ANN
Activation function - comparison

Dominik Lewy
Neural Network – Hyperparameters

What can be tested:
• Architecture (depth, number of neurons, activation function)
• Learning rate
• Weight initialization
• Regularization
**Activation functions – Forward Pass**

Example: H1

\[
\text{net}_{H1} = w_1 * I_1 + w_2 * I_2 + b_1 * 1
\]

\[
\text{out}_{H1} = \text{activation}(\text{net}_{H1})
\]
Activation functions

**Forward Pass**

Example: \( O_1 \)

\[
net_{O1} = w_5 \cdot out_{H1} + w_6 \cdot out_{H2} + b_1 \cdot 1
\]

\[
out_{O1} = activation(net_{O1})
\]
Activation functions – Error

\[ E_{\text{total}} = E_{O1} + E_{O2} \]

\[ E_{O1} = f(\text{target}_{O1} - \text{output}_{O1}) \]
Activation functions – Backward Pass

\[ E_{total} = E_{O1} + E_{O2} \]

\[
\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{O1}}{\partial out_{O1}} \cdot \frac{\partial out_{O1}}{\partial net_{O1}} \cdot \frac{\partial net_{O1}}{\partial w_5}
\]
Activation functions – Backward Pass

\[
\frac{\partial E_{\text{total}}}{\partial w_1} = \left( \frac{\partial E_{01}}{\partial out_{01}} \right) \left( \frac{\partial out_{01}}{\partial net_{01}} \frac{\partial net_{01}}{\partial out_{h1}} + \frac{\partial E_{02}}{\partial out_{02}} \right) \left( \frac{\partial out_{02}}{\partial net_{02}} \frac{\partial net_{02}}{\partial out_{h1}} \right) \left( \frac{\partial out_{h1}}{\partial net_{h1}} \frac{\partial net_{h1}}{\partial w_1} \right)
\]
Activation functions - Identity

\[ a_j^i = \sigma(z_j^i) = z_j^i \]
• Constant gradient of identity function – weight updates are independent of input values

• No non-linearity – we lose the ability to stack layers because the output of the network remains a linear combination of the input
Activation functions - Sigmoid

\[ a_j^i = \sigma(z_j^i) = \frac{1}{1 + \exp(-z_j^i)} \]
Activation function – Dominik Lewy

Activation functions - Sigmoid

• Small gradient of sigmoid function – makes it difficult to train the network to acceptable level

Gradient value of identity function

Gradient value of sigmoid function

• Incorrect weight initialization and outliers in data can lead to neuron saturation, where most neurons of the network then become saturated and almost no learning will take place

Gradient value of sigmoid function

For net_01 lower than -10 and greater than 10:

\[
\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{01}}{\partial out_{01}} \times \frac{\partial out_{01}}{\partial net_{01}} \times \frac{\partial net_{01}}{\partial w_5} \\
\sim 0
\]
Vanishing gradient problem

Max value of \( \frac{\partial \text{Sigmoid}(z)}{\partial z} = \frac{1}{4} \)

One hidden layer:

\[
\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial \text{output}} \times \frac{\partial \text{output}}{\partial \text{hidden1}} \times \frac{\partial \text{hidden1}}{\partial w_1}
\]

Two hidden layers:

\[
\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial \text{output}} \times \frac{\partial \text{output}}{\partial \text{hidden2}} \times \frac{\partial \text{hidden2}}{\partial \text{hidden1}} \times \frac{\partial \text{hidden1}}{\partial w_1}
\]

Three hidden layers:

\[
\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial \text{output}} \times \frac{\partial \text{output}}{\partial \text{hidden3}} \times \frac{\partial \text{hidden3}}{\partial \text{hidden2}} \times \frac{\partial \text{hidden2}}{\partial \text{hidden1}} \times \frac{\partial \text{hidden1}}{\partial w_1}
\]

Best case scenario:

\[
\frac{1}{4} \times \ldots
\]

\[
\frac{1}{4} \times \frac{1}{4} \times \ldots
\]

\[
\frac{1}{4} \times \frac{1}{4} \times \frac{1}{4} \times \ldots
\]
Activation functions - Sigmoid

• The sigmoid function "squashes" values to the range between 0 and 1 – compared to (-inf, inf) for linear function. So we have our activations bound in a range and the activations won't blow up
  • In accordance with the analogy to human brain and neurons
  • Interpretation in terms of probability

• The output is not zero centered – all the weights will be either increased or decreased because the weights update depends only on upstream gradient

\[
\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{O1}}{\partial out_{O1}} \cdot \frac{\partial out_{O1}}{\partial net_{O1}} \cdot \frac{\partial net_{O1}}{\partial w_5} + \]

\[
net_{O1} = w_5 \cdot out_{H1} + w_6 \cdot out_{H2} + b_1 \cdot 1
\]

\[
\frac{\partial net_{O1}}{\partial w_5} = out_{h1} + \]

11/12/2017
Activation functions – Tanh

\[ a_j^i = \sigma(z_j^i) = \tanh(z_j^i) \]
Activation function – Dominik Lewy

Activation functions - Tanh

• Tanh neuron is simply a scaled sigmoid neuron, in particular the following holds

\[ \tanh(x) = 2 \text{sigmoid}(2x) - 1 \]

• Incorrect weight initialization and outliers in data can lead to neuron saturation, where most neurons of the network then become saturated and almost no learning will take place

![Gradient value of tanh function](image)

For net\textsubscript{01} lower than -10 and greater than 10:

\[
\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{01}}{\partial \text{out}_{01}} \ast \frac{\partial \text{out}_{01}}{\partial \text{net}_{01}} \ast \frac{\partial \text{net}_{01}}{\partial w_5}
\]

\[\sim 0\]

• Problems resolved by Tanh
  • The output is not zero centered
  • Small gradient of sigmoid function
Activation function

Activation functions - ReLU

\[ a_j^i = \sigma(z_j^i) = \max(0, z_j^i) \]
Activation functions - ReLU

- It was found to greatly accelerate the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. It is argued that this is due to its linear, non-saturating form.

- Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.

- Unfortunately, ReLU units can be fragile during training and can "die"
  - On the other hand it introduces sparsity into the net
  - There are variations of ReLU that deal with this problem like Leaky ReLU

$$a^i_j = \sigma(z^i_j) = \max(0.1z^i_j, z^i_j)$$
"What activation function should I use?"

- Treat ReLU as your go-to function
  - monitor the fraction of "dead" units in a network
- If the result are not good enough try Leaky ReLU
- Never use sigmoid
- You can try tanh, but expect it to work slower and worse than ReLU

<table>
<thead>
<tr>
<th>Function</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity</td>
<td></td>
<td>Blown up activations</td>
</tr>
<tr>
<td>Sigmoid</td>
<td>Output in range (0,1)</td>
<td>Saturated Neurons, Not zero centered,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Small gradient, Vanishing gradient</td>
</tr>
<tr>
<td>Tanh</td>
<td>Zero centered, Output in range (-1,1)</td>
<td>Saturated Neurons</td>
</tr>
<tr>
<td>ReLU</td>
<td>Computational efficiency, Accelerated convergence</td>
<td>Dead Neurons, Not zero centered</td>
</tr>
</tbody>
</table>
Activation functions - Summary
Activation functions - Softmax

\[ a_j^i = \sigma(z_j^i) = \frac{\exp(z_j^i)}{\sum_{k=1}^{K} \exp(z_k^i)} \]

Properties:
- It is used as last layer for categorization problems
- It returns a probability of each output
- All the probabilities sum up to one
It’s time to look at some experiments. MNIST data set was used for this exercise. Nets with the same architecture, differing only by the activation function used in the hidden layer, were trained for 100 epochs.
Experiment

Accuracy achieved on test dataset

11/12/2017
## Experiment

<table>
<thead>
<tr>
<th>Function</th>
<th>Sigmoid</th>
<th>Tanh</th>
<th>ReLU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations to reach 0.8 accuracy</td>
<td>267</td>
<td>174</td>
<td>154</td>
</tr>
<tr>
<td></td>
<td>x1.5 faster</td>
<td>x1.7 faster</td>
<td></td>
</tr>
<tr>
<td>Iterations to reach 0.9 accuracy</td>
<td>3852 (3781 first)</td>
<td>1254</td>
<td>765 (692 first)</td>
</tr>
<tr>
<td></td>
<td>x3 faster</td>
<td>x5 faster</td>
<td></td>
</tr>
<tr>
<td>Iterations to reach 0.95 accuracy</td>
<td>17113 (16249 first)</td>
<td>6417 (6053 first)</td>
<td>3705 (3694 first)</td>
</tr>
<tr>
<td></td>
<td>x2.7 faster</td>
<td>x4.6 faster</td>
<td></td>
</tr>
<tr>
<td>Average accuracy achieved</td>
<td>0.974</td>
<td>0.975</td>
<td>0.975</td>
</tr>
<tr>
<td>Training time of 1 epoch</td>
<td>0.0224</td>
<td>0.0236</td>
<td>0.0227</td>
</tr>
</tbody>
</table>
**Gradient Clipping**

Gradient Clipping is a technique to prevent exploding gradients in very deep networks, typically Recurrent Neural Networks. There exist various ways to perform gradient clipping, but the a common one is to normalize the gradients of a parameter vector when its L2 norm exceeds a certain threshold according to

\[
\text{new gradients} = \text{gradients} \times \frac{\text{threshold}}{\text{12 norm(gradient)}}.
\]

\[
L_2 = \|(x_1, x_2)\|_2 = \left(\sum_{i=1}^{2} x_i^2\right)^{1/2}
\]


\[
L_1 = \|(x_1, x_2)\|_1 = \sum_{i=1}^{2} |x_i|
\]

\[
L_\infty = \|(x_1, x_2)\|_\infty = \max_i |x_i|
\]