Assignment 1

1.2.1) Underparameterized $d<n$ case: $\min_{\hat{w}} \frac{1}{n}\|X\hat{w}-t\|_2^2 = \min_{\hat{w}} \frac{1}{n}(X\hat{w}-t)^T(X\hat{w}-t)$

Minimizing: $\frac{2}{n}(X\hat{w}-t)^T X = 0$

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

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

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

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

∴ this solution is the optimal solution achieved by the network assuming the training converges, in which case the gradient descent result approximates this true optima.

1.2.2) $\varepsilon_i \sim \mu=0, \sigma^2$, $t_i = \omega^{*T} x_i + \varepsilon_i \rightarrow t = X\omega^* + \varepsilon \rightarrow \varepsilon$: independent RV

$L = \frac{1}{n}\|X\hat{w}-t\|_2^2$

$L = \frac{1}{n}\|X(X^TX)^{-1}X^Tt - t\|_2^2$

$L = \frac{1}{n}\|X(X^TX)^{-1}X^T(X\omega^*+\varepsilon) - X\omega^* - \varepsilon\|_2^2$

$L = \frac{1}{n}\|X\omega^* + X(X^TX)^{-1}X^T\varepsilon - X\omega^* - \varepsilon\|_2^2$

$L = \frac{1}{n}\|(X(X^TX)^{-1}X^T - I)\varepsilon\|_2^2$

Expectation of training loss wrt n, d, σ:

$E[L] = E\left[\frac{1}{n}\|(X(X^TX)^{-1}X^T-I)\varepsilon\|_2^2\right]$

$E[L] = E\left[\frac{1}{n}((X(X^TX)^{-1}X^T-I)\varepsilon)^T((X(X^TX)^{-1}X^T-I)\varepsilon)\right]$

$E[L] = \frac{1}{n}E\left[\varepsilon^T(X(X^TX)^{-1}X^T-I)^T(X(X^TX)^{-1}X^T-I)\varepsilon\right]$

Trace of 1x1: $tr(E[L]) = \frac{1}{n}tr\left(E\left[\varepsilon^T(X(X^TX)^{-1}X^T-I)^T(X(X^TX)^{-1}X^T-I)\varepsilon\right]\right)$

Expectation ↔ trace: $E[L] = \frac{1}{n}E\left[tr\left(\left[\varepsilon^T(X(X^TX)^{-1}X^T-I)^T(X(X^TX)^{-1}X^T-I)\varepsilon\right]\right)\right]$

Cyclic property: $E[L] = \frac{1}{n}E\left[tr\left(\left[\varepsilon^T\varepsilon(X(X^TX)^{-1}X^T-I)^T(X(X^TX)^{-1}X^T-I)\right]\right)\right]$

Expectation ↔ trace: $E[L] = \frac{1}{n}tr\left(E[\varepsilon^T\varepsilon]E\left[(X(X^TX)^{-1}X^T-I)^T(X(X^TX)^{-1}X^T-I)\right]\right)$

Variance: $E[L] = \frac{1}{n}tr\left(E[(\varepsilon-0)^T(\varepsilon-0)]E\left[(X(X^TX)^{-1}X^T-I)^T(X(X^TX)^{-1}X^T-I)\right]\right)$

Expectation of scalar: $E[L] = \frac{1}{n}\sigma^2 tr\left((X(X^TX)^{-1}X^T-I)^T(X(X^TX)^{-1}X^T-I)\right)$

$E[L] = \frac{\sigma^2}{n}tr\left[\|(X(X^TX)^{-1}X^T-I)\|_2^2\right]$

$E[L] = \frac{\sigma^2}{n}tr\left[\|-(I-X(X^TX)^{-1}X^T)\|_2^2\right]$

$I - X(X^TX)^{-1}X^T$ is symmetric →

$$E[L] = \frac{\sigma^2}{n} tr\left[\|I - X(X^TX)^{-1}X^T\|_2^2\right]$$
$$E[L] = \frac{\sigma^2}{n} tr\left[(I - X(X^TX)^{-1}X^T)^2\right]$$
$$E[L] = \frac{\sigma^2}{n} tr\left(I - X(X^TX)^{-1}X^T\right)$$
$$E[L] = \frac{\sigma^2}{n}(n - rank(X))$$
$$E[L] = \frac{\sigma^2}{n}(n-d)$$

$X(X^TX)^{-1}X^T$ is idempotent and

Idempotent: $(I-A)^2 = I - 2A + A^2 = I - 2A + A = I - A$

trace of projection matrix

$d < n$ so $rank(X) = d$ since X is full rank

1.3.1) $d > n$ overparametrized case, $n = 1$, $d = 2$

$X_1 = [1\ 1]$ $t_1 = 3$ $\hat{w}^T x_1 = t_1$

$$\hat{w}^T \begin{bmatrix} 1 \\ 1 \end{bmatrix} = t_1$$
$$w_1 + w_2 = 3$$
$$w_1 = -w_2 + 3$$

→ infinitely many solutions as w_2 is a free variable constraining w_1 on a line. This equation can be written wrt w_1 as the free variable as well.

1.3.2) $d > n$, $w(0) = 0$

$\nabla_w L$ is minimized when $XX^Ta - t = 0$

$$\nabla_w L = \frac{1}{n} X^T(X\hat{w} - t)$$
$$\nabla_w L = \frac{1}{n} X^T(X(X^Ta) - t)$$
$$\nabla_w L = \frac{1}{n} X^T(XX^Ta - t)$$

$$t = XX^Ta$$
$$(XX^T)^{-1}t = a$$

$$\therefore \hat{w} = X^Ta = X^T(XX^T)^{-1}t \text{ at convergence}$$

Gradient descent:

$$\hat{w}_1 = \hat{w}(0) - \alpha \nabla_w L$$
$$\hat{w}_1 = -\frac{\alpha}{n} X^T(XX^Ta - t)$$
$$\hat{w}_2 = \hat{w}_1 - \alpha \nabla_w L(\hat{w}_1)$$
$$\vdots$$
$$\hat{w}_{k+1} = \hat{w}_k - \alpha \nabla_w L(\hat{w}_k)$$

If it converges, $\nabla_w L(\hat{w}_k) = 0$: $\hat{w}_{k+1} = \hat{w}_k \longrightarrow \hat{w}_{k+1} = \hat{w}_k - \frac{\alpha}{n} X^T(X\hat{w}_k - t)$

$$0 = -\frac{\alpha}{n} X^T(X\hat{w}_k - t)$$
$$0 = X^TX\hat{w}_k - X^Tt$$

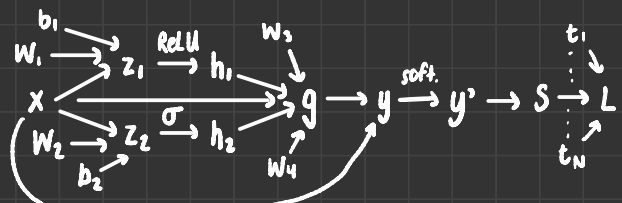
→ solution exists!

With convergence, $\hat{w}_k = \hat{w} = X^T(XX^T)^{-1}t$ so:

$$0 = X^TX(X^T(XX^T)^{-1}t) - X^Tt$$
$$0 = X^T(XX^T)(XX^T)^{-1}t - X^Tt$$
$$0 = X^TIt - X^Tt = 0$$

$\therefore$ with $\hat{w}(0)=0$ and convergence, we get $\hat{w}=X^T(XX^T)^{-1}t$ which is a unique minimizer from gradient descent results. Note that $(X^TX)^{-1}$ is not possible here since $d>n$ so $\text{rank}(X)=n<d$ (X is rank deficient for $X^TX\in R^{d\times d}$, needed $\text{rank}(X)=d$). This is why $XX^T\in R^{n\times n}$ was needed. **Note: 1.3.4 attached at the end of the PDF**

2.1.1)



N classes → $y\in R^{N\times 1}$

2.1.2) $\frac{\partial L}{\partial S}=-1$ Variable vectors and error vectors are column, Jacobian matrices follow num. not.

Vectorize S: $S=\sum_{k=1}^{N} I(t=k)\log(y'_k)=I(t)^T\log(y')$ where $I(t)$ is one hot encoding vector is 0 or 1 depending on if we're at the target class index or not

$\frac{\partial L}{\partial y'}=\frac{\partial L}{\partial S}\left(\frac{\partial S}{\partial y'}\right)=-I(t)\circ\frac{1}{y'}$ → gives ∂L wrt each y_k as jacobian

$\frac{\partial L}{\partial y}=\frac{\partial L}{\partial y'}\left(\frac{\partial y'}{\partial y}\right)=\left(-I(t)\circ\frac{1}{y'}\right)\text{softmax}'(y)=\bar{y}'\,\text{softmax}'(y)$ $n\times 1$

$\frac{\partial L}{\partial g}=\left(\frac{\partial y}{\partial g}\right)^T\left(\frac{\partial L}{\partial y}\right)=W_3^T\bar{y}$ $(n\times m)^T\ (n\times 1)$

$\frac{\partial L}{\partial h_2}=\frac{\partial L}{\partial g}\left(\frac{\partial g}{\partial h_2}\right)=\bar{g}\circ h_1$ since $g=h_1\circ h_2$, g, h_1, h_2 are same shape

$\frac{\partial L}{\partial h_1}=\frac{\partial L}{\partial g}\left(\frac{\partial g}{\partial h_1}\right)=\bar{g}\circ h_2$

$\frac{\partial L}{\partial z_1}=\left(\frac{\partial L}{\partial h_1}\right)\frac{\partial h_1}{\partial z_1}=\bar{h}_1\circ\text{ReLU}'(z_1)$

$\frac{\partial L}{\partial z_2}=\left(\frac{\partial L}{\partial h_2}\right)\frac{\partial h_2}{\partial z_2}=\bar{h}_2\circ\sigma'(z_2)$

$\frac{\partial L}{\partial x}=\left(\frac{\partial z_1}{\partial x}\right)^T\frac{\partial L}{\partial z_1}+\left(\frac{\partial z_2}{\partial x}\right)^T\frac{\partial L}{\partial z_2}+\left(\frac{\partial y}{\partial x}\right)^T\frac{\partial L}{\partial y}$ $(n\times m)^T(n\times 1)$

$\frac{\partial L}{\partial x}=W_1^T\bar{z}_1+W_2^T\bar{z}_2+W_4^T\bar{y}$

2.2.1) $x=\begin{bmatrix}1\\3\\1\end{bmatrix}$ $\bar{y}^T=\frac{\partial L}{\partial y}=[1\ 1\ 1]$

$z=W_1x$

$z=\begin{bmatrix}1&2&1\\-2&1&0\\1&-2&-1\end{bmatrix}\begin{bmatrix}1\\3\\1\end{bmatrix}$

$z=\begin{bmatrix}8\\1\\-6\end{bmatrix}$ → $h=\text{ReLU}(z)$

$h=\max(0,z)$

$h=\begin{bmatrix}8\\1\\0\end{bmatrix}$

$\frac{\partial L}{\partial h}=\frac{\partial L}{\partial y}\left(\frac{\partial y}{\partial h}\right)$

$\frac{\partial L}{\partial h}=\bar{y}^TW_2$

$\frac{\partial L}{\partial h}=[1\ 1\ 1]\begin{bmatrix}-2&4&1\\1&-2&-3\\-3&4&6\end{bmatrix}$

$\frac{\partial L}{\partial h}=[-4\ 6\ 4]$

$\frac{\partial L}{\partial W_1}=\left(\frac{\partial L}{\partial z}\right)^T\left(\frac{\partial z}{\partial W_1}\right)^T$ $(1\times 3)^T$ $(3\times 1)^T$

$\frac{\partial L}{\partial W_1}=\bar{z}x^T$

$\frac{\partial L}{\partial W_2}=\frac{\partial L}{\partial y}\left(\frac{\partial y}{\partial W_2}\right)^T$ 3×3 $\bar{y}_{3\times 1}$ $(3\times 1)^T$

$\frac{\partial L}{\partial W_2}=\bar{y}h^T$

$\frac{\partial L}{\partial W_2}=\begin{bmatrix}1\\1\\1\end{bmatrix}[8\ 1\ 0]$

$\frac{\partial L}{\partial W_2}=\begin{bmatrix}8&1&0\\8&1&0\\8&1&0\end{bmatrix}$

$\frac{\partial L}{\partial z}=\frac{\partial L}{\partial h}\left(\frac{\partial h}{\partial z}\right)$

$\frac{\partial L}{\partial z}=\bar{h}\circ\text{ReLU}'(z)$

$\frac{\partial L}{\partial z}=[-4\ 6\ 4]\circ[1\ 1\ 0]$

$\bar{z}^T=\frac{\partial L}{\partial z}=[-4\ 6\ 0]$ ↑ following question format

$\frac{\partial L}{\partial W_1}=\begin{bmatrix}-4\\6\\0\end{bmatrix}[1\ 3\ 1]$

$\frac{\partial L}{\partial W_1}=\begin{bmatrix}-4&-12&-4\\6&18&6\\0&0&0\end{bmatrix}$

$\left\| \frac{\partial L}{\partial W_1} \right\|_F^2 = \text{trace}\left(\frac{\partial L}{\partial W_1}^T \frac{\partial L}{\partial W_1} \right)$

$= \text{trace}\left(\begin{bmatrix} -4 & 6 & 0 \\ -12 & 18 & 0 \\ -4 & 6 & 0 \end{bmatrix} \begin{bmatrix} -4 & -12 & -4 \\ 6 & 18 & 6 \\ 0 & 0 & 0 \end{bmatrix} \right)$

$= \text{trace}\left(\begin{bmatrix} 52 & 156 & 52 \\ 156 & 468 & 156 \\ 52 & 156 & 52 \end{bmatrix} \right)$

$= 572$

$\left\| \frac{\partial L}{\partial W_2} \right\|_F^2 = \text{trace}\left(\begin{bmatrix} 8 & 8 & 8 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 8 & 1 & 0 \\ 8 & 1 & 0 \\ 8 & 1 & 0 \end{bmatrix} \right)$

$= \text{trace}\left(\begin{bmatrix} 192 & 24 & 0 \\ 24 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right)$

$= 195$

2.2.2) $\left\| \frac{\partial L}{\partial W_1} \right\|_F^2 = \|x\|_2^2 \|\bar{z}\|_2^2$

$= \|(52)\|$

$= 572$

$\left\| \frac{\partial L}{\partial W_2} \right\|_F^2 = \text{trace}(h\bar{y}^T \bar{y}h^T)$

$= \text{trace}(\bar{y}^T \bar{y} h^T h)$

$= \|\bar{y}\|_2^2 \|h\|_2^2$

$= 3(65)$

$= 195$

→ Note: since I used $\frac{\partial L}{\partial W_1} = \bar{z}x^T$ and $\frac{\partial L}{\partial W_2} = \bar{y}h^T$ in prev. q., I kept it consistent here since final answer will be the same

$\therefore$ same answers as before.

2.2.3)

- Naive approach: forward pass on batch, store gradients for each input in batch independently, backprop for each input/sample independently.
- Efficient approach: forward pass on batch, compute and store gradients for all samples simultaneously in one backward pass (parallelized)

	T (Naive)	T (Efficient)	M (Naive)	M (Efficient)
Forward Pass	$D^2 \times N \times K$	$D^2 \times N \times K$	$O(K \times D^2)$	$O(D^2)$
Backward Pass	$2 \times D^2 \times N \times K$	$2 \times D^2 \times N \times K$	$O(K \times D^2)$	$O(D^2)$
Gradient Norm Computation	$D^2 \times K$	$2D \times K$	$O(K \times D^2)$	$O(K \times D^2)$

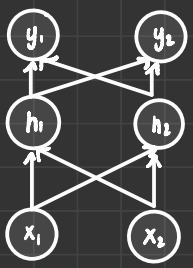
$y = Wx$ → weight memory dominates

$\sim D^2$ gradient info used for norm

- K-1 hidden layers, K weight matrices, all units with dimension D. N input vectors ($X \in \mathbb{R}^{N \times D}$)
- $N \times D$ input vector params
- Weight matrix will be $D \times D$ since all units have dim. D, $K \times D \times D$ parameters for K layer network
- Forward: D^2 mult. per layer, per input.
- Backward: 2 mult. per weight to get $\bar{w}_{ij}$ and $\bar{h}_i$. $\therefore$ 2 × the cost of forward pass
- Naive memory: storing parameter information for each element available among all layers (× K) since each input sample is backpropped individually.
- Efficient memory: only need to store parameter information at current layer since computations are parallized over all samples.
- Gradient norm computation is done for each parameter gradient vector: $\frac{\partial L}{\partial W}$

- Naive Grad Norm: $\sum_i^N \sum_j^N |a_{ij}|^2$ → for D^2 weights, we have one mult. operation for each. Then, since we need $D^2 \times K$ weight entries for the norm, memory is dominated by this.
- Efficient Grad Norm: $(x^T x)(\vec{z}^T \vec{z})$ → $x^T x$ is D mult., same with $\vec{z}^T \vec{z}$, so $2D$ mult. per $\frac{\partial L}{\partial W_i}$. This gives $2D \times K$ cost and $O(K \times D^2)$ memory for storing and using all layer weight matrices to compute all norms.

3.1)



$y_1 \leq y_2$

$\max(x_1, x_2) = \frac{1}{2}(x_1 + x_2) + \frac{1}{2}|x_1 - x_2|$ $\quad \min(x_1, x_2) = \frac{1}{2}(x_1 + x_2) - \frac{1}{2}|x_1 - x_2|$

$W_1 = \begin{bmatrix} 0.5 & 0.5 \\ 1 & -1 \end{bmatrix}$ $\quad W_1 x = \begin{bmatrix} 0.5x_1 + 0.5x_2 \\ x_1 - x_2 \end{bmatrix}$ $\quad \phi_1(z) = |z|$ $\quad b_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

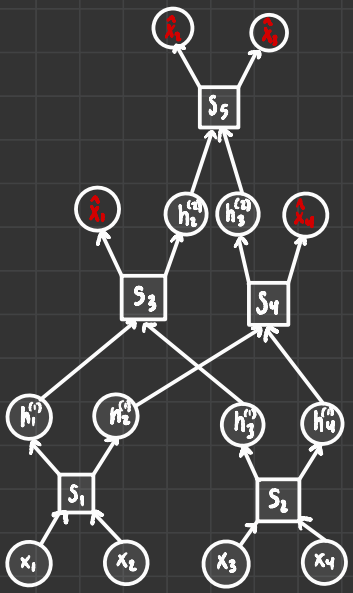
p would need explicit $\phi'(0) = 1$ for grad descent

$W_2 = \begin{bmatrix} 1 & -0.5 \\ 1 & 0.5 \end{bmatrix}$ $\quad W_2 h = \begin{bmatrix} \frac{1}{2}(x_1 + x_2) - \frac{1}{2}|x_1 - x_2| \\ \frac{1}{2}(x_1 + x_2) + \frac{1}{2}|x_1 - x_2| \end{bmatrix}$ $\quad b_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\quad \phi_2(z) = z$

3.2) $x_1 \leq x_2 \leq x_3 \leq x_4$

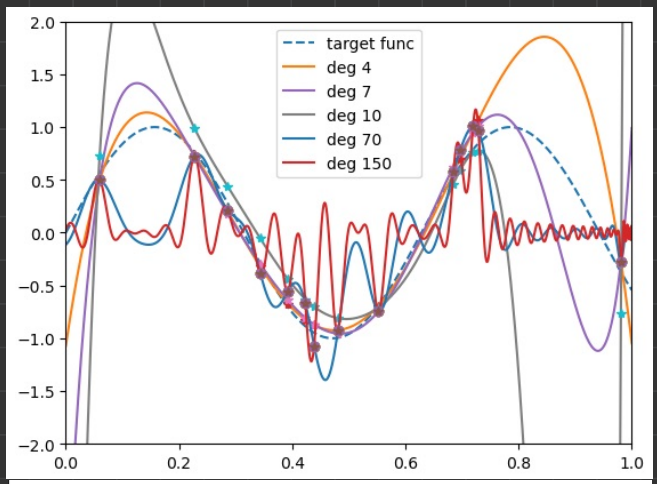
Merge sort: take the 4 numbers in a list, divide into two lists, divide again into single elements, merge elements into two sorted subarrays, then merge sorted subarrays to reconstruct full list

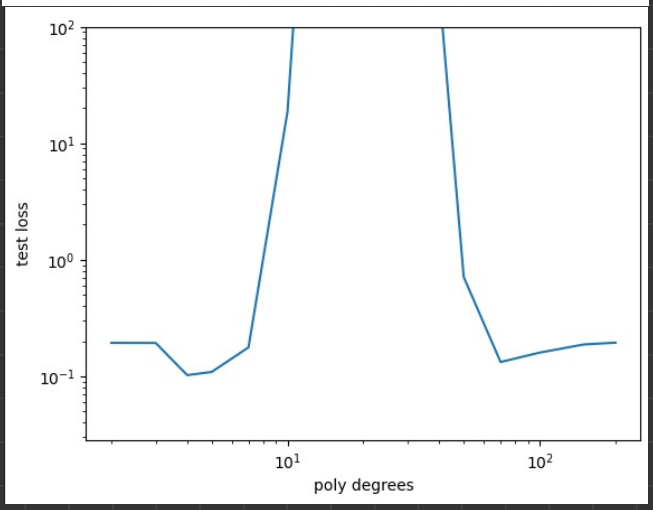
x_1 cmp x_2, x_3 cmp x_4, $h_1^{(1)}$ cmp $h_3^{(1)}$ (smallest), $h_2^{(1)}$ cmp $h_4^{(1)}$ (largest), $h_2^{(2)}$ cmp $h_3^{(2)}$



1.3.4) Snippets from .ipynb :

```python
def fit_poly(X, d,t):
  X_expand = poly_expand(X, d=d, poly_type = poly_type)
  n = X.shape[0]
  if d > n:
    ## Solution for Part 1.3.2: overparameterized case
    W = X_expand.T @ linalg.inv(X_expand @ X_expand.T) @ t
  else:
    ## Solution for Part 1.2.1: underparameterized case
    W = linalg.inv(X_expand.T @ X_expand) @ X_expand.T @ t
  return W
```





The best dimension size (polynomial degree) is d=4, where the loss was 0.10255222991253363. Since n = 14, the underparameterized case works better for fitting to the target sine function. The overparameterized cases lead to overfitting as they fit more exactly to the data points, but the loss graph shows that from d=50 onwards, the regression fit is so exact to the data points that the error steadies back down to within a reasonable range. The loss grows exponentially at d=15, when the problem becomes overparameterized, resulting in the parabolic loss behaviour shown above. However, the overparameterized cases in general do not model the behaviour of the target function well, even though the loss comes back down to a lower value from d=50 onwards. Thus, the underparameterized cases work better for fitting the function in terms of generalizability to the overall behaviour of the function, and in terms of achieving the minimal loss.