Question 1 Hard-Coding Networks.

(a) Verify Sort. *Find a set of weights and biases for a multilayer perceptron which determines if a list of length 4 is in sorted order.*

$$\mathbf{W}^{(1)} = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix} \quad \mathbf{b}^{(1)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad w^{(2)} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \quad b^{(2)} = -3$$

(b) Perform Sort. *Describe how to implement a function $\hat{f} : \mathbb{R}^4 \to \mathbb{R}$ where $\hat{f}(x_1, x_2, x_3, x_4) = (\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4)$ where $(\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4)$ is (x_1, x_2, x_3, x_4) in sorted order. Implement f using a feedforward or recurrent neural network with element wise activations.*

Bubble sort approach.
1. Make a min-max net. If input is A and B, $min = A - (A - B)_+, max = A + (B - A)_+$. Min-max net can be used to swap disordered numbers.
2. Given N inputs, construct $(N-1)!$ hidden layers, which all consist of the same number of outputs as the input layer. Each layer compares two neighboring inputs and swaps them if disordered. The order of comparison is as in the bubble sort. Specifically, the first two numbers are first compared, then, the second and the third, and so on. After the first round sorting from left to right, the last output will become the maximum value, and the second to last will become the second largest value.

Brute-force approach.
In previous section, we get Verify Sort network, named $vs(\cdot)$.
1. Make permutations of the input. Assuming the number of input is N, we need $\mathbb{A}_N^N = N!$ permutations. The permutations are generated on the first hidden layer. Each node on the first hidden layer is only connected to the corresponding node on the input layer. For example, to generate a permutations $(\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \tilde{x}_4) = (x_2, x_3, x_1, x_4)$, the four nodes on the upper layer can only be connected to node $(\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \tilde{x}_4)$ respectively, i.e. only their weights are 1.
2. Verify sort. Employ Verify Sort network on all the permutations. We get $N!$ verification results, $vs(\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \tilde{x}_4)$. At the same time, we save the sum of all the verification results, $V = \sum vs(\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \tilde{x}_4)$, in case of duplicate input (if no duplicate input, $V = 1$).
3. Multiply verification results and original input, $\hat{f}(x_1, x_2, x_3, x_4) = \frac{\sum_{\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \tilde{x}_4}((\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \tilde{x}_4) \times vs(\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \tilde{x}_4))}{\sum_{\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \tilde{x}_4}((\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \tilde{x}_4)}$.

(c) Universal Approximation Theorem.

1.3.1 *Consider a bump function $g(h, a, b, x) = h \cdot \mathbb{I}(a \leq x \leq b)$ visualized in Figure 2. Given some (h, a, b) show $\exists \tau : \hat{f}_\tau(x) = g(h, a, b, x)$.*

n = 2 $\quad \mathbf{W}_0 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad \mathbf{b}_0 = \begin{bmatrix} -a \\ b \end{bmatrix} \quad \mathbf{W}_1 = \begin{bmatrix} h & h \end{bmatrix} \quad \mathbf{b}_1 = -h$

1.3.2 *Given $f(x) = -x^2 + 1$ where where $I = [-1, 1]$ and some initial function $\hat{f}_0(x) = 0$ which is identically 0, construct a new function $\hat{f}_1(x) = \hat{f}_0(x) + g(h_1, a_1, b_1, x)$ such that $||f - \hat{f}_1|| < ||f - \hat{f}_0||$. Note that h_1, a_1, and b_1 are going to depend on our choice of f, $\hat{f}_0$ and I. Plot f and $\hat{f}_1$.*

$f(x) = -x^2 + 1$.

$\hat{f}_1(x) = \hat{f}_0(x) + g(h_1, a_1, b_1, x) = g(h_1, a_1, b_1, x)$. I set $h_1 = \frac{3}{4}, a_1 = -\frac{1}{2}, b_1 = \frac{1}{2}$, $\hat{f}_1(x) = g(\frac{3}{4}, -\frac{1}{2}, \frac{1}{2}, x)$.

Question 1 [(c)] continued on next page...

Figure 1 shows the figure of f, $\hat{f}_0$ and $\hat{f}_1$.

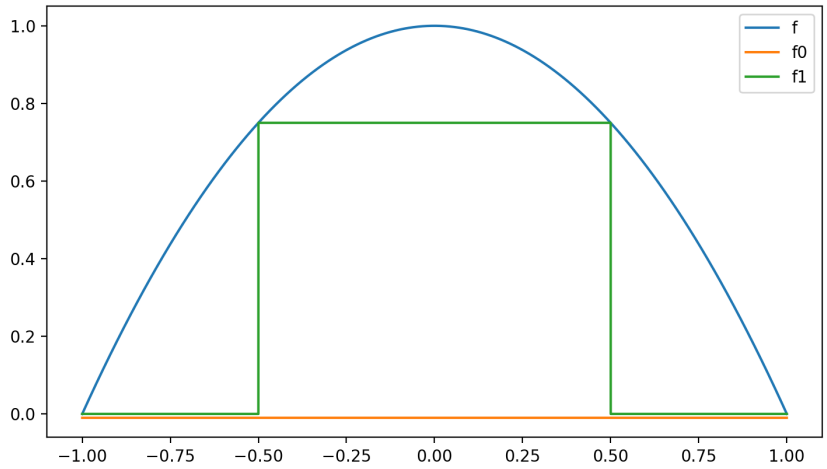

Figure 1: Figure of f, $\hat{f}_0$ and $\hat{f}_1$

1.3.3 *Describe a procedure which starts with $\hat{f}_0(x) = 0$ and a fixed N then construct a series $\{\hat{f}_i\}_{i=0}^{N}$ where $\hat{f}_{i+1}(x) = \hat{f}_i(x) + g(h_{i+1}, a_{i+1}, b_{i+1}, x)$ which satisfies $||f - \hat{f}_{i+1}|| < ||f - \hat{f}_i||$. Plot f, $\hat{f}_1$, $\hat{f}_2$, and $\hat{f}_3$.*

As we should construct N functions, I first equally divide $[-1, 0]$ into $(N+1)$ parts,

$$[-1, -1 + \frac{1}{N+1}], ..., [-1 + \frac{1}{N+1} * (k-1), -1 + \frac{1}{N+1} * k], ..., [-\frac{1}{N+1}, 0]$$

Thus,

$$a_{k+1} = -1 + \frac{k+1}{N+1}, \quad b_{k+1} = 1 - \frac{k+1}{N+1},$$

$$h_{k+1} = -(1 - \frac{k+1}{N+1})^2 + 1 - \sum_{i=1}^{k} h_i = -(1 - \frac{k+1}{N+1})^2 + 1 - \sum_{i=1}^{k} [-(1 - \frac{i}{N+1})^2 + 1]$$

$$= -(1 - \frac{k+1}{N+1})^2 + 1 - (-(1 - \frac{k}{N+1})^2 + 1) = (1 - \frac{k}{N+1})^2 - (1 - \frac{k+1}{N+1})^2$$

Suppose $N = 3$,

$$f(x) = -x^2 + 1$$

$$\hat{f}_0(x) = 0$$

$$\hat{f}_1(x) = \begin{cases} \frac{7}{16} & -\frac{3}{4} \leq x \leq \frac{3}{4} \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{f}_2(x) = \begin{cases} \frac{5}{16} & -\frac{1}{2} \leq x \leq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{f}_3(x) = \begin{cases} \frac{3}{16} & -\frac{1}{4} \leq x \leq \frac{1}{4} \\ 0 & \text{otherwise} \end{cases}$$

Question 1 [(c)] continued on next page...

Figure 2 shows the figure of $f, \hat{f}_1, \hat{f}_2$, and $\hat{f}_3$.

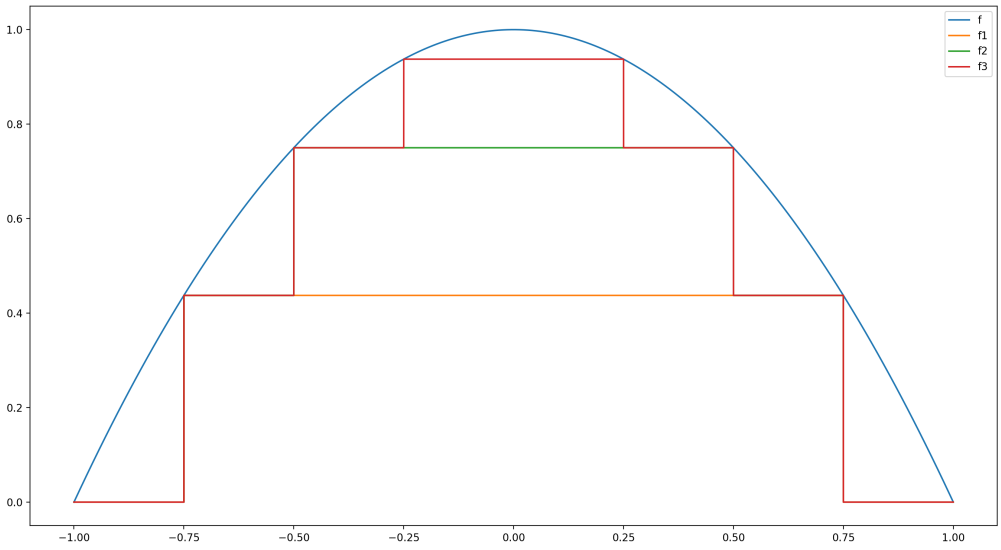

Figure 2: Figure of $f, \hat{f}_1, \hat{f}_2$, and $\hat{f}_3$

Question 2 Backprop.

(a) Computational Graph. *Consider a neural network with N input units, M output units, and K hidden units.*

2.1.1 *Draw the computation graph relating* $\boldsymbol{x}, t, \boldsymbol{z}, \boldsymbol{h}, \boldsymbol{y}, \boldsymbol{y}', \boldsymbol{r}, R, S$, *and* J.

Figure 3 shows the figure of the computation graph.

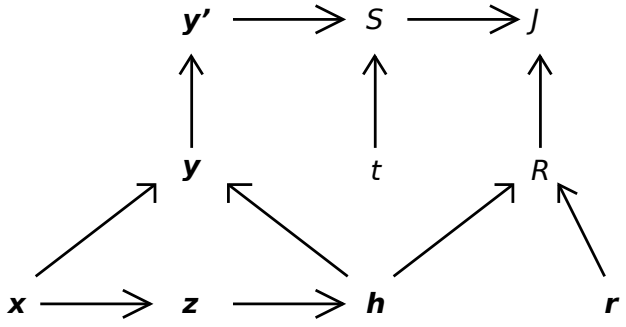

Figure 3: Figure of computation graph relating $\mathbf{x}, t, \mathbf{z}, \mathbf{h}, \mathbf{y}, \mathbf{y}', \mathbf{r}, R, S$, and J.

2.1.2 *Derive the backprop equations for computing* $\bar{x} = \frac{\partial J}{\partial x}$. *You may use softmax′ to denote the derivative of the softmax function (so you don't need to write it out explicitly).*

$$\bar{J} = 1$$
$$\bar{R} = \bar{S} = \bar{J} = 1$$
$$\bar{y'} = \bar{S} = 1$$
$$\bar{y} = \bar{y'} \cdot softmax'(y) = softmax'(y)$$
$$\bar{h} = \bar{R} \cdot r + W^{(2)T}\bar{y} = r + W^{(2)T}softmax'(y)$$
$$\bar{z} = \bar{h} \cdot ReLU'(z) = \begin{cases} \bar{h} & z > 0 \\ 0 & otherwise \end{cases}$$
$$\bar{x} = W^{(1)T}\bar{z} + \bar{y} = W^{(1)T}((r + W^{(2)T}softmax'(y)) \cdot ReLU'(z)) + softmax'(y)$$

(b) Vector-Jacobian Products (VJPs).

2.2.1 Compute $J \in \mathbb{R}^{n \times n}$ for $n = 3$ and $v^T = [1, 2, 3]$ - i.e, weite down the values in $J = \begin{pmatrix} j_{1,1} & j_{1,2} & \cdots & j_{1,n} \\ j_{2,1} & j_{2,2} & \cdots & j_{2,n} \\ \cdots & \cdots & \cdots & \cdots \\ j_{n,1} & j_{n,2} & \cdots & j_{n,n} \end{pmatrix}$.

$$J = \frac{\partial f}{\partial x} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{pmatrix}$$

2.2.2 *What is the time and memory cost of evaluating the Jacobian of function* $\boldsymbol{f}$ *in terms of n?*

$$J = \frac{\partial f}{\partial x} = vv^T$$

Thus, the time complexity of computing Jacobian is $\mathbf{O}(n^2)$. Space complexity is also $\mathbf{O}(n^2)$.

2.2.3 *Describe how to evaluate* $J^T y$ *with a time and memory cost that is linear in n. Then, compute* $z = J^T y$ *where* $v^T = (1, 2, 3)$ *and* $y^T = [1, 1, 1]$ *- i.e., write down the entries in* $z^T = [z_1, ..., z_n]$.

$$J^T y = vv^T y = v(v^T y)$$

We could first calculate $v^T y$, the time and space complexity of which is both $\mathbf{O}(n)$; then, multiply v with this scaler $v^T y$ to get final result, which has the same time and space complexity as last process.

For $v^T = (1, 2, 3)$ and $y^T = [1, 1, 1]$,

$$z = J^T y = vv^T y = v(v^T y) = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} (\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}) = 6 \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 6 \\ 12 \\ 18 \end{bmatrix}$$

Question 3 Linear Regression.

Question 3 continued on next page...

(a) Deriving the Gradient. *Write down the gradient of the loss w.r.t. the learned parameter vector $\hat{w}$.*

$$\frac{\partial L}{\partial \hat{w}} = \frac{\partial \frac{1}{n}(X\hat{w}-t)^2}{\partial \hat{w}} = \frac{2}{n}X^T(X\hat{w}-t) = \frac{2}{n}(X^TX\hat{w}-X^Tt)$$
$$or \quad \frac{\partial L}{\partial \hat{w}} = \frac{\partial \frac{1}{n}\sum_{i-1}^{n}(\hat{w}^Tx^{(i)}-t^{(i)})^2}{\partial \hat{w}} = \frac{2}{n}\sum(\hat{w}^Tx^{(i)}-t^{(i)})x$$

(b) Underparameterized Model.

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

$$X^TX\hat{w} - X^Tt = 0$$
$$\hat{w} = (X^TX)^{-1}X^Tt$$

It is unique, as the answer above is calculated by setting vectorized gradient to 0. As a concave function, there should be only one lowest point.

3.2.2 *Assume that ground truth labels are generated by a linear target: $t_i = w^{*T}x_i$. Show that the solution in part (b) achieves perfect generalization when $d < n$, i.e. $\forall x \in \mathbb{R}^d, (w^{*T}x - \hat{w}^Tx)^2 = 0$*

As $t_i = w^{*T}x_i$, $\mathbf{t} = \mathbf{X}\hat{\mathbf{w}}^*$.
In part **3.2.1**, we have

$$\begin{aligned}\hat{w} &= (X^TX)^{-1}X^Tt \\ &= (X^TX)^{-1}X^TXw^* \quad = w^*\end{aligned}$$

Thus,

$$(w^{*T}x - \hat{w}^Tx)^2 = \quad (w^{*T}x - w^{*T}x)^2 = 0$$

(c) Overparameterized Model: 2D Example.

3.3.1

$$\hat{w}^Tx_1 = t_1$$
$$\begin{bmatrix} w_1 & w_2 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \end{bmatrix} = 2$$
$$2w_1 + w_2 = 2$$

We have showed that there exists infinitely many $\hat{w}$ satisfying $\hat{w}^Tx_1 = y_1$ on a real line.

3.3.2 *what is the direction of the gradient? You should write down a unit-norm vector. Does the direction change along the trajectory? Based on this geometric intuition, which solution - along the line of solutions - does gradient descent find? Provide a pictorial sketch or a short description of your reasoning.*

$$\frac{\partial L}{\partial \hat{w}} = \frac{2}{n}(X^TX\hat{w} - X^Tt)$$
$$\frac{\partial L}{\partial \hat{w}(0)} = 2(X^TX\hat{w} - X^Tt) = \begin{bmatrix} -8 \\ -4 \end{bmatrix}$$

Question 3 [(c)] continued on next page...

Thus, the direction of the gradient is $\begin{bmatrix} -\frac{2\sqrt{5}}{5} \\ -\frac{\sqrt{5}}{5} \end{bmatrix}$.

After the first step of gradient descent, the $\hat{w}(1)$ becomes $\begin{bmatrix} 2m \\ m \end{bmatrix}$. ($m$ could be influenced by learning rate)

$$\frac{\partial L}{\partial \hat{w}(1)} = \frac{2}{n}(X^T X\hat{w} - X^T t) = 2(\begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} 2m \\ m \end{bmatrix} - \begin{bmatrix} 4 \\ 2 \end{bmatrix})$$
$$= 2(\begin{bmatrix} 10m-4 \\ 5m-2 \end{bmatrix}) = 2(\begin{bmatrix} 2(5m-2) \\ 5m-2 \end{bmatrix})$$

Thus, at the beginning of the learning process, the direction will not be changed. The direction will not change, if learning rate is low enough.

However, if learning rate is relatively high, the direction could flip from $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$ to $\begin{bmatrix} -2 \\ -1 \end{bmatrix}$.

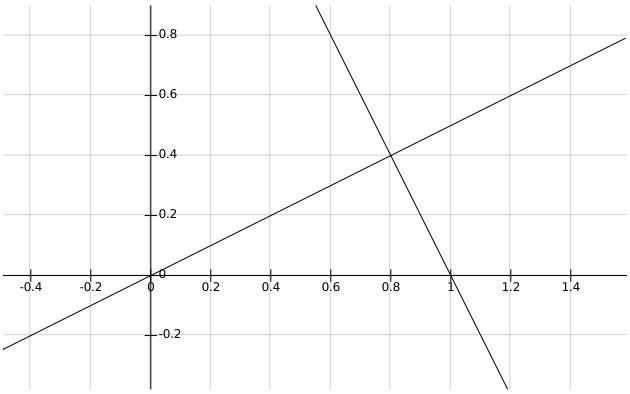

Figure 4: Function $w_2 = 0.5w_1$ and $w_2 = -2w_1 + 2$. The x-axis denotes w_1 and the y-axis denotes w_2.

Figure 4 shows the function of the gradient descent direction and the solution equation in part **3.3.1**. The crossing point is $w_1 = 0.8, w_2 = 0.4$.

3.3.3 *Give a geometric argument that among all the solutions on the line, the gradient descent solution from the previous part has the smallest Euclidean norm.*

See the Figure 4 above, the line passing $(1, 0)$ is the solution line. The Euclidean norms of all the solutions are the distance between origin and the solution point. Therefore, the intersection, $(w_1 = 0.8, w_2 = 0.4)$, of the solution line and the line, which is perpendicular to the solution line and passing the origin and the solution point, is the optimal solution. Using Pythagorean theorem, we can find that the triangle formed by these two lines and the x-axis is a right-angled triangle, since the squares of the three sides are 1, 0.8, and 0.2.

(d) Overparameterized Model: General Case.

3.4.1 *Now we generalize the previous geometric insight developed to general $d > n$. Show that gradient descent from zero initialization i.e. $\hat{w}(0) = 0$ finds a unique minimizer if it converges. Write down the solution and show your work.*

As we have

$$\begin{aligned}\frac{\partial L}{\partial \hat{w}} = \frac{2}{n}(X^T X\hat{w} - X^T t) &= 0\\ X^T X\hat{w} &= X^T t\\ XX^T X\hat{w} &= XX^T t\\ (XX^T)^{-1}XX^T X\hat{w} &= (XX^T)^{-1}XX^T t\\ X\hat{w} &= t\end{aligned}$$

Possible solutions are on hyperplane $X\hat{w} = t$. The iterative gradient descent equation is

$$\begin{aligned}&\mathbf{w}(0) \leftarrow \frac{\alpha}{n}\mathbf{X}^T\mathbf{t}\\ &\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{n}(\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{t}) = \mathbf{w} - \frac{\alpha}{n}\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{t})\end{aligned}$$

Thus, $\hat{w}$ can be written as

$$\hat{w} = X^T d$$

where $d \in \mathbb{R}^n$.
To show the gradient descent from zero initialization finds a unique minimizer:

$$\begin{aligned}X\hat{w} &= t\\ XX^T d &= t\\ d &= (XX^T)^{-1}t\end{aligned}$$

Thus,

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

3.4.2 *Given the gradient descent solution from the previous part $\hat{w}$ and another zero-loss solution $\hat{w}$, evaluate $(\hat{w} - \hat{w}_1)^T\hat{w}$ .Use this quantity to show that among all the empirical risk minimizers for $d > n$, the gradient descent solution has the smallest Euclidean norm.*

The gradient descent solution from the previous part is

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

and $\hat{w}_1$ is on hyperplane

$$X\hat{w}_1 = t \quad or \quad \hat{w}_1^T X^T = t^T$$

then, we have

$$\begin{aligned}(\hat{w} - \hat{w}_1)^T\hat{w} &= (\hat{w}^T - \hat{w}_1^T)\hat{w}\\ &= \hat{w}^T\hat{w} - \hat{w}_1^T\hat{w}\\ &= t^T(XX^T)^{-1}XX^T(XX^T)^{-1}t - \hat{w}_1^T X^T(XX^T)^{-1}t\\ &= t^T(XX^T)^{-1}t - t^T(XX^T)^{-1}t\\ &= 0\end{aligned}$$

The product between $\hat{w}$ and the difference between $\hat{w}$ and another zero-loss solution $\hat{w}$ is always 0, which means these two vectors are perpendicular. The norm of other solutions is larger than $\hat{w}$; thus, among all the empirical risk minimizers for $d > n$, the gradient descent solution has the smallest Euclidean norm.

Question 3 continued on next page...

(e) Benefit of Overparameterization. *Visualize and compare underparameterized with overparameterized polynomial regression.*

```
def fit_poly(X, d,):
    X_expand = poly_expand(X, d=d, poly_type = poly_type)
    if n > d:
        ## W = ... (Your solution for Part 3.2.1)
        W = np.linalg.inv(X_expand.T.dot(X_expand)).dot(X_expand.T).dot(t)
    else:
        ## W = ... (Your solution for Part 3.4.1)
        W = X_expand.T.dot(np.linalg.inv(X_expand.dot(X_expand.T))).dot(t)
  return W
```