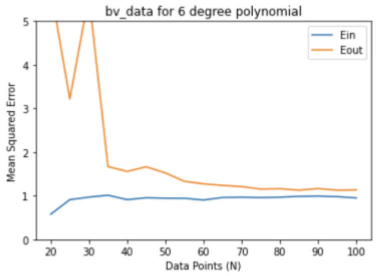
[redacted]

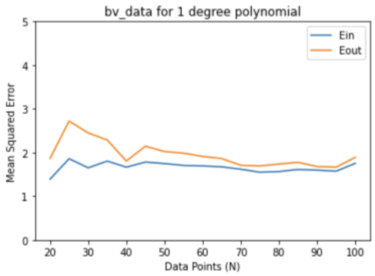
[redacted]

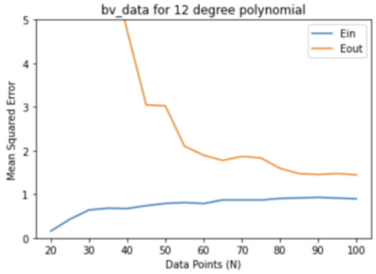
See code for 2

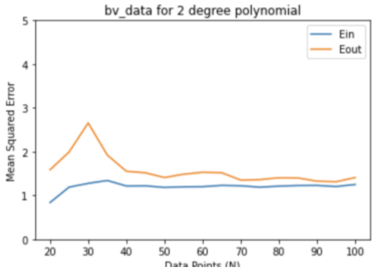


Plot with all scaled:









b. The 1st degree polynomial has the highest because in training error (Ein) and the validation error (Eout) converges to the highest mean squared error compared to the other models meaning that it has the highest bias as the bias is the height of that convergence line as discussed in CS 156a. Furthemore, compared to the other models, we see that the 1st order polynomial yielded the highest average squared training error (Ein) meaning that the model is not complex enough to capture the training data trends implying underfitting due to high bias.

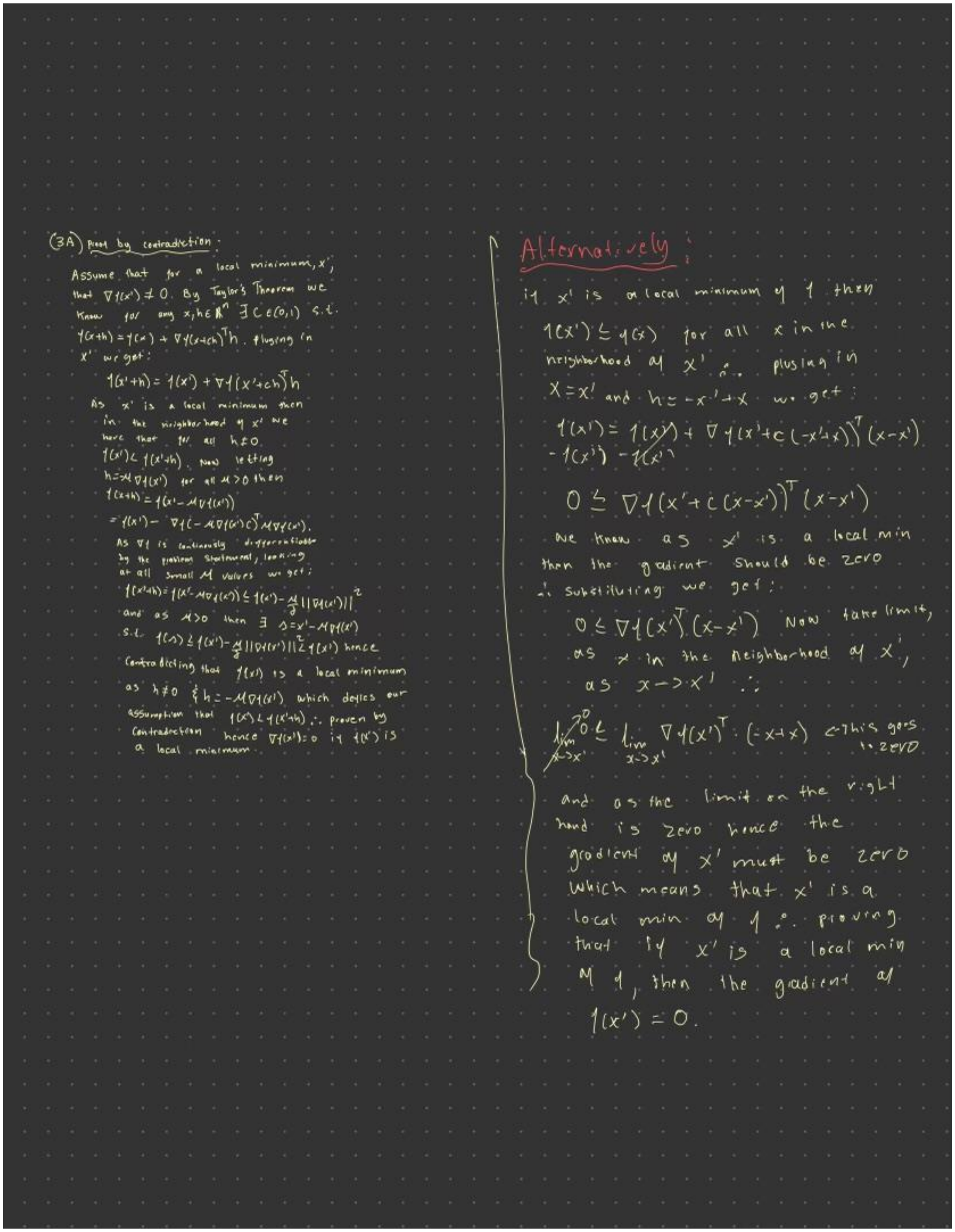
c. The 12 degree model has the highest variance which can as it has the lowest average squared training error (Ein) which implies overfitting meaning that it has the lowest bias and highest variance as it has the lowest average Ein of all the models. Furthermore, as the model is so complex, too complex in this case, it is capturing and fitting noise which further explains why it has the highest variance which can be seen by the extremely high validation error for smaller data points (the highest of the four actually in that region).

d. Looking at the learning curve, we see that for the quadratic model the training error (Ein) seems to reach its max after around 40-ish data points hence adding more training points will not significantly improve Ein, the learning curve. Still, we see that the validation error, Eout, continues to slowly decrease with more data points hence adding more training points would Improve Eout hence improving the model's generalization, but not the learning curve.

e. Training error is lower than validation error because our model is learning on the training data hence it is picking a hypothesis that best fits the training data and then it applies that hypothesis to the validation set where it tries to generalize hence it will always have a lower training error because that is the data it is formulating it's hypothesis around.

f. I think the 6 degree polynomial would perform the best on some unseen data drawn from the same distribution as the training data because its training and validation error converges to the lowest mean squared error with more data points hence it would be best for generalization. Furthermore, after around 35 points, the 6 degree polynomial has the lowest validation error (Eout) hence making it the best for generalization on unseen data drawn from the same distribution as the training data.

3. Stochastic Gradient Descent:

a. 

(3A) Proof by contradiction:

Assume that for a local minimum, x'; that $\nabla f(x') \neq 0$. By Taylor's Theorem we know for any $x, h \in \mathbb{R}^n$ $\exists\, c \in (0,1)$ s.t. $f(x+h) = f(x) + \nabla f(x+ch)^T h$. pluging in x' we get:

$$f(x'+h) = f(x') + \nabla f(x'+ch)^T h$$

As x' is a local minimum then in the neighborhood of x' we have that for all $h \neq 0$ $f(x') < f(x'+h)$. Now letting $h = -\mu \nabla f(x')$ for all $\mu > 0$ then

$$f(x+h) = f(x' - \mu \nabla f(x'))$$
$$= f(x') - \nabla f(-\mu \nabla f(x') c)^T \mu \nabla f(x').$$

As ∇f is continuously differentiable by the problem statement, looking at all small μ values we get:

$$f(x'+h) = f(x' - \mu \nabla f(x')) \leq f(x') - \frac{\mu}{2} \|\nabla f(x')\|^2$$

and as $\mu > 0$ then $\exists\, s = x' - \mu \nabla f(x')$ s.t. $f(s) \leq f(x') - \frac{\mu}{2} \|\nabla f(x')\|^2 < f(x')$ hence contradicting that $f(x')$ is a local minimum as $h \neq 0$ & $h = -\mu \nabla f(x')$ which defies our assumption that $f(x') < f(x'+h)$ $\therefore$ proven by contradiction hence $\nabla f(x') = 0$ if $f(x')$ is a local minimum.

Alternatively:

if x' is a local minimum of f then $f(x') \leq f(x)$ for all x in the neighborhood of x' $\therefore$ plusing in $x = x'$ and $h = -x' + x$ we get:

$$f(x') = \cancel{f(x')} + \nabla f(x' + c(-x'+x))^T (x - x')$$
$$- f(x') - \cancel{f(x')}$$

$$0 \leq \nabla f(x' + c(x-x'))^T (x - x')$$

We know as x' is a local min then the gradient should be zero $\therefore$ substituting we get:

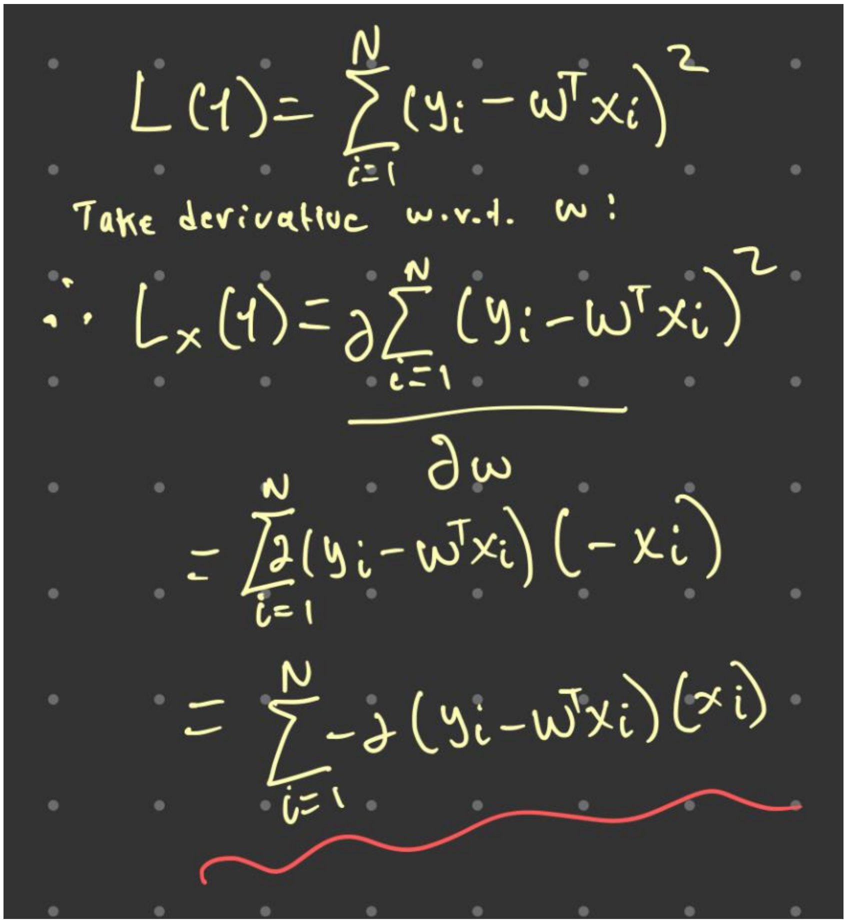
$$0 \leq \nabla f(x')^T (x - x')$$

Now take limit, as x in the neighborhood of x', as $x \to x'$ $\therefore$

$$\cancel{\lim_{x \to x'}} \overset{0}{0} \leq \lim_{x \to x'} \nabla f(x')^T \cdot (-x' + x) \leftarrow \text{This goes to zero}$$

and as the limit on the right hand is zero hence the gradient of x' must be zero which means that x' is a local min of f $\therefore$ proving that if x' is a local min of f, then the gradient of $f(x') = 0$.

b. We could add a constant 1 to **x** as x_0 and add a corresponding w_0 which will equal the bias term, b, to **w**. We would add these to the front of **x** and **w** hence making them the first elements of the **x** and **w** vectors to incorporate the bias.

c. 

$$L(1) = \sum_{i=1}^{N} (y_i - w^T x_i)^2$$

Take derivative w.r.t. w:

$$\therefore L_x(1) = \frac{\partial \sum_{i=1}^{N} (y_i - w^T x_i)^2}{\partial w}$$

$$= \sum_{i=1}^{N} 2(y_i - w^T x_i)(-x_i)$$

$$= \sum_{i=1}^{N} -2(y_i - w^T x_i)(x_i)$$

d. [redacted] **See code for 3 part 1**

e. Despite the starting point, the convergence behavior is always a straight line from all points to the global minimum point hence the convergence behavior is the same no matter the starting point. This behavior is consistent in both dataset 1 and 2.

f. As seen below, as the learning rate increases it takes less epochs for the SGD to converge hence it converges faster as eta increases. This makes sense because for smaller steps it will take longer for the SGD to converge as it is taking smaller stepping sizes hence it will take longer. The graph also shows a decrease in training error with more epochs which makes sense as the weights get better with each iteration along the gradient hence lowering error.

g. [redacted] (3G - 3I). **w** =[-0.22789033, -5.97852063, 3.98841013, -11.85698791, 8.91131758]] was the final weight vector printed out including the bias term which is -0.22789033 as w_0. **See code for 3 part 2**

h. We see that larger values of eta take the least number of epochs to converge which makes sense as the weight changes based on multiples of eta hence the smaller eta is the longer it will take to converge. The graph also shows a decrease in training error with more epochs which makes sense as the weights get better with each iteration along the gradient hence lowering error.

i. With the closed form i got **w** = [-0.31644251, -5.99157048, 4.01509955, -11.93325972, 8.99061096] with a bias term of -0.31644251. The weight vectors are not quite the same, but they are really close.

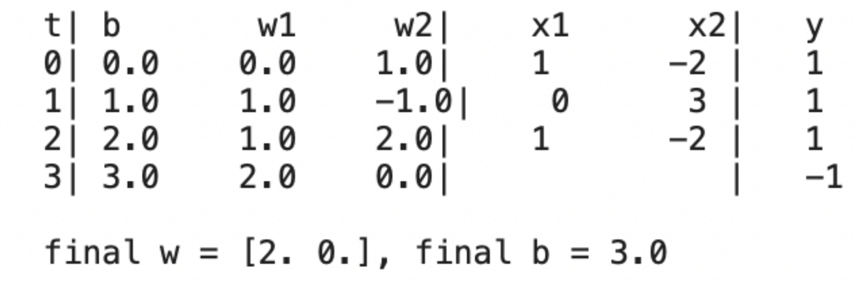
j. Yes because calculating the transpose for an extremely large input matrix, X, can be computationally expensive hence making SGD more practical as it does not do this calculation.. Further with SGD, we can control when we stop hence allowing for us to stop early to save time with extremely large datasets because we are content with good enough or we can choose a much longer stop for more accuracy hence this controllable parameter makes SGD advantages at times.

k. Another stopping condition could be stopping after a certain training error is achieved since as we've seen the training converges after a number of epochs hence we could stop then instead of continuing because any further iteration won't be as impactful in lowering our training error. As the weights are continuously fluctuating in SGD, we could iterate until the change in weight is so small that it's insignificant hence there is no point in continuing to iterate, which was a method we used in CS 156a.

4. The Perceptron

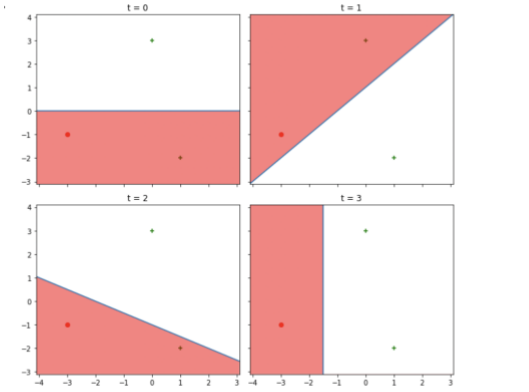
a. [redacted]

See code for 4



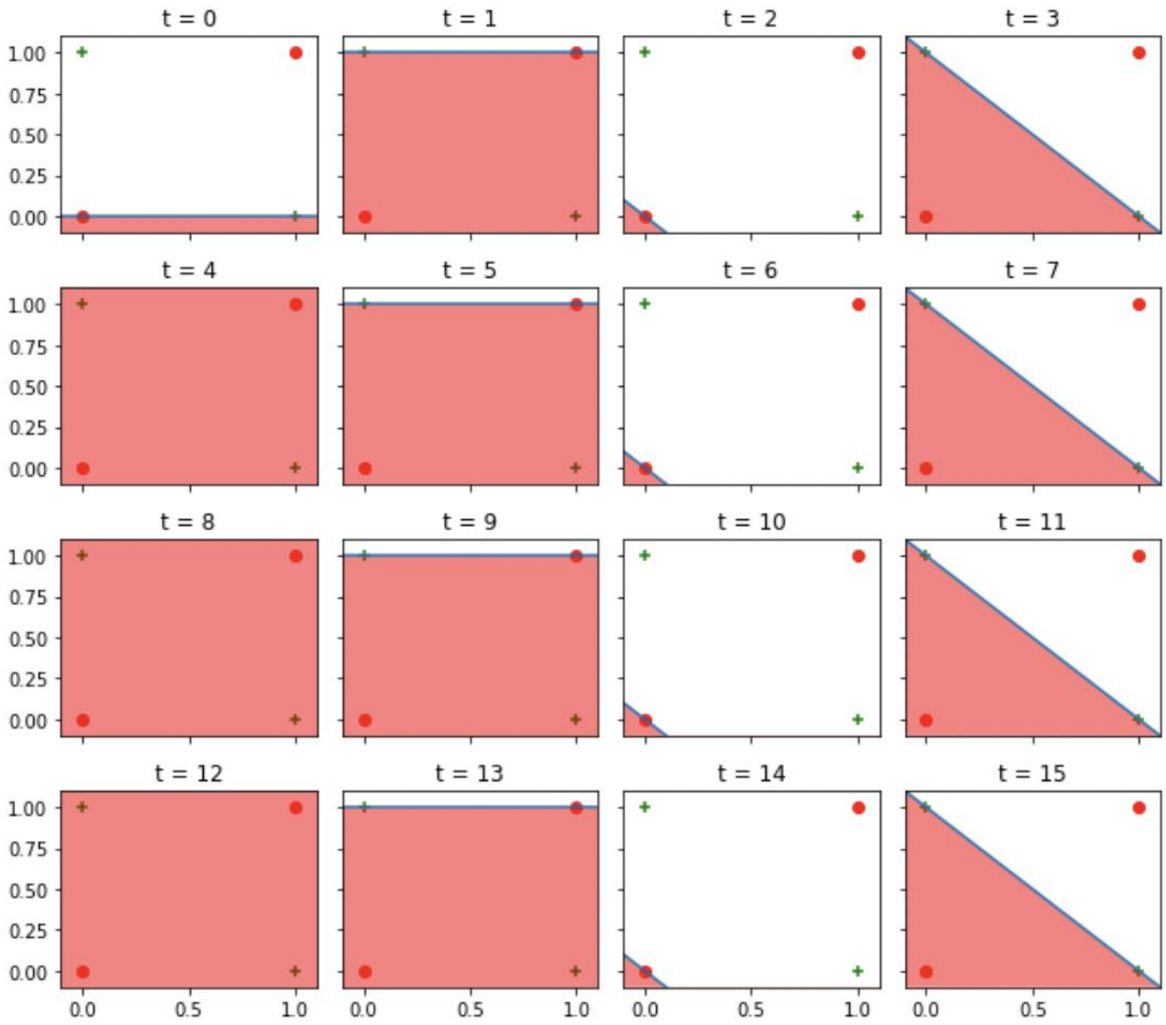
```
t| b       w1      w2|     x1      x2|    y
0| 0.0    0.0    1.0|     1      -2 |    1
1| 1.0    1.0   -1.0|     0       3 |    1
2| 2.0    1.0    2.0|     1      -2 |    1
3| 3.0    2.0    0.0|               |   -1

final w = [2. 0.], final b = 3.0
```



b. We know from 156a that the vc dimension, dvc, is the most points that a hypothesis can shatter independent of the model type, dataset, and target function hence any number of points greater than dvc, a model will fail to shatter them in that dimension meaning that the learning model fails to get all possible dichotomies in that dimension; the minimum number of points required for the model to fail to shatter is the break point. As dvc = dimension + 1 then for our perceptron, the most number of points it can shatter in 2D are 2 + 1 = 3 hence 4 points is it's breakpoint meaning that is the minimum number of points that are not linearly separable in 2D which makes sense if you think of four points {-1, -1, +1, +1} laid out like a cross with the identical points directly across from each other then they are not linearly separable. For 3D dataset then the breakpoint at 5 points because dvc = 3 + 1 = 4. Hence, for N dimensional space, the minimum size of a dataset that is not linearly separable is N+2.

c. No, the perceptron learning algorithm will never converge in this case as it will always try to classify a misclassified point resulting in another misclassified point hence repeating the cycle until the max iterations are reached which forces it to stop instead of going on indefinitely as the data is not linearly separable. For the perceptron to converge on its own, all points must be correctly classified as stated in part A hence it won't converge if the data is not linearly separable.



d. One way that the convergence behavior of the PLA and SGD weight vectors differ is that the SGD weight vector will always converge whereas the perceptron might not always converge as explained in C. Furthermore, whereas the training error continually decreases in SGD, in the PLA, the training error doesn't always decrease as it fluctuates up and down because as PLA is handling one point at a time hence in handling that one point it will disregard all other points hence misclassifying already classified points hence increasing training error.

e.