Combining Perceptrons
\[ g(\bar{x}) = OR \left( AND \left( h_1(\bar{x}), \overline{h_2(\bar{x})} \right), \overline{AND \left( h_1(\bar{x}), h_2(\bar{x}) \right)} \right) \]
• Boolean variables are either +1 ("true") or −1 ("false")

• Basic Boolean operations
  • Negation: $\bar{z} = -1 \times z$
  
  • And: $\text{AND}(z_1, z_2) = \text{sign}(z_1 + z_2 - 1.5)$
  
  • Or: $\text{OR}(z_1, z_2) = \text{sign}(z_1 + z_2 + 1.5)$
Building a Network

\[ g(\vec{x}) = OR \left( AND \left( h_1(\vec{x}), \overline{h_2}(\vec{x}) \right), AND \left( \overline{h_1}(\vec{x}), h_2(\vec{x}) \right) \right) \]

\[ g(\vec{x}) = \text{sign} \left( \text{sign} \left( \text{sign} \left( w_1^T \vec{x} \right) - \text{sign} \left( \overline{w_2}^T \vec{x} \right) - 1.5 \right) + \text{sign} \left( -\text{sign} \left( w_1^T \vec{x} \right) + \text{sign} \left( \overline{w_2}^T \vec{x} \right) - 1.5 \right) + 1.5 \right) \]
Multi-Layer Perceptron (MLP)
What happens if $\theta(\cdot)$ is a linear function e.g. $\theta(x) = Cx$ for some constant $C$?
Feed-Forward Neural Network (NN)

Input layer: $l = 0$

Hidden layers: $0 < l < L$

Output layer: $l = L$

Layer $l$ has dimension $d^{(l)}$ → Layer $l$ has $d^{(l)} + 1$ nodes, counting the bias node.
The architecture of a NN is the vector $\vec{d} = [d^{(0)}, d^{(1)}, \ldots, d^{(L)}]$

Architecture

Every architecture corresponds to a hypothesis set: $\mathcal{H}_{NN}(\vec{d})$

A hypothesis $h \in \mathcal{H}_{NN}(\vec{d})$ is specified by setting all the weights between nodes
The weights between layer $l - 1$ and layer $l$ are a matrix: $W^{(l)} \in \mathbb{R}^{(d^{(l-1)}+1) \times d^{(l)}}$

$w_{ij}^{(l)}$ is the weight between node $i$ in layer $l - 1$ and node $j$ in layer $l$
Every node has an incoming signal and outgoing output

\[ \bar{x}^{(l)} = \begin{bmatrix} 1 \\ \omega_s^{(l)} \end{bmatrix} \text{ and } \bar{s}^{(l)} = W^{(l)T} \bar{x}^{(l-1)} \]
Forward Propagation

- Input: weights $W^{(1)}, \ldots, W^{(L)}$ and a query point $\vec{x}$
- Initialize $x^{(0)} = \begin{bmatrix} 1 \\ \vec{x} \end{bmatrix}$
- For $l = 1, \ldots, L$
  - $s^{(l)} = W^{(l)T}x^{(l-1)}$
  - $x^{(l)} = \begin{bmatrix} 1 \\ \theta(s^{(l)}) \end{bmatrix}$
- Output: $x^{(1)}, \ldots, x^{(L)}$
• Theorem: any function that can be decomposed into perceptrons can be modelled exactly using a 3-layer MLP
• Any smooth decision boundary can be approximated to an arbitrary precision using a finite number of perceptrons
Theorem: any function that can be decomposed into perceptrons can be modelled exactly using a 3-layer MLP.

Any smooth decision boundary can be approximated to an arbitrary precision using a finite number of perceptrons.

Theorem: Any smooth decision boundary can be approximated to an arbitrary precision using a 3-layer MLP.
Theorem: Any smooth decision boundary can be approximated to an arbitrary precision using a 2-layer feed-forward NN if the activation function, $\theta$, is continuous, bounded and non-constant.
\[ \theta(\cdot) \]

- Hyperbolic tangent:

\[
\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}
\]

- \[
\frac{\partial \tanh(z)}{\partial z} = 1 - \tanh(z)^2
\]
Error of a Neural Network

\[ E_{in}(W^{(1)}, ..., W^{(L)}) = \frac{1}{n} \sum_{i=1}^{n} (h(x_i \mid W^{(1)}, ..., W^{(L)}) - y_i)^2 \]
Recall: Gradient Descent

- Iterative method for minimizing functions
- Requires the gradient to exist everywhere
Gradient Descent for Neural Networks

- **Input:** \( \mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}, \eta_0 \)

- Initialize all weights \( W^{(1)}_{(0)}, \ldots, W^{(L)}_{(0)} \) to small, random numbers and set \( t = 0 \)

- While some termination condition is not satisfied
  - For \( l = 1, \ldots, L \)
    - Compute \( G^{(l)} = \nabla_{W^{(l)}} E_{in} \left( W^{(1)}_{(t)}, \ldots, W^{(L)}_{(t)} \right) \)
    - Update \( W^{(l)}: W^{(l)}_{(t+1)} = W^{(l)}_{(t)} - \eta_0 G^{(l)} \)
    - Increment \( t: t = t + 1 \)

- **Output:** \( W^{(1)}_{(t)}, \ldots, W^{(L)}_{(t)} \)
\[
E_{in}(W^{(1)}_t, \ldots, W^{(L)}_t) = \frac{1}{n} \sum_{i=1}^{n} e \left( h(x_i | W^{(1)}_t, \ldots, W^{(L)}_t), y_i \right)
\]

\[\nabla_{W^{(l)}} E_{in}(W^{(1)}_t, \ldots, W^{(L)}_t) = \begin{bmatrix}
\frac{\partial E_{in}}{\partial w^{(l)}_{01}} & \frac{\partial E_{in}}{\partial w^{(l)}_{02}} & \cdots & \frac{\partial E_{in}}{\partial w^{(l)}_{0d^{(l)}}} \\
\frac{\partial E_{in}}{\partial w^{(l)}_{11}} & \frac{\partial E_{in}}{\partial w^{(l)}_{12}} & \cdots & \frac{\partial E_{in}}{\partial w^{(l)}_{1d^{(l)}}} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial E_{in}}{\partial w^{(l)}_{d^{(l-1)}1}} & \frac{\partial E_{in}}{\partial w^{(l)}_{d^{(l-1)}2}} & \cdots & \frac{\partial E_{in}}{\partial w^{(l)}_{d^{(l-1)}d^{(l)}}}
\end{bmatrix}
\]
Computing Gradients

\[ E_{in} \left( W^{(1)}_{(t)}, ..., W^{(L)}_{(t)} \right) = \frac{1}{n} \sum_{i=1}^{n} e \left( h \left( \overrightarrow{x}_i \mid W^{(1)}_{(t)}, ..., W^{(L)}_{(t)} \right), y_i \right) \]

\[ \frac{\partial E_{in}}{\partial w^{(l)}_{ab}} \left( W^{(1)}_{(t)}, ..., W^{(L)}_{(t)} \right) = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial e \left( h \left( \overrightarrow{x}_i \mid W^{(1)}_{(t)}, ..., W^{(L)}_{(t)} \right), y_i \right)}{\partial w^{(l)}_{ab}} \]

\[ = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial e \left( \overrightarrow{x}^{(L)}, y_i \right)}{\partial w^{(l)}_{ab}} \]
Computing $\nabla_{W^{(l)}} E_{in} \left( W^{(1)}_{(t)}, ..., W^{(L)}_{(t)} \right)$ reduces to computing

$$\frac{\partial e \left( x^{(L)}, y_i \right)}{\partial w^{(l)}_{ab}}$$

Insight: $w^{(l)}_{ab}$ only affects $e \left( x^{(L)}, y_i \right)$ via $s^{(l)}_b$

Use the chain rule:

$$\frac{\partial e \left( x^{(L)}, y_i \right)}{\partial w^{(l)}_{ab}} = \frac{\partial e \left( x^{(L)}, y_i \right)}{\partial s^{(l)}_b} \left( \frac{\partial s^{(l)}_b}{\partial w^{(l)}_{ab}} \right)$$
To compute \( \frac{\partial e(x^{(L)}, y_i)}{\partial S_b^{(l)}} \left( \frac{\partial S_b^{(l)}}{\partial w_{ab}^{(l)}} \right) \), recall:

\[
S_b^{(l)} = \sum_{a=0}^{d^{(l-1)}} w_{ab}^{(l)} x_a^{(l-1)} \quad \rightarrow \quad \frac{\partial S_b^{(l)}}{\partial w_{ab}^{(l)}} = x_a^{(l-1)}
\]

Can compute all outputs \( \vec{x}^{(l)} \ \forall \ l \in \{0, \ldots, L\} \) using forward propagation

Let \( \delta_b^{(l)} = \frac{\partial e(x^{(L)}, y_i)}{\partial S_b^{(l)}} \)

\( \overrightarrow{\delta^{(l)}} \) is called the sensitivity vector for layer \( l \)
Computing Gradients

Insight: $s_b^{(l)}$ only affects $e \left( x^{(l)}, y_i \right)$ via $x_b^{(l)}$

Use the chain rule:
$$
\delta_b^{(l)} = \frac{\partial e \left( x^{(l)}, y_i \right)}{\partial x_b^{(l)}} \left( \frac{\partial x_b^{(l)}}{\partial s_b^{(l)}} \right)
$$

Recall:
$$
x_b^{(l)} = \theta \left( s_b^{(l)} \right) \Rightarrow \frac{\partial x_b^{(l)}}{\partial s_b^{(l)}} = \frac{\partial \theta \left( s_b^{(l)} \right)}{\partial s_b^{(l)}}
$$

$$
= 1 - \left( x_b^{(l)} \right)^2
$$

when $\theta(\cdot) = \tanh(\cdot)$
Computing Gradients

Insight: $x_b^{(l)}$ affects $e \left( x^{(L)}, y_i \right)$ via $s_1^{(l+1)}, \ldots, s_d^{(l+1)}$

Use the chain rule:

$$\frac{\partial e \left( x^{(L)}, y_i \right)}{\partial x_b^{(l)}} = \sum_{c=1}^{d^{(l+1)}} \frac{\partial e \left( x^{(L)}, y_i \right)}{\partial s_c^{(l+1)}} \left( \frac{\partial s_c^{(l+1)}}{\partial x_b^{(l)}} \right)$$

$$= \sum_{c=1}^{d^{(l+1)}} \delta_c^{(l+1)} \left( w_{bc}^{(l+1)} \right)$$
\[ \delta_b^{(l)} = \frac{\partial e(x^{(L)}, y_i)}{\partial x_b^{(l)}} \left( \frac{\partial x_b^{(l)}}{\partial s_b^{(l)}} \right) \]

\[ = \left( \sum_{c=1}^{d^{(l+1)}} \delta_c^{(l+1)} (w_{bc}^{(l+1)}) \right) \left( 1 - (x_b^{(l)})^2 \right) \]

\[ \overline{\delta}^{(l)} = W^{(l+1)} \delta^{(l+1)} \otimes \left( 1 - x^{(l)} \otimes x^{(l)} \right) \]

where \( \otimes \) is the element-wise product operation

\[ \frac{\partial e(x^{(L)}, y_i)}{\partial w_{ab}^{(l)}} = \delta_b^{(l)} \left( \frac{\partial s_b^{(l)}}{\partial w_{ab}^{(l)}} \right) = \delta_b^{(l)} \left( x_a^{(l-1)} \right) \]

\[ \nabla_{W^{(l)}} e(x^{(L)}, y_i) = x^{(l-1)} (\delta^{(l)})^T \]
Can recursively compute $\overrightarrow{\delta^{(L)}}$ using $\overrightarrow{\delta^{(L+1)}}$; need to compute the base case: $\overrightarrow{\delta^{(L)}}$

Assume the output layer is a single node and the error function is the squared error:

$$
\overrightarrow{\delta^{(L)}} = \delta^{(L)}_1, \ x^{(L)} = x^{(L)}_1 \ \text{and} \ e\left(x^{(L)}_1, y_i\right) = \left(x^{(L)}_1 - y_i\right)^2
$$

$$
\delta^{(L)}_1 = \frac{\partial e \left(x^{(L)}_1, y_i\right)}{\partial s^{(L)}_1} = \frac{\delta}{\partial s^{(L)}_1} \left(x^{(L)}_1 - y_i\right)^2
$$

$$
= 2 \left(x^{(L)}_1 - y_i\right) \frac{\delta x^{(L)}_1}{\partial s^{(L)}_1} = 2 \left(x^{(L)}_1 - y_i\right)^2 \left(1 - \left(x^{(L)}_1\right)^2\right)
$$

when $\theta(\cdot) = \tanh(\cdot)$
• Input: weights $W^{(1)}, \ldots, W^{(L)}$ and a query point $\vec{x}$

• Run forward propagation to get $\vec{x}^{(1)}, \ldots, \vec{x}^{(L)}$

• Initialize $\delta^{(L)}_1 = 2 \left( x^{(L)}_1 - y_i \right) \left( 1 - \left( x^{(L)}_1 \right)^2 \right)$

• For $l = L - 1, \ldots, 1$
  • Compute $\vec{\delta}^{(l)} = W^{(l+1)} \vec{\delta}^{(l+1)} \otimes \left( 1 - \vec{x}^{(l)} \otimes \vec{x}^{(l)} \right)$

• Output: $\vec{\delta}^{(1)}, \ldots, \vec{\delta}^{(L)}$
• Input: $W^{(1)}, \ldots, W^{(L)}$ and $\mathcal{D} = \{(\overrightarrow{x_1}, y_1), \ldots, (\overrightarrow{x_n}, y_n)\}$
• Initialize $E_{in} = 0$ and $G^{(l)} = 0 \times W^{(l)}$ for $l = 1, \ldots, L$
• For $i = 1, \ldots, n$
  • Run forward propagation to get $\overrightarrow{x^{(1)}}, \ldots, \overrightarrow{x^{(L)}}$
  • Run backpropagation to get $\overrightarrow{\delta^{(1)}}, \ldots, \overrightarrow{\delta^{(L)}}$
  • Increment $E_{in}$: $E_{in} = E_{in} + \frac{1}{n}(\overrightarrow{x^{(L)}} - y_i)^2$
• For $l = 1, \ldots, L$
  • Compute $G^{(l)}_i = \overrightarrow{x^{(l-1)}} \left(\overrightarrow{\delta^{(l)}}\right)^T$
  • Increment $G^{(l)}$: $G^{(l)} = G^{(l)} + \frac{1}{n} G^{(l)}_i$
• Output: $G^{(1)}, \ldots, G^{(L)}$, the gradients of $E_{in}$ w.r.t $W^{(1)}, \ldots, W^{(L)}$
- Both forward and backpropagation contain matrix multiplications involving $W^{(1)}, ..., W^{(L)} \rightarrow$ both take time $O(|W^{(1)}| + \cdots + |W^{(L)}|)$ ...
- Computing $G^{(1)}, ..., G^{(L)}$ requires running forward and backpropagation for each training point $(\tilde{x}, y) \in D$ ...
- Each iteration of gradient descent for a neural network takes time $O \left( n(|W^{(1)}| + \cdots + |W^{(L)}|) \right)$
- Use stochastic gradient descent instead!
- Also use parallelization and GPUs / TPUs!
Stochastic Gradient Descent for Neural Networks

- **Input:** $\mathcal{D} = \{ (\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n) \}, \eta_0$

- Initialize all weights $W^{(1)}(0), ..., W^{(L)}(0)$ to small, random numbers and set $t = 0$

- While some termination condition is not satisfied
  - For $l = 1, ..., L$
    - Randomly select a point $(\mathbf{x}_1, y_1) \in \mathcal{D}$
    - Compute $G^{(l)'} = \nabla_{W^{(l)}} e \left( h(\mathbf{x}_i | W^{(1)}(t), ..., W^{(L)}(t)), y_i \right)$
    - Update $W^{(l)}: W^{(l)}(t+1) = W^{(l)}(t) - \eta_0 G^{(l)'}$
  - Increment $t: t = t + 1$

- **Output:** $W^{(1)}(t), ..., W^{(L)}(t)$
Initialization and Termination

- Initialization:
  - Randomness is good for non-convex optimization
  - Initialize weights by sampling from $N(0, \sigma^2)$

- Termination:
  - For complicated surfaces, the gradient’s magnitude is not a good metric for proximity to a minimum
  - A simple solution: combine multiple termination criteria e.g. stop if enough iterations have passed and the improvement in error is small