Today's Schedule

- Perceptron (recap)
- Problems with Perceptron
- Extensions of Perceptron
- Training of a single-layer neural network

Perceptron (recap)

- Input-to-output function
  \[ a(x) = w^T x + w_0 = w^T x \]
  where \( w = (w_0, w^T)^T \), \( x = (1, x^T)^T \)
  \[ y(x) = g(a(x)) = g(w^T x) \]
  where \( g(a) = \begin{cases} 
  1, & \text{if } a \geq 0, \\
  0, & \text{if } a < 0
  \end{cases} \)
  \( g(a) \): activation/transfer function

Perceptron structures and decision boundaries

Limitations of Perceptron

- Single-layer perceptron is just a linear classifier
  (Marvin Minsky and Seymour Papert, 1969)
- Multi-layer perceptron can form complex decision boundaries (piecewise-linear), but it is hard to train
- Training does not stop if data are linearly non-separable
- Weights \( w \) are adjusted for misclassified data only (correctly classified data are not considered at all)

A limitation of Perceptron

\[ y = g(w^T x) \]
\[ z_1 = g(w_1^{(1)} x) + w_2^{(1)} x + w_0^{(1)} \]
\[ z_2 = g(w_1^{(2)} x) + w_2^{(2)} x + w_0^{(2)} \]
\[ y = g(w_1^{(2)} z_1) + w_2^{(2)} z_2 + w_0^{(2)} \]
**Single Layer Neural Network (continued)**

**Single Layer Neural Network**

Assume a single-layer neural network with a single output node with a logistic sigmoid function:

\[
y(x) = g(w^T x) = g\left(\sum w_i x_i\right) = \frac{1}{1 + e^{-a}}
\]

\[g(a) = \frac{1}{1 + e^{-a}}\]

**Training set**: \(D = \{(x_1, t_1), \ldots, (x_N, t_N)\}\)

where \(t_i \in \{0, 1\}\)

- **Error function**: \(E(w) = \frac{1}{2} \sum_{i=1}^{N} (y_i - t_i)^2\)

- **Definition of the training problem as an optimisation problem**

\[
\min_{w} E(w)
\]

**Optimisation problem**: \(\min_{w} E(w)\)

- **No analytic solution**
- **Employ an iterative method (requires initial values)** e.g., **Gradient descent** (steepest descent), Newton’s method, Conjugate gradient methods
- **Gradient descent**
  - (scalar rep.)
  \[w^{(new)}_i = w_i - \eta \frac{\partial E(w)}{\partial w_i}, \quad (\eta > 0)\]
  - (vector rep.)
  \[w^{(new)} = w - \eta \nabla E(w), \quad (\eta > 0)\]

**Local minimum problem with the gradient descent**

**Training of the single-layer neural network**

- **Replace \(g()\) with a differentiable non-linear function** e.g., **Logistic sigmoid function**

\[g(a) = \frac{1}{1 + e^{-a}}\]

- **Conjugate gradient methods**

**Gradient descent**

\[w^{(new)}_i = w_i - \eta \frac{\partial E(w)}{\partial w_i}, \quad (\eta > 0)\]

\[E(w) = \frac{1}{2} \sum_{i=1}^{N} (y_i - t_i)^2 = \frac{1}{2} \sum_{i=1}^{N} \left( g\left( \sum w_i x_i \right) - t_i \right)^2\]

where \(y_i = g(a_i), \quad a_i = \sum w_i x_i, \quad \frac{\partial a_i}{\partial w_i} = x_i\)

\[
\frac{\partial E(w)}{\partial w_i} = \sum_{n=1}^{N} (y_n - t_n) \frac{\partial g(a_n)}{\partial a_n} \frac{\partial a_n}{\partial w_i} = \sum_{n=1}^{N} (y_n - t_n) g'(a_n) x_i = \sum_{n=1}^{N} (y_n - t_n) g(a_n) (1 - g(a_n)) x_i
\]

**How can we resolve the problem of training?**

- **Use the least squares error criterion for training**

\[E_i(w) = \sum_{i=0}^{N} (y_i - t_i)^2\]

- **Replace \(g()\) with a differentiable function**

What about removing \(g()\) in the hidden layer?

\[z_i = g(w_i^{(1)} T x) \Rightarrow z_i = w_i^{(1)} T x\]

**Choices of decision boundaries**

- \((a)\)
- \((b)\)
- \((c)\)

**Question:** Show networks with linear hidden nodes reduce to single-layer networks

**Training of the single-layer neural network**

- **Gradient descent**

\[w^{(new)} = w - \eta \nabla E(w), \quad (\eta > 0)\]
### Another training criterion – cross-entropy error

- Training problem with the mean squared error (MSE) criterion with the sigmoid function
  \[
  E_{\text{MSE}}(w) = \frac{1}{2} \sum_{n=1}^{N} (y_n - t_n)^2, \quad y_n = g(a_n)
  \]
  \[
  \frac{\partial E_{\text{MSE}}(w)}{\partial w_i} = \frac{1}{N} \sum_{n=1}^{N} (y_n - t_n) x_{ni},
  \]
  For such \( a \) that \( g(a) \approx 0 \) or \( 1 \), \( g'(a) \approx 0 \).

- **Cross-entropy error** (NE)
  \[
  E_{\text{H}}(w) = -\frac{1}{N} \sum_{n=1}^{N} \{ t_n \ln y_n + (1-t_n) \ln (1-y_n) \}
  \]
  It can be shown that:
  \[
  \frac{\partial E_{\text{H}}(w)}{\partial w_i} = \frac{1}{N} \sum_{n=1}^{N} (y_n - t_n) x_{ni}
  \]

### Other activation functions

- **Tanh**
  \[
  g(a) = \tanh(a) = \frac{1 - e^{-2a}}{1 + e^{-2a}}
  \]
  Mapping \((-\infty, +\infty) \rightarrow (-1, 1)\)
  0 (zero) centred → faster convergence than sigmoid

- **ReLU (Rectified Linear Unit)**
  \[
  g(a) = \max(0, a)
  \]
  Several times faster than tanh.
  ‘Dying ReLU’ problem – a unit of outputting 0 always

### Exercise

1. Show networks with linear nodes in all hidden layers reduce to single-layer networks.
2. Prove that the derivative of the logistic sigmoid function \( g(a) \) is given as \( g'(a) = g(a)(1 - g(a)) \), and sketch the graph of it.
3. Explain about the learning rate \( \eta \) for the gradient descent method.
4. Explain the problem with the training of a neural network with the MSE criterion when the sigmoid function is used as the activation function.
5. **(NE)** Prove that the partial derivative of the cross-entropy error is given as
   \[
   \frac{\partial E_{\text{H}}(w)}{\partial w_i} = \frac{1}{N} \sum_{n=1}^{N} (y_n - t_n) x_{ni}.
   \]

### Summary

- Limitations of Perceptron
- Solutions to the problems
- Neural network with differentiable non-linear functions (e.g. logistic sigmoid function)
- Training of the network with the gradient descent algorithm
- Considered only a single-layer network with a single-output node