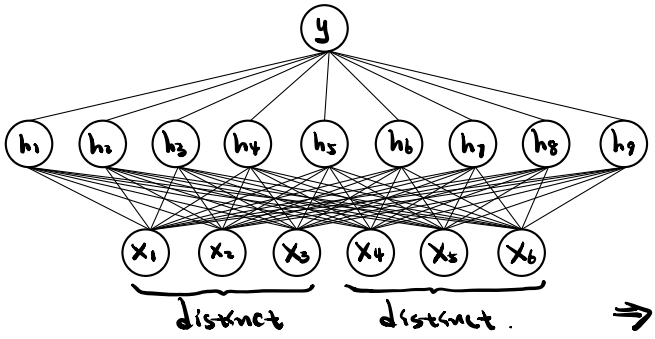
1. Hard-Coding Networks

1.1 Verify Element in List

$$W^1 = \begin{bmatrix} -2 & 0 & 0 & 2 \\ 0 & -2 & 0 & 2 \\ 0 & 0 & -2 & 2 \end{bmatrix} \quad b^1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad w^2 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad b^2 = -2$$

1.2 Verify Permutation



⇒ Just like the neural network in Section 1.1, it has 4 inputs which compares the last number to the other three numbers and output a 1 to hidden layer when there is a match.

⇒ Same idea applies here, this neural network can compare x_4, x_5 and x_6 with other 3 numbers x_1, x_2 and x_3, respectively. By applying proper weights and bias terms, and same activation function, the outputs of hidden layer could be: (output 0 if no match).

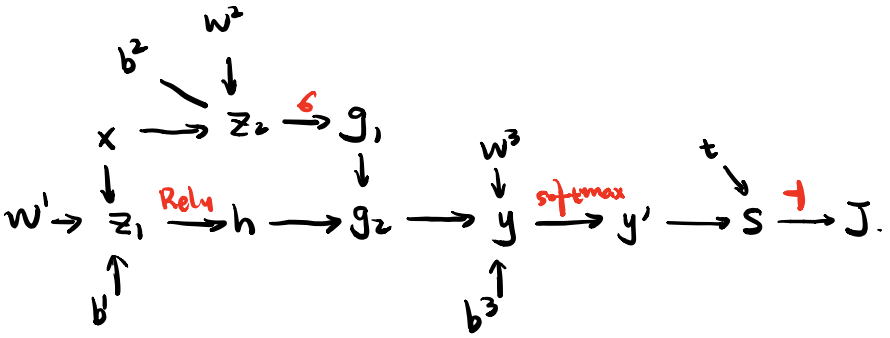
- x_4 vs. x_{1-3}: output to h_{1-3}; has a 1 if x_4 matches either one from x_{1-3}
- x_5 vs. x_{1-3}: output to h_{4-6}; has a 1 if x_5 matches either one from x_{1-3}
- x_6 vs. x_{1-3}: output to h_{7-9}; has a 1 if x_6 matches either one from x_{1-3}

If x_{4-6} are a permutation of x_{1-3}, the sum of all hidden units will be 3 and output a 1 to the output layer by applying proper parameters.

2. Backpropagation.

2.1

2.1.1 computational graph.



2.1.2 Backward Pass

$$
\begin{aligned}
\mathbf{z}_1 &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\
\mathbf{h} &= \mathrm{ReLU}(\mathbf{z}_1) \\
\mathbf{z}_2 &= \mathbf{W}^{(2)}\mathbf{x} + \mathbf{b}^{(2)} \\
\mathbf{g}_1 &= \sigma(\mathbf{z}_2) \\
\mathbf{g}_2 &= \mathbf{h} \odot \mathbf{g}_1 \\
\mathbf{y} &= \mathbf{W}^{(3)}\mathbf{g}_2 + \mathbf{b}^{(3)}, \\
\mathbf{y}' &= \mathrm{softmax}(\mathbf{y}) \\
\mathcal{S} &= \sum_{k=1}^{N} \mathbb{I}(t=k)\log(\mathbf{y}'_k) \\
\mathcal{J} &= -\mathcal{S}
\end{aligned}
$$

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

$$\bar{S} = \bar{J}\frac{\partial J}{\partial S} = -\bar{J}$$

$$\bar{y}' = \bar{S}\frac{\partial S}{\partial y'} = (-1)\sum_{k=1}^{N} \mathbb{I}(t=k)\frac{1}{y'_k} = \sum_{k=1}^{N} \mathbb{I}(t=k)\frac{-1}{y'_k}$$

$$\bar{y} = \bar{y}'\frac{\partial y'}{\partial y} = \bar{y}' \odot \mathrm{Softmax}'(y)$$

$$= \begin{cases} \bar{y}'\left[\mathrm{softmax}(y_k) \times (1 - \mathrm{softmax}(y_j))\right] & k = j \\ \bar{y}'\left[-\mathrm{softmax}(y_k) \times \mathrm{softmax}(y_j)\right] & k \neq j \end{cases}$$

$$\bar{g}_2 = \bar{y}\frac{\partial y}{\partial g_2} = W^{3T}\bar{y}$$

$$\bar{g}_1 = \bar{g}_2\frac{\partial g_2}{\partial g_1} = \bar{g}_2 \odot h$$

$$\bar{h} = \bar{g}_2\frac{\partial g_2}{\partial h} = \bar{g}_2 \odot g_1$$

$$\bar{z}_2 = \bar{g}_1\frac{\partial g_1}{\partial z_2} = \bar{g}_1 \odot \sigma'(z_2)$$

$$= \bar{g}_1 \odot [\sigma(z_2) \cdot (1 - \sigma(z_2))]$$

$$\bar{z}_1 = \bar{h}\frac{\partial h}{\partial z_1} = \bar{h} \odot \mathrm{ReLU}(z_1)$$

$$= \bar{h}_i \odot \begin{cases} 1 & \text{when } z_{1i} > 0. \\ 0 & \text{when } z_{1i} \leq 0. \end{cases} \quad \text{for } i \in \{1 \cdots M\}$$

$$\bar{x} = \bar{z}_1 \frac{\partial z_1}{\partial x} + \bar{z}_2 \frac{\partial z_2}{\partial x} = W^{1T}\bar{z}_1 + W^{2T}\bar{z}_2$$

2.2 Automatic Differentiation.

2.2.1 compute Hessian. $L(x) = x^T V V^T x$

$\Rightarrow \nabla L(x) = 2VV^T x \quad \Rightarrow H(x) = 2VV^T$

$$= 2 \times \begin{bmatrix} 4 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 4 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 32 & 16 & 24 \\ 16 & 8 & 12 \\ 24 & 12 & 18 \end{bmatrix}$$

2.2.2 computation cost

$\Rightarrow$ number of scalar multiplication : $O(n^2)$

$\Rightarrow$ memory cost : $O(n^2)$.

2.3 Vector-Hessian Products.

$\Rightarrow$ Reverse Mode: $M = V^T y = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 1 + 2 + 3 = 6$

$$z = VM = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \times 6 = \begin{bmatrix} 6 \\ 12 \\ 18 \end{bmatrix}$$

$\Rightarrow$ time and memory cost = $O(n)$

$\Rightarrow$ Forward Mode: $H = VV^T = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix}$

$$z = Hy = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 6 \\ 12 \\ 18 \end{bmatrix}$$

$\Rightarrow$ time and memory cost = $O(n^2)$

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

⇒ Reverse mode: $Z = V(V^T(y_1 y_2^T))$

$= [n\times 1]([1\times n]([n\times 1][1\times m]))$

⇒ computation cost $= 3nm$.

⇒ Forward mode: $Z = V V^T y_1 y_2^T$

$= [n\times 1][1\times n][n\times 1][1\times m]$

⇒ computation cost $= 2n^2 + nm$.

- wide ($m \gg n$): m dominates n ⇒ $2n^2+nm < 3nm$ ⇒ forward mode is better.
- tall ($n \gg m$): n dominate n ⇒ $2n^2+m \gg 3nm$ ⇒ Backward mode is better

3. Linear Regression

$$\min_{\hat{\mathbf{w}}} \frac{1}{n}\sum_{i=1}^{n}(\hat{\mathbf{w}}^\top \mathbf{x}_i - t_i)^2 = \min_{\hat{\mathbf{w}}} \frac{1}{n}\|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2.$$

3.1 Deriving the Gradient.

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

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

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

$$\Rightarrow \frac{\partial L}{\partial \hat{w}} = \frac{1}{n}(2X^T X\hat{w} - X^T t - X^T t + 0)$$

$$= \underline{\frac{2}{n}(X^T X\hat{w} - X^T t)}.$$

3.2 Underparameterized model. ($d < n$)

set $\frac{\partial L}{\partial \hat{w}} = 0 \Rightarrow \frac{2}{n}(X^T X\hat{w} - X^T t) = 0.$

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

$$X^T X\hat{w} = X^T t.$$

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

X^TX is invertible when $n > d$ ⇒ solution is unique.

0 d le d

3.3 verparameterize mo (>n)

3.3.1 Let $\hat{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$.

⇒ $\hat{w}^T x_1 = t_1$

Substitute vectors: $\begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 3 \Rightarrow w_2 = 3 - w_1$

⇒ therefore, there are infinite number of weights on this line.

3.3.2 from 3.1 ⇒ $\frac{\partial L}{\partial \hat{w}} = \frac{2}{n}(X^TX\hat{w} - X^Tt)$.

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

initialization $\hat{w} = 0 \Rightarrow \frac{\partial L}{\partial \hat{w}_0} = \frac{-2}{n} X^Tt$

first update: $\hat{w}_1 \leftarrow \hat{w}_0 - \alpha \frac{\partial L}{\partial \hat{w}_0} = 0 + \frac{2\alpha}{n} X^Tt = \frac{2\alpha}{n} X^Tt$.

⇒ $\frac{2\alpha}{n} X^Tt$ can be written as X^TC_1 for a vector of constants with dimension $C_1 \in R^{n\times 1}$.

Second update: $\frac{\partial L}{\partial w_1} = \frac{2}{n} X^T(X\hat{w}_1 - t)$

$= \frac{2}{n} X^T(X(X^TC_1) - t)$.

$\hat{w}_2 \leftarrow \hat{w}_1 - \alpha \frac{\partial L}{\partial w_1} = X^TC_1 - \frac{2\alpha}{n} X^T(X(X^TC_1) - t)$.

$= X^T(C_1 - \frac{2\alpha}{n}(X(X^TC_1) - t))$

still a vector of constants in $R^{n\times 1}$

therefore, $\hat{w}_2$ can be written as X^TC_2 for $C_2 \in R^{n\times 1}$.

Conclusion: as we can see from the two weight updates

⇒ for each $\hat{w}_i$, $\hat{w}_i = X^TC$ for $X \in R^{n\times d}$, $C \in R^{n\times 1}$.

therefore, from $\hat{w} = X^T C$ and $X\hat{w} = t$
① ②

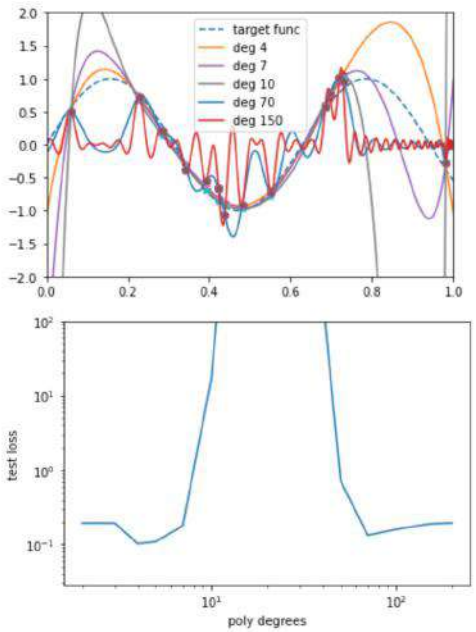
substitute ① to ②: $XX^T C = t \Rightarrow C = (XX^T)^{-1} t$

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

3.3.3

```
# to be implemented; fill in the derived solution for the underparameterized (d<n) and overparameterized (d>n) problem

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



overparameterization (high degree polynomial) does not always lead to overfitting. By observing the second graph, polynomial degree around 70 or bigger shows better generalization in test data than degree 10 - 40 (underparametrization)