Activation Functions for Hidden Layers

M. R. Hasan

Deep Learning
What We Will Cover

• Various activation (non-linear) functions for hidden layers
Various Activation Functions
Artificial Neural Network

• Previously we mentioned that an ANN uses activation functions that are continuously nonlinear for converting the weighted sum of the input.

\[ y(\vec{x}, \vec{w}) = g \left( \sum_i \vec{w}_i^T \vec{x}_i \right) \]

We will now discuss various choices for the activation function.
Artificial Neural Network

- Sigmoid or logistic function.
- The logistic sigmoid is motivated somewhat by biological neurons.
- It can be interpreted as the probability of an artificial neuron “firing” given its inputs.

\[ g(a) \equiv \sigma(a) \]

\[ \sigma(a) = \frac{1}{1 + e^{-a}} \]
Artificial Neural Network

- Sigmoid or logistic function.

\[ g(a) \equiv \sigma(a) \]

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

The **derivative** of sigmoid:

\[ g'(a) = \frac{dg}{da} = \sigma(a)[1 - \sigma(a)] \]
Artificial Neural Network

- The logistic sigmoid has a nice **biological interpretation**.
- However, it turns out that the **logistic sigmoid** can cause a neural network to **get “stuck”** during training.

\[ \sigma(a) = \frac{1}{1 + e^{-a}} \]
Artificial Neural Network

- This is due in part to the fact that if a strongly-negative input is provided to the logistic sigmoid, it outputs values very near zero.

\[
\sigma(a) = \frac{1}{1 + e^{-a}}
\]
Artificial Neural Network

• Note that ANNs use the **feed-forward activations** to calculate parameter gradients.

• Zero output from sigmoid can result in $W^{(k)}$s that are **updated less regularly** than we would like.

Thus, they could get **“stuck”** in their current state.

\[
W^{(k)} = W^{(k)} - \eta \nabla \mathcal{L}(W^{(k)})
\]

\[
\nabla \mathcal{L}(W^{(k-1)}) = a^{(k-1)^T} \delta^{(k)}
\]

\[
a_j^{(k-1)} = g(z_j^{(k-1)})
\]
Artificial Neural Network

- An alternative to the logistic sigmoid is the hyperbolic tangent, or tanh function.
- The tan is a trigonometric function that relates to a circle.
- The tanh is a hyperbolic function related to a hyperbola.

Tanh creates sigmoid-like “S” shaped curve!
Artificial Neural Network

- The tanh function is the ratio of hyperbolic sine and cosine functions.

\[ g(a) \equiv tanh(a) \]

\[ tanh(a) = \frac{\sinh(a)}{\cosh(a)} \]

Formula for computing tanh.

\[ tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \]
Artificial Neural Network

- Like the logistic sigmoid, the tanh function is also **sigmoidal** ("s"-shaped).
- Unlike logistic sigmoid, it outputs values that **range** (-1, 1).
- Thus **strongly negative inputs** to the tanh will map to **negative outputs (not zero)**.

\[
tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}
\]
Artificial Neural Network

• Additionally, **only zero-valued inputs** are mapped to near-zero outputs.

• These properties make the network **less likely to get “stuck” during training**.

\[ g(a) \equiv \tanh(a) \]

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

The **derivative** of tanh:

\[ g'(a) = \frac{dg}{da} = 1 - g(a)^2 \]
Artificial Neural Network

• **Comparison** of these 3 functions:
Artificial Neural Network

• Both the logistic and hyperbolic tangent are sigmoidal functions.
• They are closely related: \( \tanh(x) = 2\sigma(2x) - 1 \).

The problem with sigmoidal units is that they saturate across most of their domain.
Artificial Neural Network

• They **saturate to a high value** when \( x \) is very positive.
• And **saturate to a low value (or zero)** when \( x \) is very negative.
• They are **only strongly sensitive** to their input when \( x \) is **near 0**.

The **widespread saturation** of sigmoidal units can make gradient-based learning very difficult.
Artificial Neural Network

• The saturated sigmoid output will cause the derivative of sigmoid to be very small, i.e., \( g'(.) \to 0 \)

This will cause very small loss gradient:
\[ \nabla \mathcal{L}(W^{(k)}) \to 0 \]

Forward Propagation
\[ z_j^{(k)} = a^{(k-1)} W^{(k-1)} \]
\[ a_j^{(k)} = g(z_j^{(k)}) \]

\[ \delta^{(k)} = (\delta^{(k+1)} W^{(k-1)})^T \text{ no bias} \ast g'(z^{(k)}) \]
\[ \nabla \mathcal{L}(W^{(k-1)}) = a^{(k-1)^T} \delta^{(k)} \]

\[ = [\text{output} \ast \text{error}] \]
Artificial Neural Network

• For this reason, their use as hidden units in feed-forward networks is now discouraged.
Artificial Neural Network

- To overcome the limitations of the sigmoidal functions, a linear piecewise function is used in modern neural networks.
- It is called the **rectified linear unit or ReLU**.
- It is the default recommendation.

ReLU is defined by the activation function:

\[ g(z) = \max\{0, z\} \]
Artificial Neural Network

- ReLU is **easy to optimize** because it is so **similar to linear units**.
- The only difference between a linear unit and a rectified linear unit is:
- A rectified linear unit **outputs zero across half its domain**.

This makes the derivatives through a rectified linear unit **remain large** whenever the unit is active (output is positive).
Artificial Neural Network

• Applying this function to the output of a linear transformation yields a **nonlinear transformation**.

• However, the function **remains very close to linear**, in the sense that it is a piecewise linear function with **two linear pieces**.
Artificial Neural Network

- Unfortunately, the ReLU activation function is not perfect.
- It suffers from a problem known as the *dying ReLUs*.
- During training, some neurons effectively “die” meaning they stop outputting *anything other than 0*. 
Artificial Neural Network

• In some cases, we may find that half of the network’s neurons are dead, especially if we use a large learning rate.

• A neuron dies when its weights get tweaked in such a way that the weighted sum of its inputs are negative for all instances in the training set.

\[
\begin{align*}
    z_j^{(k)} &= a^{(k-1)} W^{(k-1)} \\
    a_j^{(k)} &= g(z_j^{(k)})
\end{align*}
\]
Artificial Neural Network

- When this happens, it just keeps outputting zeros, and **Gradient Descent does not affect it** anymore.
- Because the gradient of the ReLU function is zero when its input is negative.

\[
\begin{align*}
    z_j^{(k)} &= a^{(k-1)}W^{(k-1)} \\
    a_j^{(k)} &= g(z_j^{(k)})
\end{align*}
\]
Artificial Neural Network

- Two standard solutions exist to overcome the dying ReLU problem.
  - Leaky ReLU
  - Exponential Linear Unit (ELU)

\[ \text{ELU}_\alpha (z) = \begin{cases} \alpha (\exp (z) - 1) & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases} \]

\[ \text{LeakyReLU}_\alpha (z) = \max(\alpha z, z) \]
Leaky ReLU: ReLU with a small slope for the negative values.

The hyperparameter $\alpha$ defines how much the function “leaks”: it is the slope of the function for $z < 0$ and is typically set to 0.01.

$$\text{LeakyReLU}_\alpha(z) = \max(\alpha z, z)$$

ELU: It takes on negative values when $z < 0$, which allows the unit to have an average output closer to 0 and helps alleviate the vanishing gradient problem.

The hyperparameter $\alpha$ defines the value that the ELU function approaches when $z$ is a large negative number.

$$\text{ELU}_\alpha(z) = \begin{cases} \alpha(\exp(z) - 1) & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$