CENG 501
Deep Learning
Week 4

Sinan Kalkan
Regression loss

\[ L = \frac{1}{N} \sum_i \sum_j (s_{ij} - y_{ij})^2 + \lambda \sum_i \sum_j w_{i,j}^2 \]

In general:
\[ \sum_j |s_j - y_j|^q \]

• \( q = 1 \): Absolute Value Loss
• \( q = 2 \): Square Error Loss.

Figure 1.29  Plots of the quantity \( L_q - |y - t|^q \) for various values of \( q \). Bishop
\[ \theta^* = \arg \min_{\theta} L(x; \theta) \]

\[ \theta_{t+1} \leftarrow \theta_t - \eta \nabla_{\theta} L(x; \theta) \]
\[ \theta_{t+1} \leftarrow \theta_t - \eta \nabla_{\theta} L(x; \theta) \]

\[ \mathcal{L}(x; \theta) = \frac{\partial L(x; \theta)}{\partial \theta} \]

Learning Rate (Step Size)
Gradient Descent

• True Gradient Descent
  • Calculate the loss & the gradient on the whole dataset
  • Then make the update

• Stochastic Gradient Descent
  • Calculate the loss & the gradient on examples one at a time
  • Update the weights after each example

• Batch Gradient Descent
  • Calculate the loss & the gradient on a set of examples (batch)
  • Update the weights after each batch
Gradient descent

https://en.wikipedia.org/wiki/Gradient_descent

(Goodfellow vd., 2016)
Derive the gradients of hinge loss

\[
\frac{\partial L_i}{\partial w_{mk}} = ?
\]

\[
\frac{\partial L_i}{\partial w_{mk}} = \frac{\partial L_i}{\partial e_m} \frac{\partial e_m}{\partial s_m} \frac{\partial s_m}{\partial w_{mk}}
\]

\[
\frac{\partial L_i}{\partial w_{mk}} = \mathbb{I}(s_m - s_{y_i} + \Delta > 0) x_{ik}
\]

1 \hspace{2cm} x_{ik}

Previously on CENG501!

\[x_i\]

\[x_{i1}\]

\[
\ldots
\]

\[x_{ik}\]

\[
\ldots
\]

\[x_{in}\]

1. What happens if that’s not the case? See the next page.

\[L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)
\]

\[
e_j
\]

This assumed that \(m \neq y_i\). What happens if that’s not the case? See the next page.
Derive the gradients of hinge loss

\[
\frac{\partial L_i}{\partial w_{y_i k}} = ?
\]

\[
\frac{\partial L_i}{\partial w_{y_i k}} = \sum_{j \neq y_i} \frac{\partial L_i}{\partial e_j} \frac{\partial e_j}{\partial s_{y_i}} \frac{\partial s_{y_i}}{\partial w_{y_i k}}
\]

\[
1 \quad \mathbb{I}(s_j - s_{y_i} + \Delta > 0)(-1)
\]

\[
\frac{\partial L_i}{\partial w_{y_i k}} = \sum_{j \neq y_i} \mathbb{I}(s_j - s_{y_i} + \Delta > 0)(-1)x_{ik}
\]

\[
L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)
\]

\[
s_{y_i} = w_{y_i} x_i = \sum_k w_{y_i k} x_{ik}
\]

Sinan Kalkan
Non-linear Classification/Regression

Previously on CENG501!

\[ t = \sum w_i x_i + b \]

\[ y = f(x) = f(x; \theta) \]

\[ = g\left( \sum_i w_i x_i + b \right) = g(w \cdot x) \]
\[
\begin{align*}
a_1 &= g(W_0 x^T) \\
a_2 &= g(W_1 a_1^T) \\
a_t &= g(W_{t-1} a_{t-1}^T)
\end{align*}
\]

\[
f() = g\left(g\left(g\left(g(\ldots)\right)\right)\right)
\]
\[ L(x; \theta) = d(y, f(x; \theta)) \]

\[ \theta_{t+1} \leftarrow \theta_t - \eta \nabla_{\theta} L(x; \theta) \]

\[ \theta^* = \arg \min_{\theta} L(x; \theta) \]

\[ \nabla_{\theta} L(x; \theta) = \frac{\partial L(x; \theta)}{\partial \theta} \]
Previously on CENG501!

\[
\frac{\partial L(x; \theta)}{\partial w_j} = \frac{\partial L}{\partial \hat{y}_m} \frac{\partial \hat{y}_m}{\partial g^m_l} \frac{\partial g^m_l}{\partial t_m} \frac{\partial t_m}{\partial g^{l-1}_k} \frac{\partial g^{l-1}_k}{\partial t_k} \ldots
\]

\[
\frac{\partial L(x; \theta)}{\partial w_{mi}} = \frac{\partial L}{\partial \hat{y}_m} \frac{\partial \hat{y}_m}{\partial g^m_l} \frac{\partial g^m_l}{\partial t_m} \frac{\partial t_m}{\partial g^{l-1}_k} \frac{\partial g^{l-1}_k}{\partial t_k} \ldots
\]

Backpropagation
Previously on CENG501!

\[
\frac{\partial L(x; \theta)}{\partial w_{ij}} = \frac{\partial L(x; \theta)}{\partial a_i^l} \frac{\partial a_i^l}{\partial w_{ij}}
\]

\[
\frac{\partial L(x; \theta)}{\partial a_i^{l-1}} = \sum_j \frac{\partial L(x; \theta)}{\partial a_j^l} \frac{\partial a_j^l}{\partial a_i^{l-1}}
\]
Importance of increasing layers

• Continuous functions:
  • Every bounded continuous function can be approximated with small error with two layers

• Arbitrary functions:
  • Three layers can approximate any arbitrary function

• Why do we need deep layers then?
  • If the problem has a hierarchical nature, more layers yield better performance
  • Lin vd., “Why does deep and cheap learning work so well?”, 2017.

Multi-layered Fully-connected Networks

To be able to have solutions for linearly non-separable cases, we need a non-linear and differentiable unit, e.g.:

$$\hat{y} = \sigma(w \cdot x)$$

where

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Sigmoid (logistic) function
- Output is in range (0,1)
- Since it maps a large domain to (0,1) it is also called squashing function
- Alternatives: $tanh$
A neuron with sigmoid function

Previously on CENG501!
Why do we need to learn backpropagation?

• “Many frameworks implement backpropagation for us, why do we need to learn?”
  • This is not a blackbox. There are many problems/issues involved. You can only deal with them if you have a good understanding of backpropagation.

https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b#.7zawffou2
Black Magic in Deep Learning: How Human Skill Impacts Network Training

Kanav Anand$^1$
kanav92@gmail.com

Ziqi Wang$^1$
z.wang-8@tudelft.nl

Marco Loog$^{12}$
M.Loog@tudelft.nl

Jan van Gemert$^1$
vangemert@tudelft.nl

$^1$Delft University of Technology, Delft, The Netherlands
$^2$University of Copenhagen, Copenhagen, Denmark

Abstract

How does a user’s prior experience with deep learning impact accuracy? We present an initial study based on 31 participants with different levels of experience. Their task is to perform hyperparameter optimization for a given deep learning architecture. The results show a strong positive correlation between the participant’s experience and the final performance. They additionally indicate that an experienced participant finds better solutions using fewer resources on average. The data suggests furthermore that participants with no prior experience follow random strategies in their pursuit of optimal hyperparameters. Our study investigates the subjective human factor in comparisons of state of the art results and scientific reproducibility in deep learning.
The Model

Hidden activations:
\[ h_{ij} = \sigma(w^h_{ji} \cdot x_i) = \sigma(net^h_{ij}) \]

Output layer:
\[ \hat{y}_{ic} = \sigma(w^o_{ic} \cdot h_i) = \sigma(net^o_{ic}) \]

The loss function:
\[ L(\theta) = \frac{1}{2} \sum_{i=1}^{N} \sum_{c \in C} (\hat{y}_{ic} - y_{ic})^2 \]
- For one sample:
\[ L_i(\theta) = \frac{1}{2} \sum_{c \in C} (\hat{y}_{ic} - y_{ic})^2 \]

\[ h_i = \sigma(W^h x_i) \quad \hat{y}_i = \sigma(W^o h_i) \]
Backpropagation

For each output unit $c$, calculate its grad term $\delta^o_{ic}$:

$$\delta^o_{ic} = \frac{\partial L_i}{\partial net^o_{ic}} = \frac{\partial L_i}{\partial \hat{y}_{ic}} \frac{\partial \hat{y}_{ic}}{\partial net^o_{ic}} = (\hat{y}_{ic} - y_{ic})\hat{y}_{ic}(1 - \hat{y}_{ic})$$

For each hidden unit $j$, calculate its grad term $\delta^h_{ij}$:

$$\delta^h_{ij} = \frac{\partial L_i}{\partial net^h_{ij}} = \frac{\partial L_i}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial net^h_{ij}} = \left(\sum_{c \in C} \frac{\partial L_i}{\partial net^o_{ic}} \frac{\partial net^o_{ic}}{\partial h_{ij}}\right) h_{ij}(1 - h_{ij})$$

$$= (\sum_{c \in C} \delta^o_{ic} w_{cj}) h_{ij}(1 - h_{ij})$$

Update weight $w^o_{jk}$ in the output layer:

$$w^o_{jk} = w^o_{jk} - \eta \delta^o_{ij} h_{ik}$$

Update weight $w^h_{jk}$ in the hidden layer:

$$w^h_{jk} = w^h_{jk} - \eta \delta^h_{ij} x_{ik}$$

The Model

Hidden activations: $h_{ij} = \sigma(w^h_{ij} \cdot x_i) = \sigma(net^h_{ij})$

Output layer: $\hat{y}_{ic} = \sigma(w^o_{ic} \cdot h_i) = \sigma(net^o_{ic})$

The loss function:

$$L(\theta) = \frac{1}{2} \sum_{i=1}^{N} \sum_{c \in C} (\hat{y}_{ic} - y_{ic})^2$$

- For one sample:

$$L_t(\theta) = \frac{1}{2} \sum_{c \in C} (\hat{y}_{ic} - y_{ic})^2$$

Spring 2021
Forward pass

```python
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

https://cs231n.github.io/
Backward pass

\[
\text{loss} = 0.5 \times \text{np.sum}((\text{out} - \text{y})^2)
\]
\[
\text{dout} = (\text{out} - \text{y})
\]
\[
\text{dW3} = \text{np.dot} (\text{dout}, h2.T)
\]

...
Backpropagation vs. numerical differentiation

What are their complexities?

• Backpropagation:
  • $O(|\theta|)$

• Numerical differentiation
  • $O(|\theta|^2)$
Neural Engineering

• Loss functions
• On optimization
• Activation functions
• Capacity, convergence
• Preprocessing
• ...
Information Entropy

- Number of bits to represent a coin-pair:
  \[ \log_2 4 = 2 \]

- In fact, this is:
  \[ \log_2 \frac{1}{p_{\text{coin}}} = \log_2 \frac{1}{0.25} = 2 \]

- Optimal number of bits to represent an event with probability \( p \):
  \[ \log_2 \frac{1}{p} \]
Information Entropy

• For an optimal setting, we can assign bits to code information based on their probabilities.

• The smallest number of bits on avg. to represent an event with probability $p$: $\log_2 \frac{1}{p}$

• Optimal bits to represent fiat cars:

$$b_{\text{fiat}} = \log_2 \frac{1}{p_{\text{fiat}}}$$

• The optimal encoding then requires:

$$H(p) = E_p \left[ \log_2 \frac{1}{p} \right] = \sum_i p_i \log_2 \frac{1}{p_i} = -\sum_i p_i \log_2 p_i$$

<table>
<thead>
<tr>
<th>Car</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fiat</td>
<td>0.80</td>
</tr>
<tr>
<td>Mazda</td>
<td>0.15</td>
</tr>
<tr>
<td>Tesla</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Example from: https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/
Cross-entropy, Entropy

• Entropy assumes that the data follows the «correct» distribution.

• If the estimated/current distribution (call it $q$) is somewhat “wrong”, how can we quantify the number of bits required?

Entropy:

$$H(p) = \sum_i p_i \log_2 \frac{1}{p_i} = - \sum_i p_i \log_2 p_i$$

Cross-Entropy

$$H(p, q) = E_p \left[ \log_2 \frac{1}{q} \right] = \sum_i p_i \log_2 \frac{1}{q_i} = - \sum_i p_i \log_2 q_i$$

Example from: https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/
Today

• Continue with loss functions
  • Cross-entropy
  • More on softmax
  • Regression losses
  • Visualizing loss functions

• Activation functions
Administrative Issues

• Project paper selection (15 April)
  https://docs.google.com/spreadsheets/d/1u_4cYR5OFpJlwd8IXpH_SVLQuOh9i-Fgm6prz7eRdT/edit?usp=sharing

• Programming Assignment 1 (PA1):
  • Tentative date: 28 March – 15 April.
  • Announced!

• Midterm exam:
  • Tentative date: The week of 11-15 April.
Kullback-Leibler Divergence

• Difference between cross-entropy and entropy (this is zero when \( p_i \) equals \( q_i \)):

\[
KL(p \| q) = \sum_i p_i \log \frac{1}{q_i} - \sum_i p_i \log \frac{1}{p_i} \\
= \sum_i p_i \log \frac{p_i}{q_i}
\]
More on xentropy, entropy and KL-divergence

https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/

https://www.youtube.com/watch?v=ErfnhcEV1O8
Softmax classifier – cross-entropy loss

- Cross-entropy: $H(p, q) = E_p[- \log q] = - \sum_j p_j \log q_j$
- In our case,
  - $p$ denotes the correct probabilities of the categories. In other words, $p_j = 1$ for the correct label and $p_j = 0$ for other categories.
  - $q$ denotes the estimated probabilities of the categories
- But, our scores are not probabilities!
  - One solution: Softmax function: $sm(s_i) = \frac{e^{s_i}}{\sum_j e^{s_j}}$
  - It maps arbitrary ranges to probabilities
- Using the normalized values, we can define the cross-entropy loss for classification problem now:
  $$L_i = - \log_e \left( \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \right) = -s_{y_i} + \log e \sum_j e^{s_j}$$
Derive the gradients of NLL loss

\[
\frac{\partial L_i}{\partial w_{jk}} = ?
\]

If \( j = y_i \):

\[
\frac{\partial L_i}{\partial w_{jk}} = \frac{\partial L_i}{\partial p_{y_i}} \frac{\partial p_{y_i}}{\partial s_{y_i}} \frac{\partial s_{y_i}}{\partial w_{jk}}
\]

\[
= -\frac{1}{p_j} p_j(1 - p_j) x_{ik}
\]

\[
\frac{\partial L_i}{\partial w_{jk}} = (p_j - 1)x_{ik}
\]

This assumed that \( j = y_i \). What happens if that’s not the case? See the next page.
Derive the gradients of NLL loss

\[ \frac{\partial L_i}{\partial w_{jk}} = \ ? \]

If \( j \neq y_i \):

\[ \frac{\partial L_i}{\partial w_{jk}} = \frac{\partial L_i}{\partial p_{y_i}} \frac{\partial p_{y_i}}{\partial s_j} \frac{\partial s_j}{\partial w_{jk}} \]

\[-\frac{1}{p_{y_i}} \]

\[ \frac{\partial L_i}{\partial w_{jk}} = p_j x_{ik} \]

The Model

\[ s_j = w_j x_i \]

\[ = \sum_k w_{jk} x_{ik} \]

\[ L_i = -\log_e p_{y_i} \]

\[ p_{y_i} = \frac{e^{s_y_i}}{\sum_j e^{s_j}} \]
logistic loss

• A special case of cross-entropy for binary classification:

\[ H(p, q) = - \sum_j p_j \log q_j = -p \log q - (1 - p) \log(1 - q) \]

• Softmax function reduces to the logistic function (see [1] for the derivation):

\[ \frac{1}{1 + e^{-x}} \]

Softmax classifier: One interpretation

- Information theory
  - Cross-entropy between a true distribution and an estimated one:
    \[ H(p, q) = -\sum_x p(x) \log q(x). \]
  - In our case, \( p = [0, \ldots, 1, 0, \ldots] \), containing only one 1, at the correct label.
  - Since \( H(p, q) = H(p) + D_{KL}(p||q) \), we are minimizing the Kullback-Leibler divergence.

\[ D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \]

http://cs231n.github.io/
Softmax classifier: Another interpretation

• Probabilistic view

\[ P(y_i \mid x_i; W) = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}}. \]

• In our case, we are minimizing the negative log likelihood.
• This corresponds to Maximum Likelihood Estimation (MLE).
• See the following for a quick look at MLE:
  https://wiseodd.github.io/techblog/2017/01/01/mle-vs-map/

http://cs231n.github.io/
SVM loss vs. cross-entropy loss

• SVM is happy when the classification satisfies the margin
  • Ex: if score values = [10, 9, 9] or [10, -10, -10]
    • SVM loss is happy if the margin is 1

• Cross-entropy is more ambitious: it wants more than a margin
More on softmax

• Softmax is a smooth version of arg max:
  \[
  \text{arg max} \ (s_1, s_2, \ldots, s_n) = (y_1, y_2, \ldots, y_n) = (0, 0, \ldots, 0, 1, 0 \ldots 0)
  \]

• The base in softmax can be changed to have more skewed (or distributed) values for the largest input (\(e^\beta = b\)):
  \[
  sm_\beta(s_i) = \frac{e^{\beta s_i}}{\sum_j e^{\beta s_j}}
  \]

• When \(\beta \to \infty\), softmax converges to arg max.
  e.g.
  \[
  sm_{\beta=1}([1, 1.1]) = [0.475, 0.524] \\
  sm_{\beta=2}([1, 1.1]) = [0.451, 0.550] \\
  sm_{\beta=5}([1, 1.1]) = [0.378, 0.622] \\
  sm_{\beta=100}([1, 1.1]) = [4.5e-05, 9.9e-01]
  \]
More on softmax

• Softmax with temperature is softmax with $\beta = 1/T$:

$$sm_{1/T}(s_i) = \frac{e^{s_i/T}}{\sum_j e^{s_j/T}}$$

• Interpretation:
  • Increase $T$ => decrease $\beta$ => decrease the peak around the largest value.
  • Higher $T$ yields more confident (may be over confident) probability distribution.
  • Especially in training sequence models where we perform sampling from the output distribution, in order to allow diversity, we can increase $T$. 
More on softmax

• Exponentials may become very large. A trick:

\[
\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = \frac{Ce^{f_{y_i}}}{C \sum_j e^{f_j}} = \frac{e^{f_{y_i}} + \log C}{\sum_j e^{f_j} + \log C}
\]

• Set \( \log C = -\max_j f_j \).

See the following link for more information:

http://www.nowozin.net/sebastian/blog/streaming-log-sum-exp-computation.html

http://cs231n.github.io/
Classification Loss functions

• **A single correct** label case (classification):
  • Hinge loss:
    • $L_i = \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + 1)$
  • Cross-entropy (negative log-likelihood) loss:
    • $L_i = -\log \left( \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right)$
Classification Loss functions

- **Many correct** labels case:
  - Binary prediction for each label, independently:
    - \( L_i = \sum_j \max(0, 1 - y_{ij} f_j) \)
    - \( y_{ij} = +1 \) if example \( i \) is labeled with label \( j \); otherwise \( y_{ij} = -1 \).

- Alternatively, train logistic loss for each label (0 or 1):

\[
L_i = \sum_j y_{ij} \log(\sigma(f_j)) + (1 - y_{ij}) \log(1 - \sigma(f_j))
\]
0-1 Loss

- Minimize the # of cases where the prediction is wrong:

$$L = \sum_{i} 1(f(x_i; W, b)_{y_i} \neq y_i)$$

Or equivalently,

$$L = \sum_{i} 1(y_i f(x_i; W, b)_{y_i} < 0)$$
Absolute Value Loss, Squared Error Loss

\[ L_i = \sum_j |s_j - y_j|^q \]

- \( q = 1 \): absolute value loss
- \( q = 2 \): square error loss.

Figure 1.29 Plots of the quantity \( L_q = |y - t|^q \) for various values of \( q \). Bishop
Structured Loss functions

• What if we want to predict a graph, tree etc.? Something that has structure.
  • **Structured loss**: formulate loss such that you minimize the distance to a correct structure
Visualizing Loss Functions

• If you look at one of the example loss functions:

\[ L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + 1) \]

• Since \( W \) has too many dimensions, this is difficult to plot.

• We can visualize this for one weight direction though, which can give us some intuition about the shape of the function.
  
  • E.g., start from an arbitrary \( W_0 \), choose a direction \( W_1 \) and plot \( L(W_0 + \alpha W_1) \) for different values of \( \alpha \).
Visualizing Loss Functions

• You see that this is a convex function.
  • Nice and easy for optimization
• When you combine many of them in a neural network, it becomes non-convex.

http://cs231n.github.io/
Another approach for visualizing loss functions

• 0-1 loss:

\[ L = 1(f(x) \neq y) \]

or equivalently as:

\[ L = 1(yf(x) < 0) \]

• Square loss:

\[ L = (f(x) - y)^2 \]

in binary case:

\[ L = (1 - yf(x))^2 \]

• Hinge-loss

\[ L = \max(1 - yf(x), 0) \]

• Logistic loss (binary Cross Entropy Loss):

\[ L = -\log \left( \frac{1}{1 + e^{-yf(x)}} \right) \]

Various loss functions used in classification. Here \( t = yf(x) \).

Rosasco et al., 2003

All losses approximate 0-1 loss

Various loss functions used in classification. Here $t = yf(x)$.

Rosacco et al., 2003
Sum up

• 0-1 loss is not differentiable/helpful at training
  • It is used in testing
• Other losses try to cover the “weakness” of 0-1 loss
• Hinge-loss imposes weaker constraint compared to cross-entropy
• For classification: use hinge-loss or cross-entropy loss
• For regression: use squared-error loss, or absolute difference loss
Activation Functions
Activation function: Sigmoid/logistic

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

• Output is in range (0,1)
• Since it maps a large domain to (0,1) it is also called squashing function
• Simple derivative
  \[ \frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x)) \]

Fig: https://medium.com/@omkar.nallagoni/activation-functions-with-derivative-and-python-code-sigmoid-vs-tanh-vs-relu-44d23915c1f4
Activation function: \( \tanh \)

\[
\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}
\]

- Output is in range \((-1,1)\)
- A squashing function
- Simple derivative
  \[
  \frac{d\tanh(x)}{dx} = (1 - \tanh^2(x))
  \]

Fig: https://medium.com/@omkar.nallagoni/activation-functions-with-derivative-and-python-code-sigmoid-vs-tanh-vs-relu-44d23915c1f4
Activation Functions: Sigmoid vs. tanh

- Sigmoid is a historically important activation function
  - But nowadays, rarely used
  - Drawbacks:
    1. It gets saturated, if the activation is close to zero or one
       - This leads to very small gradient, which affects the feedback to earlier layers
       - Initialization is also very important for this reason
    2. It is not zero-centered (not very severe)

- Tanh
  - Similar to the sigmoid, it saturates
  - However, it is zero-centered.
  - Tanh is always preferred over sigmoid
  - Note: \( \tanh(x) = 2\sigma(2x) - 1 \)

They are both non-convex!

Fig: https://medium.com/@omkar.nallagoni/activation-functions-with-derivative-and-python-code-sigmoid-vs-tanh-vs-relu-44d23915c1f4
**Activation Functions: Rectified Linear Units (ReLU)**

Vinod Nair and Geoffrey Hinton (2010). Rectified linear units improve restricted Boltzmann machines, ICML.

\[ f(x) = \max(0, x) \]

Derivative: \( 1(x > 0) \)
Activation Functions: ReLU – biological motivation

Figure 1: Left: Common neural activation function motivated by biological data. Right: Commonly used activation functions in neural networks literature: logistic sigmoid and hyperbolic tangent (tanh).

Activation Functions: 
ReLU – biological motivation

Hinton argues that this is a form of model averaging

- Rectified linear units are much faster to compute than the sum of many logistic units.
- They learn much faster than ordinary logistic units and they produce sparse activity vectors.
Activation Functions:

ReLU: Pros and Cons

• Pros:
  • It converges much faster (claimed to be 6x faster than sigmoid/tanh)
    • It overfits very fast and when used with e.g. dropout, this leads to very fast convergence
    • It is simpler and faster to compute (simple comparison)

• Cons:
  • A ReLU neuron may “die” during training
  • A large gradient may update the weights such that the ReLU neuron may never activate again
    • Avoid large learning rate

• See also:
  http://www.jefkine.com/general/2016/08/24/formulating-the-relu/
Activation Functions:
Leaky ReLU

• \( f(x) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x) \)
  • When \( x \) is negative, have a non-zero slope (\( \alpha \))

• If you learn \( \alpha \) during training, this is called parametric ReLU (PReLU)

Andrew L. Maas, Awni Y. Hannun, Andrew Y. Ng (2014). Rectifier Nonlinearities Improve Neural Network Acoustic Models

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2015) Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification
Activation Functions:
maxout

• $\max(w_1^T x + b_1, w_2^T x + b_2)$

• ReLU, Leaky ReLU and PReLU are special cases of this
• Drawback: More parameters to learn!

“Maxout Networks” by Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, Yoshua Bengio, 2013.
Activation Functions: Softplus

- A smooth approximation to the ReLU unit:
  \[ f(x) = \ln(1 + e^x) \]

- Its derivative is the sigmoid function:
  \[ f'(x) = 1/(1 + e^{-x}) \]
Activation Functions: Swish: A Self-Gated Activation Function

“The choice of activation functions in deep networks has a significant effect on the training dynamics and task performance. Currently, the most successful and widely-used activation function is the Rectified Linear Unit (ReLU). Although various alternatives to ReLU have been proposed, none have managed to replace it due to inconsistent gains. In this work, we propose a new activation function, named Swish, which is simply $f(x) = x \cdot \text{sigmoid}(x)$. Our experiments show that Swish tends to work better than ReLU on deeper models across a number of challenging datasets. For example, simply replacing ReLUs with Swish units improves top-1 classification accuracy on ImageNet by 0.9% for Mobile NASNet-A and 0.6% for Inception-ResNet-v2. The simplicity of Swish and its similarity to ReLU make it easy for practitioners to replace ReLUs with Swish units in any neural network.”
Activation Functions:

Exponential Linear Unit

- Similar to the Swish function

\[
\text{ELU} = \begin{cases} 
  x & x \geq 0 \\
  \alpha(e^x - 1) & x < 0 
\end{cases}
\]
Activation Functions: To sum up

• Don’t use sigmoid
• If you really want to, use tanh but it is worse than ReLU and its variants
• ReLU: be careful about dying neurons
• Leaky ReLU and Maxout: Worth trying
average loss: 0.012535083449861551

https://cs.stanford.edu/people/karpathy/convnetjs/demo/regression.html
DEMO 2

http://playground.tensorflow.org/#activation=tanh&regularization=L2&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.24725&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification
Interactive introductory tutorial