3.2 Underparameterized Model [1pt]

First consider the underparameterized $d < n$ case. Write down the solution obtained by gradient descent assuming training converges. Show your work. Is the solution unique?

$L = \frac{1}{n}|A|^2, \ A = X\hat{w} - t.$

$\Rightarrow \bar{A} = \frac{2}{n}A$

$\Rightarrow \bar{\hat{w}} = X^T\bar{A} = X^T \cdot \frac{2}{n}A = \frac{2}{n}X^T(X\hat{w} - t)$

Gradient Descent will result in $\bar{\hat{w}} = 0 \Rightarrow X^T X \hat{w} = X^T t$

$\Rightarrow \hat{w}^* = (X^T X)^{-1} X^T t.$

Since $d < n$, then $X^T X$ is invertible, so $\hat{w}^*$ has an explicit formula so it is unique.

3.3.2 [1pt]

Now, let's generalize the previous 2D case to the general $d > n$. Show that gradient descent from zero initialization i.e. $\hat{\mathbf{w}}(0) = 0$ finds a unique minimizer if it converges. Write down the solution and show your work.

See next page.

Solution: $\hat{w} = X^T(XX^T)^{-1}t$

Proof:

$L = \frac{1}{n}\|A\|^2,\ A = X\hat{w} - t.$

$\Rightarrow \overline{A} = \frac{2}{n}A \Rightarrow \overline{\hat{w}} = X^T\overline{A} = X^T \cdot \frac{2}{n}A = \frac{2}{n}X^T(X\hat{w} - t)$ i.e. gradient

Thus, $\hat{w}_{k+1} = \hat{w}_k - \eta \cdot \frac{2}{n} X^T(X\hat{w}_k - t)$ i.e. gradient descent where $\eta > 0$ is learning rate

We'll show two things:

1) Given $\hat{w}_0 = \vec{0}$, $\forall k \in \mathbb{N}, \hat{w}_k = X^T c_k$ for some $c_k \in \mathbb{R}^d$.

2) Given that gradient descent converges, there will be $X\hat{w} - t = 0$

We'll show 1) first by induction:

- Obviously $\hat{w}_0 = X^T\vec{0}$ satisfies the requirement
- Assume that for $k \in \mathbb{N}$, $\hat{w}_k = X^T c_k$ for some $c_k \in \mathbb{R}^n$.

$$\text{Then } \hat{w}_{k+1} = X^T c_k - \eta \cdot \frac{2}{n} X^T(X\hat{w}_k - t)$$

$$= X^T(\underbrace{c_k - \eta \cdot \frac{2}{n} X\hat{w}_k + t}_{= c_{k+1} \in \mathbb{R}^n})$$

So 1) has been shown.

For 2), Since gradient descent converges, then $\lim_{k\to\infty} \hat{w}_{k+1} = \lim_{k\to\infty} \hat{w}_k = \hat{w}$, where $\hat{w}$ is the solution.

Thus, $\lim_{k\to\infty} \hat{w}_{k+1} = \lim_{k\to\infty} \hat{w}_k - \eta \cdot \frac{2}{n} X^T (X \lim_{k\to\infty} \hat{w}_k - t)$

$$\Rightarrow \hat{w} = \hat{w} - \eta \cdot \frac{2}{n} X^T (X\hat{w} - t)$$

$$\Rightarrow X^T (X\hat{w} - t) = 0$$

$$\Rightarrow XX^T (X\hat{w} - t) = 0$$

In the case of $d > n$, XX^T is $n \times n$ thus invertible.

$$\Rightarrow (XX^T)^{-1} (XX^T)(X\hat{w} - t) = (XX^T)^{-1} \vec{0}$$

$$\Rightarrow X\hat{w} - t = 0$$

Combining 1) and 2): Let $\hat{w} = X^T c$, where $\hat{w} = \lim_{k\to\infty} \hat{w}_k$ and $c = \lim_{k\to\infty} c_k$, then:

$$X(X^T c) - t = 0$$

$$\Rightarrow XX^T c = t$$

$$\Rightarrow c = (XX^T)^{-1} t$$

$$\Rightarrow \hat{w} = X^T (XX^T)^{-1} t \text{ is unique}$$

3.3.3 [1pt]

Visualize and compare underparameterized with overparameterized polynomial regression: https://colab.research.google.com/drive/1Atkk9hjUaXV-bDttCxAv9WiMsB6EJTKU. Include your code snippets for the fit_poly function in the write-up. Does overparameterization (higher degree polynomial) always lead to overfitting, i.e. larger test error?

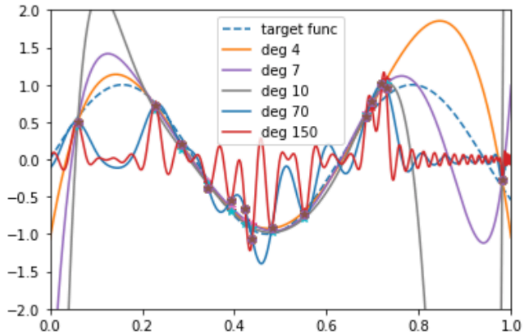
Function Fill In for fit_poly():

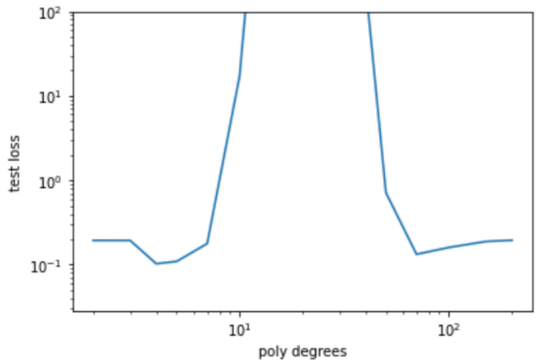
```
def fit_poly(X, d,t):
  X_expand = poly_expand(X, d=d, poly_type = poly_type)
  n = X.shape[0]
  if d > n:
    W = X_expand.T.dot(np.linalg.inv(X_expand.dot(X_expand.T))).dot(t)
  else:
    W = np.linalg.inv(X_expand.T.dot(X_expand)).dot(X_expand.T).dot(t)
  return W
```

Loss Values Output:

```
poly_type = 'chebyshev' # try legendre or chebyshev
loss_val_list = []

poly_degrees = [2, 3, 4, 5, 7, 10, 15, 20,30,50,70,100,150,200]
plot_poly_degress = [4, 7, 10, 70, 150] ## only plot these polynomials
##
plot_target_func()

for d in poly_degrees:
  W = fit_poly(X, d,t)
  plot_flag = True if d in plot_poly_degress else False
  loss_val = plot_prediction(X, W, d, domain, plot_flag)
  loss_val_list.append(loss_val)
plt.legend()

plot_val_loss(poly_degrees, loss_val_list)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:43: UserWarning: Attempted to set no
Invalid limit will be ignored.
```





```
[7] for i in range(len(poly_degrees)):
      print(f"degree = {poly_degrees[i]}, loss = {loss_val_list[i]}")
```

```
degree = 2, loss = 0.19386411705081336
degree = 3, loss = 0.193432748835684
degree = 4, loss = 0.10255222991025262
degree = 5, loss = 0.10915309162008865
degree = 7, loss = 0.177356881829107
degree = 10, loss = 17.094862480309498
degree = 15, loss = 1323020.060699293
degree = 20, loss = 15596274997.211277
degree = 30, loss = 317604.6227527557
degree = 50, loss = 0.7120003515525691
degree = 70, loss = 0.13281043237871068
degree = 100, loss = 0.1599430277105052
degree = 150, loss = 0.18800282052817618
degree = 200, loss = 0.19468029146824442
```

As can be seen, the loss function increases with polynomial degree first, then decreases back to the original level. By comparing degree 7 and degree 70, we see that degree 70 has a smaller loss. This shows that overparametrization ($n=14$, $d=71$) doesn't necessary cause overfitting.