CSC413 Homework 1 January 28, 2021

1. Hard-Coding Networks

1.1. Verify Element in List

According to the architecture, we can set the first layer to compare whether x_1 and x_4, x_2 and x_4, and x_3 and x_4 are same. Since the $x_i \in \mathbb{Z}$ for $i = 1, 2, 3, 4$, the difference between two doubled integers should at least greater than 2 if they are not same. Thus, we can set $\boldsymbol{W}^{(1)}$ and $\boldsymbol{b}^{(1)}$ be

$$\boldsymbol{W}^{(1)} = \begin{pmatrix} 2 & 0 & 0 & -2 \\ 0 & 2 & 0 & -2 \\ 0 & 0 & 2 & -2 \end{pmatrix}$$
$$\boldsymbol{b}^{(1)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Thus, for the second layer, we should let the output be 1 if 1 is the output of h_1, h_2, or h_3. Thus, we can set $\boldsymbol{W}^{(2)}$ and $\boldsymbol{b}^{(2)}$ be

$$\boldsymbol{W}^{(2)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$
$$\boldsymbol{b}^{(2)} = -2$$

1.2. Verify Permutation

Since the elements are distinct integers, I want to implement a neural network that contains one hidden layer.

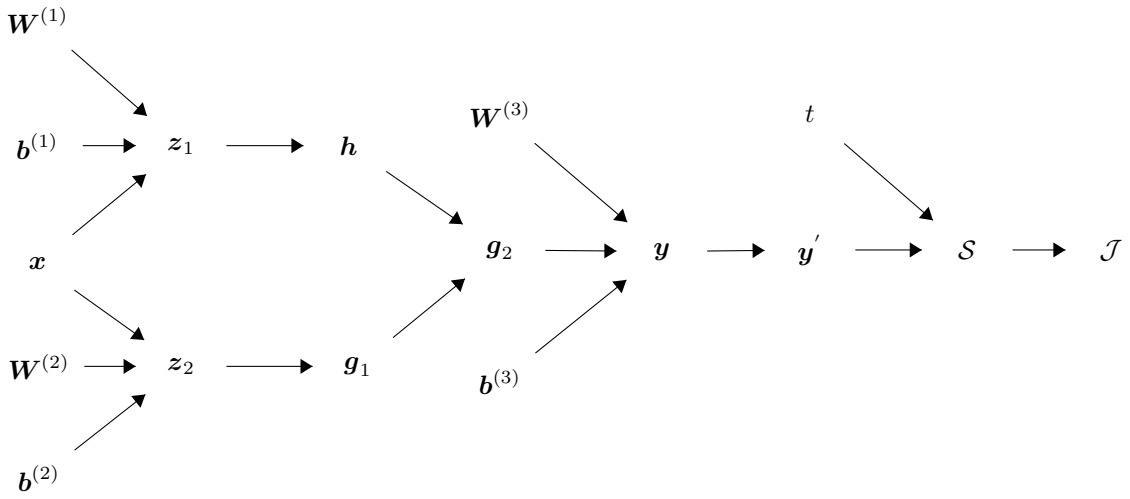
The inputs are $x_1, x_2, \ldots, x_6$. For the hidden layer, since there are 6 different permutations for a list of length 3, we set 6 outputs $h_1, h_2, \ldots, h_6$, where each unit checks whether (x_4, x_5, x_6) is one of the permutations of (x_1, x_2, x_3). It can be checked by setting the threshold for each element based on its index, and comparing the difference of one of the permutations of (x_1, x_2, x_3) and (x_4, x_5, x_6). For the activation function, we can set the output is 1 if the difference is 0. Otherwise, the output should be 0.

Then, if at least one hidden unit is 1 from the hidden layer, we can declare that (x_4, x_5, x_6) is the permutation of (x_1, x_2, x_3). Thus, we do not need other layer and can direct connects them to the output y, since we can use the activation function in 1.1. The weight would be a 6×1 $\mathbf{1}$ matrix and the bias would be 0. Thus, we can set the output to be 1 if there exists the output 1 from the hidden layer. Otherwise, the output should be 0.

2. Backpropagation

2.1. Neural Network

2.1.1.



2.1.2. According to the equation from 2.1, I can get:

$$\bar{\mathcal{J}} = \frac{\partial \mathcal{J}}{\partial \mathcal{J}} = 1$$

$$\bar{\mathcal{S}} = \frac{\partial \mathcal{J}}{\partial \mathcal{S}} = \frac{\partial}{\partial \mathcal{S}}(-\mathcal{S}) = -1$$

$$\bar{\boldsymbol{y}'} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{y}'} = \frac{\partial \mathcal{J}}{\partial \mathcal{S}}\frac{\partial \mathcal{S}}{\partial \boldsymbol{y}'} = \frac{1}{\boldsymbol{y}'}\bar{\mathcal{S}}\begin{pmatrix}0\\ \vdots\\ 0\\ 1\\ 0\\ \vdots\\ 0\end{pmatrix} \qquad \text{where 1 is at the } t^{th} \text{ index}$$

$$\bar{\boldsymbol{y}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{y}} = \frac{\partial \mathcal{J}}{\partial \mathcal{S}}\frac{\partial \mathcal{S}}{\partial \boldsymbol{y}'}\frac{\partial \boldsymbol{y}'}{\partial \boldsymbol{y}} = \bar{\boldsymbol{y}'}\text{softmax}'(\boldsymbol{y}) = xxx$$

$$\bar{\boldsymbol{g}_2} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{g}_2} = \frac{\partial \mathcal{J}}{\partial \mathcal{S}}\frac{\partial \mathcal{S}}{\partial \boldsymbol{y}'}\frac{\partial \boldsymbol{y}'}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{g}_2} = \boldsymbol{W}^{(3)^T}\bar{\boldsymbol{y}}$$

$$\bar{\boldsymbol{g}_1} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{g}_1} = \frac{\partial \mathcal{J}}{\partial \mathcal{S}}\frac{\partial \mathcal{S}}{\partial \boldsymbol{y}'}\frac{\partial \boldsymbol{y}'}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{g}_2}\frac{\partial \boldsymbol{g}_2}{\partial \boldsymbol{g}_1} = \bar{\boldsymbol{g}_2} \odot (h \odot \mathbf{1}) = \bar{\boldsymbol{g}_2} \odot h$$

$$\bar{\boldsymbol{h}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{h}} = \frac{\partial \mathcal{J}}{\partial \mathcal{S}}\frac{\partial \mathcal{S}}{\partial \boldsymbol{y}'}\frac{\partial \boldsymbol{y}'}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{g}_2}\frac{\partial \boldsymbol{g}_2}{\partial \boldsymbol{h}} = \bar{\boldsymbol{g}_2} \odot (\mathbf{1} \odot \boldsymbol{g}_1) = \bar{\boldsymbol{g}_2} \odot \boldsymbol{g}_1$$

$$\bar{\boldsymbol{z}_2} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{z}_2} = \frac{\partial \mathcal{J}}{\partial \mathcal{S}}\frac{\partial \mathcal{S}}{\partial \boldsymbol{y}'}\frac{\partial \boldsymbol{y}'}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{g}_2}\frac{\partial \boldsymbol{g}_2}{\partial \boldsymbol{g}_1}\frac{\partial \boldsymbol{g}_1}{\partial \boldsymbol{z}_2} = \bar{\boldsymbol{g}_1}\sigma'(\boldsymbol{z}_2) = \bar{\boldsymbol{g}_1}(\frac{e^{-z}}{1+e^{-z}})^2$$

$$\bar{\boldsymbol{z}_1} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{z}_1} = \frac{\partial \mathcal{J}}{\partial \mathcal{S}}\frac{\partial \mathcal{S}}{\partial \boldsymbol{y}'}\frac{\partial \boldsymbol{y}'}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{g}_2}\frac{\partial \boldsymbol{g}_2}{\partial \boldsymbol{h}}\frac{\partial h}{\partial \boldsymbol{z}_1} = \bar{\boldsymbol{h}} \odot \text{ReLU}'(\boldsymbol{z}_1) = \bar{\boldsymbol{h}} \odot \begin{pmatrix}\vdots\\ a_i\\ \vdots\end{pmatrix} \qquad a_i = 0 \text{ if } z_i \leq 0;\ 1 \text{ if } z_i > 0$$

$$\bar{\boldsymbol{x}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{x}} = \frac{\partial \mathcal{J}}{\partial \mathcal{S}}\frac{\partial \mathcal{S}}{\partial \boldsymbol{y}'}\frac{\partial \boldsymbol{y}'}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{g}_2}\frac{\partial \boldsymbol{g}_2}{\partial \boldsymbol{h}}\frac{\partial h}{\partial \boldsymbol{z}_1}\frac{\partial \boldsymbol{z}_1}{\partial x} + \frac{\partial \mathcal{J}}{\partial \mathcal{S}}\frac{\partial \mathcal{S}}{\partial \boldsymbol{y}'}\frac{\partial \boldsymbol{y}'}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{g}_2}\frac{\partial \boldsymbol{g}_2}{\partial \boldsymbol{h}}\frac{\partial h}{\partial \boldsymbol{z}_2}\frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{x}} = \boldsymbol{W}^{(1)^T}\bar{\boldsymbol{z}_1} + \boldsymbol{W}^{(2)^T}\bar{\boldsymbol{z}_2}$$

2.2. Automatic Differentiation

2.2.1. Since $\boldsymbol{H}$ is the second derivative of $\mathcal{L}$ with respect to $\boldsymbol{x}$, we can get

$$\begin{aligned}\boldsymbol{H} &= \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{x}^2} \\ &= \frac{\partial}{\partial \boldsymbol{x}}(2\boldsymbol{v}\boldsymbol{v}^T\boldsymbol{x}) \\ &= 2\boldsymbol{v}\boldsymbol{v}^T \\ &= 2\begin{pmatrix}4\\2\\3\end{pmatrix}\begin{pmatrix}4 & 2 & 3\end{pmatrix} \\ &= \begin{pmatrix}32 & 16 & 24\\16 & 8 & 12\\24 & 12 & 18\end{pmatrix}\end{aligned}$$

2.2.2. We know that $\boldsymbol{H} \in \mathbb{R}^{n\times n}$ and it is symmetric, it's easy to get that the memory cost of computing the Hessian is $\mathcal{O}(n^2)$. For the scalar multiplications, since we get $\boldsymbol{H} = 2\boldsymbol{v}\boldsymbol{v}^T$ and $\boldsymbol{v} \in \mathbb{R}^{n\times 1}$, it requires $2 \times n \times n$ multiplications at most. Thus, the number of scalar multiplications of computing the Hessian is $\mathcal{O}(n^2)$.

2.3. Vector-Hessian Products First, I want to compute $\boldsymbol{z}$ in forward-mode.

$$\begin{aligned}\boldsymbol{z} &= \boldsymbol{H}\boldsymbol{y} \\ &= \boldsymbol{v}\boldsymbol{v}^T\boldsymbol{y} \\ &= \begin{pmatrix}1\\2\\3\end{pmatrix}\begin{pmatrix}1 & 2 & 3\end{pmatrix}\begin{pmatrix}1\\1\\1\end{pmatrix} \\ &= \begin{pmatrix}1 & 2 & 3\\2 & 4 & 6\\3 & 6 & 9\end{pmatrix}\begin{pmatrix}1\\1\\1\end{pmatrix} \\ &= \begin{pmatrix}6\\12\\18\end{pmatrix}\end{aligned}$$

Thus, $\boldsymbol{z}^T = \begin{pmatrix}6 & 12 & 18\end{pmatrix}$. For the backpropagation, we have

$$\begin{aligned}\boldsymbol{z} &= \boldsymbol{v}\boldsymbol{M} \\ &= \begin{pmatrix}1\\2\\3\end{pmatrix}\left(\begin{pmatrix}1 & 2 & 3\end{pmatrix}\begin{pmatrix}1\\1\\1\end{pmatrix}\right) \\ &= 6\begin{pmatrix}1\\2\\3\end{pmatrix} \\ &= \begin{pmatrix}6\\12\\18\end{pmatrix}\end{aligned}$$

Thus, $\boldsymbol{z}^T = \begin{pmatrix}6 & 12 & 18\end{pmatrix}$. For the forward-mode calculation, since it computes $\boldsymbol{H}$ first, the time and memory cost for computing $\boldsymbol{z}$ is at least $\mathcal{O}(n^2)$ according to 2.2.2. Since $\boldsymbol{y} \in \mathbb{R}^{n\times 1}$, the memory cost depends on n, and it only takes linear time to compute $\boldsymbol{H}\boldsymbol{y}$. Thus, the time and memory cost of evaluating $\boldsymbol{z}$ in forward-mode is $\mathcal{O}(n^2)$.

For the backpropagation, since it computes $\boldsymbol{M}$ first, the time and memory cost for computing $\boldsymbol{v}^T\boldsymbol{y}$ is linear in terms of n according to their dimensions. Since $\boldsymbol{M}$ is a scalar, the time and memory cost for computing $\boldsymbol{v}\boldsymbol{M}$ is also linear in terms of n. Thus, we can conclude that the time and memory cost of evaluating $\boldsymbol{z}$ with backpropagation is $\mathcal{O}(2n)$, which is $\mathcal{O}(n)$.

2.4. Trade-off of Reverse- and Forward-mode Autodiff

Let's consider the forward-mode calculation first. Notice that we know the time and memory complexity of computing $\boldsymbol{H}$ is $\mathcal{O}(n^2)$ by 2.2.2.. Since $\boldsymbol{y}_1 \in \mathbb{R}^{n\times 1}$, multiplying a $n \times n$ matrix with a $n \times 1$ matrix requires n^2 scalar multiplications. For the memory cost, it is $\mathcal{O}(n)$ since for each calculation by row, it only stores n results and sum to one output. Similarly, since $\boldsymbol{y}_2 \in \mathbb{R}^{m\times 1}$, the time and memory complexity are $\mathcal{O}(mn)$. Thus, by adding these together, we can get that the number of scalar multiplications and memory cost in evaluating $\boldsymbol{Z}$ with forward-mode are $\mathcal{O}(n^2 + mn)$.

For the reverse-mode calculation, we firstly compute $\boldsymbol{y}_1\boldsymbol{y}_2^T$. It requires mn scalar multiplications and mn memory cost as it outputs a $n \times m$ matrix. Let's denote this matrix as $\boldsymbol{M}$. Then, we compute $\boldsymbol{v}^T\boldsymbol{M}$, which requires mn scalar multiplications and $max(n, m)$ memory cost. It will outputs a $1 \times m$ matrix. Let $\boldsymbol{N}$ be that matrix. Finally, we need to compute $\boldsymbol{vN}$. Since the result is a $n \times m$ matrix, the number of scalar multiplications and memory cost are $\mathcal{O}(mn)$. Thus, by adding these together, we can get that the number of scalar multiplications and memory cost in evaluating $\boldsymbol{Z}$ with reverse-mode are $\mathcal{O}(mn)$.

Based on that, we can easily get if $n > m$, the reverse-mode is more efficient. If $n = m$, $\mathcal{O}(n^2+nm)$ is exactly $\mathcal{O}(n^2)$, and $\mathcal{O}(nm)$ would be $\backslash^{\in}$. In this case, we can choose either forward-mode or reverse-mode. If $n < m$, we can conclude that the forward-mode is a better choice, since it only requires one step mn, while the reverse-mode will do mn three times. In other words, we choose the forward-mode if the shape of $\boldsymbol{Z}$ is "wide".

3. Linear Regression

3.1. Deriving the Gradient

We know the loss is $\frac{1}{n}\sum_{i=1}^{n}(\hat{\boldsymbol{w}}^T\boldsymbol{x}_i - t_i)^2$. Thus, the gradient should be

$$\begin{aligned}\frac{\partial}{\partial\hat{\boldsymbol{w}}}\frac{1}{n}\sum_{i=1}^{n}(\hat{\boldsymbol{w}}^T\boldsymbol{x}_i - t_i)^2 &= \frac{\partial}{\partial\hat{\boldsymbol{w}}}\frac{1}{n}||X\hat{\boldsymbol{w}} - \boldsymbol{t}||_2^2\\ &= \frac{2}{n}(X\hat{\boldsymbol{w}} - \boldsymbol{t})^T X\end{aligned}$$

3.2. Underparameterized Model

Since $d < n$, X^TX is invertible. In order to find the minimum, we just need to find the solution when the gradient is 0. Thus, we can get

$$\begin{aligned}&\frac{2}{n}(X\hat{\boldsymbol{w}} - \boldsymbol{t})^T X = 0\\ \implies\ & X^T(X\hat{\boldsymbol{w}} - \boldsymbol{t}) = 0\\ \implies\ & X^TX\hat{\boldsymbol{w}} = X^T\boldsymbol{t}\\ \implies\ & \hat{\boldsymbol{w}} = (X^TX)^{-1}X^T\boldsymbol{t} \qquad \text{since } X^TX \text{ is invertible}\end{aligned}$$

Since $d < n$ gives that the invertibility holds, we can conclude that the solution is unique.

3.3. Overparameterized Model

3.3.1. Since $n = 1$, and $d = 2$, we can get that the dimension of $\boldsymbol{w}$ should be 1×2. To minimize $\boldsymbol{w}^T\boldsymbol{x}_1 - t_1$, we have

$$\begin{aligned}\boldsymbol{w}^T\boldsymbol{x}_1 - t_1 &= w_1x_1 + w_2x_2 - t_1\\ &= w_1 + w_2 - 3\end{aligned}$$

Thus, if w_1 and w_2 satisfy that $w_1 + w_2 = 3$, that could be the solution of the minimizer. Thus, there are infinitely many empirical risk minimizers.

3.3.2. Since it starts at zero initialization, we can get

$$\begin{aligned}\hat{\boldsymbol{w}} &\leftarrow 0 - \alpha\frac{\partial\mathcal{J}}{\partial\hat{\boldsymbol{w}}}\\ &= -\frac{2\alpha}{n}(X\hat{\boldsymbol{w}} - \boldsymbol{t})^T X\end{aligned}$$

Thus, $\boldsymbol{w}$ can be represented as the linear combination of rows of X. In order to make the calculation easier, let $v \in \mathbb{R}^n$ and set $\boldsymbol{w} = X^T v$.
Recall that if $d > n$, XX^T is invertible. Thus, we have

$$X\hat{\boldsymbol{w}} - \boldsymbol{t} = 0 \tag{1}$$
$$\implies XX^T v - \boldsymbol{t} = 0 \tag{2}$$
$$\implies XX^T v = \boldsymbol{t} \tag{3}$$
$$\implies v = (XX^T)^{-1}\boldsymbol{t} \qquad \text{since } XX^T \text{ is invertible} \tag{4}$$
$$\implies \hat{\boldsymbol{w}} = X^T(XX^T)^{-1}\boldsymbol{t} \tag{5}$$

Since $d > n$ gives that the invertibility holds, we can conclude that the solution is unique.

3.3.3. The code for this question is shown below.

```
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 = ... (Your solution for Part 3.3.2)
    W = X_expand.T.dot(np.linalg.inv(X_expand.dot(X_expand.T))).dot(t)
  else:
    ## W = ... (Your solution for Part 3.2)
    W = np.linalg.inv(X_expand.T.dot(X_expand)).dot(X_expand.T).dot(t)
  return W

```

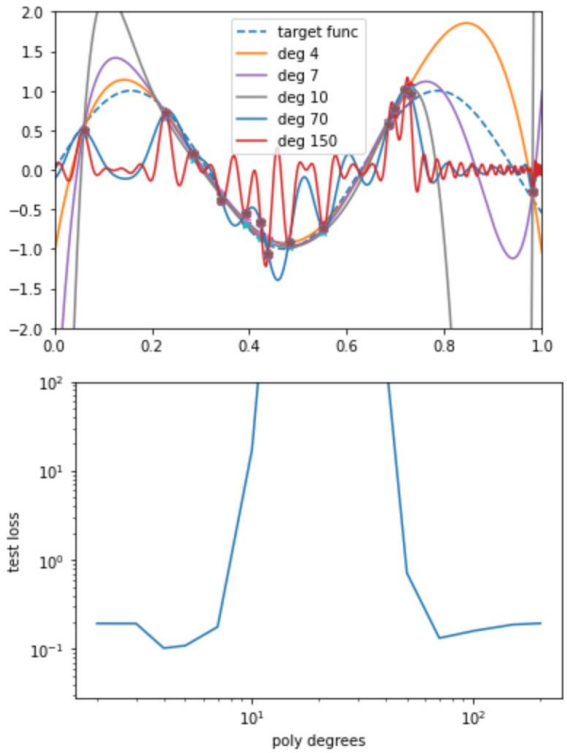


Figure 1: The result of different polynomial regressions.

According to Figure 1, we can see that overparameterization does not always lead to overfitting. Although the test error is high for models which the polynomial degree is between 10 and 100 approximately, the test error of models which the polynomial degree is over 100 are similar to the test error of low degree models, which has a small test error.