DSC 190

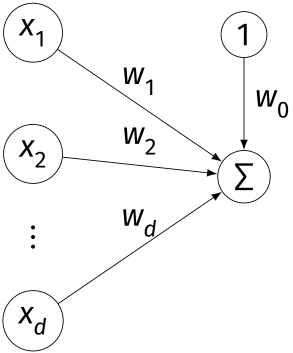
Machine Learning: Representations

Lecture 12 | Part 1

Neural Networks

Recall: Linear Predictor

- **Input**: features $\vec{x} = (x_1, \ldots, x_d)^T$
- **Parameters**:
  $\vec{w} = (w_0, w_1, \ldots, w_d)^T$
- **Output**: $w_0 + w_1 x_1 + \ldots + w_d x_d$

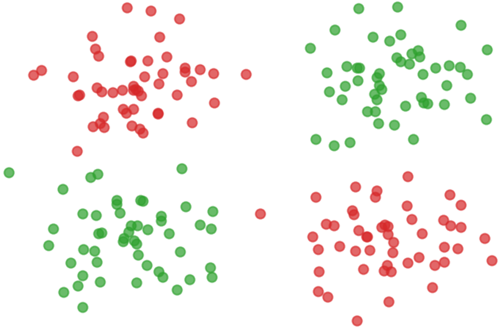


Linear Predictors

- **Pro**: simple, usually easy to optimize $\vec{w}$
  - With square loss, solution given by normal equations

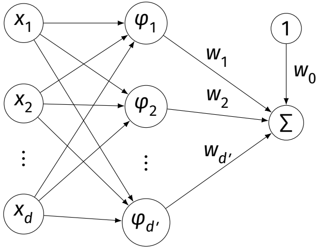
- **Con**: Decision boundary is linear

Example



Recall: Basis Functions

- **Input**: features $\vec{x}$, basis functions $\varphi_1, \ldots, \varphi_d : \mathbb{R}^d \to \mathbb{R}$
- **Parameters**: $\vec{w} = (w_0, w_1, \ldots, w_d)^T$
- **Output**: $w_0 + w_1 \varphi_1(\vec{x}) + \ldots + w_d \varphi_d(\vec{x})$



Basis Functions

- **Note**: the basis functions and the weights $\vec{w}$ are **not** chosen at the same time
- Two step process
- First, basis functions are chosen and fixed
  - By hand, by k-means clustering, etc.
- *Then* the weights $\vec{w}$ are learned

Exercise

Why do this in two steps as opposed to one?

Answer

- By fixing basis functions *then* finding best $\vec{w}$, optimization is easy again

- Using square loss, normal equations still work

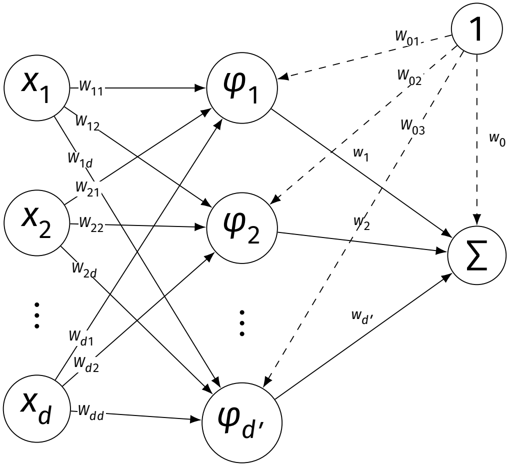
Idea

- Try to learn basis functions at same time as weights, $\vec{w}$

- Attempt #1: linear basis functions?

$$\varphi_i(\vec{x}) = W_{1i} x_1 + \ldots + W_{di} x_d$$

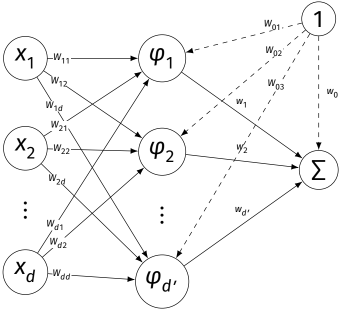
The Model



$$\varphi_i(\vec{x}) = W_{1i}x_1 + \ldots + W_{di}x_d$$

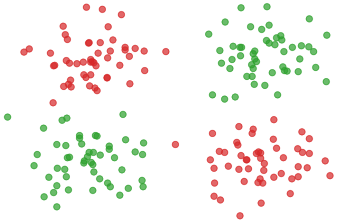
Neural Network

- **Input**: features $\vec{x}$,
- **Parameters**:
  $\vec{w} = (w_0, w_1, \ldots, w_d)^T$,
  $(d+1) \times d'$ matrix W
- **Output**:
  $w_0 + w_1\varphi_1(\vec{x}) + \ldots + w_d\varphi_d(\vec{x})$
- This is a **neural network**



Problem

- If φ_i is linear, so is the decision boundary!

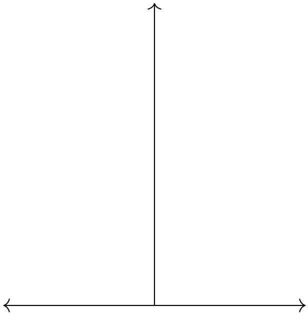


Activation Function

- To make φ_i nonlinear, we often apply a **activation function**.

- Very commonly: **rectified linear unit** (ReLU)

$$g(z) = \max\{0, z\}$$



$$\begin{aligned}\varphi_i(\vec{x}) &= g(W_{0i} + W_{1i}x_1 + W_{2i}x_2 + \ldots + W_{di}x_d A) \\ &= \max\{0, W_{0i} + W_{1i}x_1 + W_{2i}x_2 + \ldots + W_{di}x_d A\}\end{aligned}$$

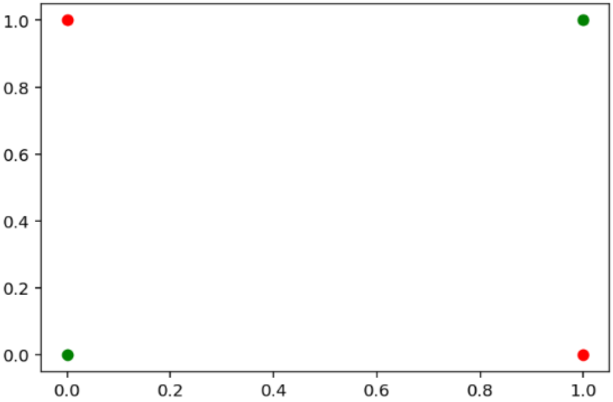
Neural Networks as Functions

- A neural network is simply a special kind of **function**.
- $f(\vec{x}; \vec{w}, W)$

Example

$$W = \begin{pmatrix} 2 & -1 \\ 3 & -2 \\ -2 & 1 \end{pmatrix} \qquad \vec{w} = \begin{pmatrix} 4 \\ 0 \\ 2 \end{pmatrix} \qquad \vec{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

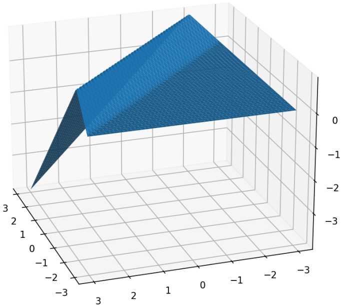
The Xor Problem



A Solution

$$W = \begin{pmatrix} 0 & -1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \qquad \vec{w} = \begin{pmatrix} 0 \\ 1 \\ -2 \end{pmatrix}$$

Prediction Surface



Learning with NNs

- We can **learn** weights by gathering data, picking a loss function and minimizing loss.
- The square loss works:

$$R(\vec{w}, W) = \frac{1}{n} \sum_{i=1}^{n} (f(\vec{x}^{(i)}; \vec{w}, W) - y_i)^2$$

Problem

- Now that the basis function weights are learnable, too, there is no simple solution for the best weights.
- We must instead use **gradient descent**.

DSC 190

Machine Learning: Representations

Lecture 12 | Part 2

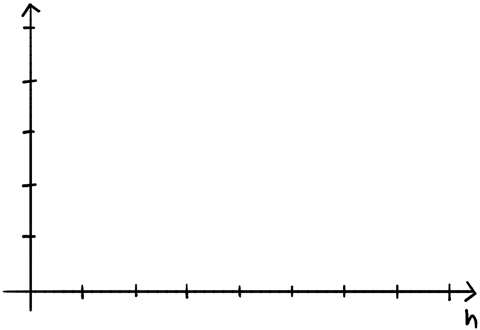
Gradient Descent

Gradient Descent

- We have a function $f : \mathbb{R} \to \mathbb{R}$
- We can't solve for the x that minimizes (or maximizes) $f(x)$
- Instead, we use the derivative to "walk" towards the optimizer

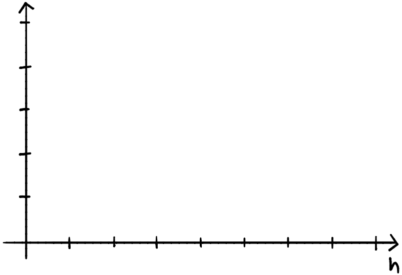
Meaning of the Derivative

- We have the derivative; can we use it?
- $\dfrac{df}{dx}(x)$ is a function; it gives the **slope** at x.



Key Idea Behind Gradient Descent

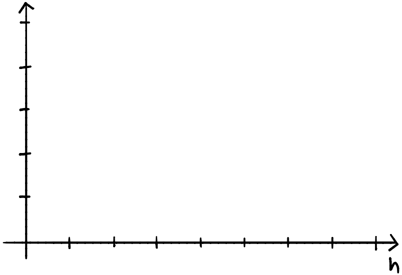
- If the slope of f at x is **positive** then moving to the **left** decreases the value of f.
- i.e., we should **decrease** x



Key Idea Behind Gradient Descent

- If the slope of f at x is **negative** then moving to the **right** decreases the value of f.

- i.e., we should **increase** x



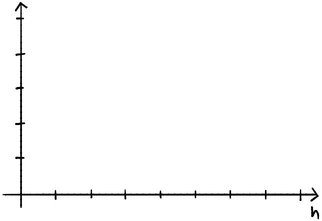
Key Idea Behind Gradient Descent

- Pick a starting place, x_0. Where do we go next?
- Slope at x_0 negative? Then increase x_0.
- Slope at x_0 positive? Then decrease x_0.
- This will work:

$$x_1 = x_0 - \frac{df}{dx}(x_0)$$

Gradient Descent

- Pick α to be a positive number. It is the **learning rate**.
- Pick a starting prediction, x_0.
- On step i, perform update $x_i = x_{i-1} - \alpha \cdot \frac{df}{dx}(x_{i-1})$
- Repeat until convergence (when x doesn't change much).



```python
def gradient_descent(derivative, x, alpha, tol=1e-12):
    """Minimize using gradient descent."""
    while True:
        x_next = x - alpha * derivative(x)
        if abs(x_next - x) < tol:
            break
        x = x_next
    return h
```

Example: Minimizing Mean Squared Error

- Recall the mean squared error and its derivative:

$$R_{\text{sq}}(x) = \frac{1}{n}\sum_{i=1}^{n}(x - y_i)^2 \qquad \frac{dR_{\text{sq}}}{dx}(x) = \frac{2}{n}\sum_{i=1}^{n}(x - y_i)$$

Exercise

Let $y_1 = -4, \quad y_2 = -2, \quad y_3 = 2, \quad y_4 = 4.$

Pick $x_0 = 4$ and $\alpha = 1/4$. What is x_1?

a) -1
b) 0
c) 1
d) 2

Example

Gradient Descent in > 1 dimensions

- The derivative of f becomes the gradient:

$$\frac{df}{dx} \to \nabla f(\vec{x})$$

- Meaning of **differentiable**: locally, f looks linear.
- **Key**: $\nabla f(\vec{w})$ is a function; it returns a vector pointing in direction of steepest ascent.

Gradient Descent in > 1 dimensions

- Pick α to be a positive number.
  - It is the **learning rate**.
- Pick a starting guess, $\vec{w}^{(0)}$.
- On step i, update $\vec{w}^{(i)} = \vec{w}^{(i-1)} - \alpha \cdot \nabla f(\vec{w}^{(i-1)})$
- Repeat until convergence
  - when $\vec{w}$ doesn't change much
  - equivalently, when $\|\nabla f(\vec{w}^{(i)})\|$ is small

```python
def gradient_descent(gradient, w, alpha, tol=1e-12):
    """Minimize using gradient descent."""
    while True:
        w_next = w - alpha * gradient(x)
        if np.linalg.norm(w_next - w) < tol:
            break
        w = w_next
    return w
```