Deep Learning: Forward Propagation and Backpropagation

Lecture 02

Razvan C. Bunescu
School of Electrical Engineering and Computer Science

bunescu@ohio.edu
Neuron Function

- **Algebraic interpretation:**
  - The output of the neuron is a **linear combination** of inputs from other neurons, **rescaled by** the synaptic **weights**.
    - weights $w_i$ correspond to the synaptic weights (activating or inhibiting).
    - summation corresponds to combination of signals in the soma.
  - It is often transformed through a monotonic **activation function**.
Activation Functions

unit step \( f(z) = \begin{cases} 
0 & \text{if } z < 0 \\
1 & \text{if } z \geq 0 
\end{cases} \)

logistic \( f(z) = \frac{1}{1 + e^{-z}} \)

ReLU \( f(z) = \begin{cases} 
0 & \text{if } z < 0 \\
z & \text{if } z \geq 0 
\end{cases} \)

Perceptron
Logistic Regression
Rectified Linear Unit
ReLU and Generalizations

- It has become more common to use piecewise linear activation functions for hidden units:
  - **ReLU**: the rectifier activation \( g(a) = \max\{0, a\} \).
  - **Absolute value ReLU**: \( g(a) = |a| \).
  - **Maxout**: \( g(a_1, ..., a_k) = \max\{a_1, ..., a_k\} \).
    - needs \( k \) weight vectors instead of 1.
  - **Leaky ReLU**: \( g(a) = \max\{0, a\} + \alpha \min(0, a) \).

\( \Rightarrow \) the network computes a *piecewise linear function* (up to the output activation function).
ReLU vs. Sigmoid and Tanh

- Sigmoid and Tanh saturate for values not close to 0:
  - “kill” gradients, bad behavior for gradient-based learning.
- ReLU does not saturate for values > 0:
  - greatly accelerates learning, fast implementation.
  - fragile during training and can “die”, due to 0 gradient:
    - initialize all $b$’s to a small, positive value, e.g. 0.1.
ReLU vs. Softplus

- Softplus $g(a) = \ln(1+e^a)$ is a smooth version of the rectifier.
  - Saturates less than ReLU, yet ReLU still does better [Glorot, 2011].
Perceptron vs. Logistic vs. ReLU vs. Tanh

• **Logistic neuron:**
  – At inference time, same decision function as perceptron, for binary classification with equal misclassification costs (prove it):
    \[
    \hat{i}(x) = \begin{cases} 
    1 & \text{if } w^T x > 0 \\
    0 & \text{otherwise}
    \end{cases}
    \]
  – **Perceptron** cannot represent the XOR function:
    • Logistic neuron, ReLU, Tanh have the same limitation.

• How can we use (logistic) neurons to achieve better representational power?

Lecture 02
Universal Approximation Theorem


Let $\sigma$ be a nonconstant, bounded, and monotonically-increasing continuous function;

Let $I_m$ denote the $m$-dimensional unit hypercube $[0,1]^m$;

Let $C(I_m)$ denote the space of continuous functions on $I_m$;

**Theorem**: Given any function $f \in C(I_m)$ and $\varepsilon > 0$, there exist an integer $N$ and real constants $\alpha_i, b_i \in \mathbb{R}, w_i \in \mathbb{R}^m$, where $i = 1, \ldots, N$, such that:

$$|F(x) - f(x)| < \varepsilon, \quad \forall x \in I_m$$

where

$$F(x) = \sum_{i=1}^{N} \alpha_i \sigma(w_i^T x + b_i)$$

Lecture 02
Universal Approximation Theorem


\[ F(x) = \sum_{i=1}^{N} \alpha_i \sigma(w_i^T x + b_i) \]

\[ |F(x) - f(x)| < \varepsilon, \forall x \in I_m \]
Neural Network Model

- Put together many neurons in layers, such that the output of a neuron can be the input of another:

![Diagram of a neural network model](image)

- **input layer**
- **hidden layer**
- **output layer**
- \( n_l = 3 \) is the number of layers.
  - \( L_1 \) is the input layer, \( L_3 \) is the output layer.
- \((W, b) = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})\) are the parameters:
  - \( W^{(l)}_{ij} \) is the weight of the connection between unit \( j \) in layer \( l \) and unit \( i \) in layer \( l + 1 \).
  - \( b^{(l)}_i \) is the bias associated unit unit \( i \) in layer \( l + 1 \).
- \( a^{(l)}_i \) is the activation of unit \( i \) in layer \( l \), e.g. \( a^{(1)}_i = x_i \) and \( a^{(3)}_1 = h_{W,b}(x) \).
Inference: Forward Propagation

- The activations in the hidden layer are:
  \[ a_1^{(2)} = f(W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)} x_3 + b_1^{(1)}) \]
  \[ a_2^{(2)} = f(W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{23}^{(1)} x_3 + b_2^{(1)}) \]
  \[ a_3^{(2)} = f(W_{31}^{(1)} x_1 + W_{32}^{(1)} x_2 + W_{33}^{(1)} x_3 + b_3^{(1)}) \]

- The activations in the output layer are:
  \[ h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)} + W_{13}^{(2)} a_3^{(2)} + b_1^{(2)}) \]

- Compressed notation:
  \[ a_i^{(l)} = f(z_i^{(l)}) \text{ where } z_i^{(2)} = \sum_{j=1}^{n} W_{ij}^{(1)} x_j + b_i^{(1)} \]
Forward Propagation

• Forward propagation (unrolled):

\[
\begin{align*}
    a_1^{(2)} &= f(W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)} x_3 + b_1^{(1)}) \\
    a_2^{(2)} &= f(W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{23}^{(1)} x_3 + b_2^{(1)}) \\
    a_3^{(2)} &= f(W_{31}^{(1)} x_1 + W_{32}^{(1)} x_2 + W_{33}^{(1)} x_3 + b_3^{(1)}) \\
    h_{W,b}(x) &= a_1^{(3)} = f(W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)} + W_{13}^{(2)} a_3^{(2)} + b_1^{(2)})
\end{align*}
\]

• Forward propagation (compressed):

\[
\begin{align*}
    z^{(2)} &= W^{(1)} x + b^{(1)} \\
    a^{(2)} &= f(z^{(2)}) \\
    z^{(3)} &= W^{(2)} a^{(2)} + b^{(2)} \\
    h_{W,b}(x) &= a^{(3)} = f(z^{(3)})
\end{align*}
\]

• Element-wise application:

\[f(z) = [f(z_1), f(z_2), f(z_3)]\]
Forward Propagation

- Forward propagation (compressed):

\[
\begin{align*}
    z^{(2)} &= W^{(1)}x + b^{(1)} \\
    a^{(2)} &= f(z^{(2)}) \\
    z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\
    h_{W,b}(x) &= a^{(3)} = f(z^{(3)})
\end{align*}
\]

- Composed of two *forward propagation steps*:

\[
\begin{align*}
    z^{(l+1)} &= W^{(l)}a^{(l)} + b^{(l)} \\
    a^{(l+1)} &= f(z^{(l+1)})
\end{align*}
\]
Multiple Hidden Units, Multiple Outputs

• Write down the forward propagation steps for:

```markdown
\[ h_{w,b}(x) \]
```
Learning: Backpropagation

- Regularized sum of squares error:

\[ J(W, b, x, y) = \frac{1}{2} \| h_{W,b}(x) - y \|^2 \]

\[ J(W, b) = \frac{1}{m} \sum_{k=1}^{m} J(W, b, x^{(k)}, y^{(k)}) + \frac{\lambda}{2} \sum_{l=1}^{n_f-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (W_{ij}^{(l)})^2 \]

- Gradient:

\[
\frac{\partial J(W, b)}{\partial W_{ij}^{(l)}} = \frac{1}{m} \sum_{k=1}^{m} \frac{\partial J(W, b, x^{(k)}, y^{(k)})}{\partial W_{ij}^{(l)}} + \lambda W_{ij}^{(l)} \\
\frac{\partial J(W, b)}{\partial b_i^{(l)}} = \frac{1}{m} \sum_{k=1}^{m} \frac{\partial J(W, b, x^{(k)}, y^{(k)})}{\partial b_i^{(l)}}
\]
Backpropagation

• Need to compute the gradient of the squared error with respect to a single training example \((x, y)\):

\[
J(W, b, x, y) = \frac{1}{2} \| h_{w,b}(x) - y \|^2 = \frac{1}{2} \| a^{(n_l)} - y \|^2
\]

\[
\frac{\partial J}{\partial W_{ij}^{(l)}} = ?
\]

\[
\frac{\partial J}{\partial b_i^{(l)}} = ?
\]
Univariate Chain Rule for Differentiation

• Univariate Chain Rule:

\[ f = f \circ g \circ h = f(g(h(x))) \]

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial h} \frac{\partial h}{\partial x} \]

• Example:

\[ f(g(x)) = 2g(x)^2 - 3g(x) + 1 \]

\[ g(x) = x^3 + 2x \]
Multivariate Chain Rule for Differentiation

- Multivariate Chain Rule:

\[ f = f(g_1(x), g_2(x), \ldots, g_n(x)) \]

\[ \frac{\partial f}{\partial x} = \sum_{i=1}^{n} \frac{\partial f}{\partial g_i} \frac{\partial g_i}{\partial x} \]

- Example:

\[ f(g_1(x), g_2(x)) = 2g_1(x)^2 - 3g_1(x)g_2(x) + 1 \]

\[ g_1(x) = 3x \]

\[ g_2(x) = x^2 + 2x \]
Backpropagation: $W_{ij}^{(l)}$

- $J$ depends on $W_{ij}^{(l)}$ only through $a_i^{(l+1)}$, which depends on $W_{ij}^{(l)}$ only through $z_i^{(l+1)}$.

\[
J(W, b, x, y) = \frac{1}{2} \left\| a_1^{(n_l)} - y \right\|^2
\]

\[
a_i^{(l+1)} = f(z_i^{(l+1)})
\]

\[
z_i^{(l+1)} = \sum_{j=1}^{s_l} W_{ij}^{(l)} a_j^{(l)} + b_i^{(l)}
\]
Backpropagation: \( b_i^{(l)} \)

- \( J \) depends on \( b_i^{(l)} \) only through \( a_i^{(l+1)} \), which depends on \( b_i^{(l)} \) only through \( z_i^{(l+1)} \).

\[
J(W, b, x, y) = \frac{1}{2} \|a^{(n_l)} - y\|^2
\]

\[
a_i^{(l+1)} = f(z_i^{(l+1)})
\]

\[
z_i^{(l+1)} = \sum_{j=1}^{s_l} W_{ij} a_j^{(l)} + b_i^{(l)}
\]