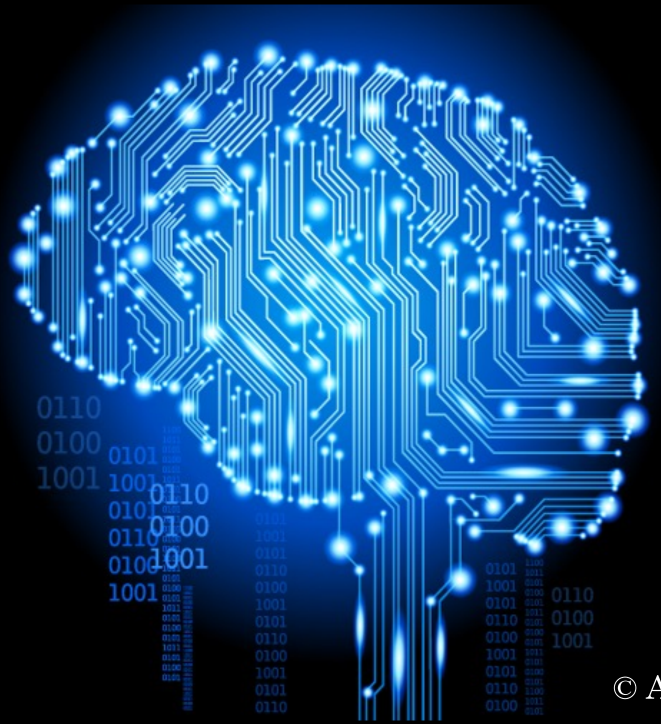
Deep Learning
What society thinks I do
What my friends think I do
What other computer scientists think I do
What mathematicians think I do
What I think I do
from theano import *
What I actually do

https://twitter.com/born2data/status/686943722236461056



© AlchemyAPI

CENG 501 Deep Learning Week 5

Sinan Kalkan

Derive the gradients of hinge loss

Previously on CENG501!

$$\frac{\partial L_i}{\partial w_{mk}} = ?$$

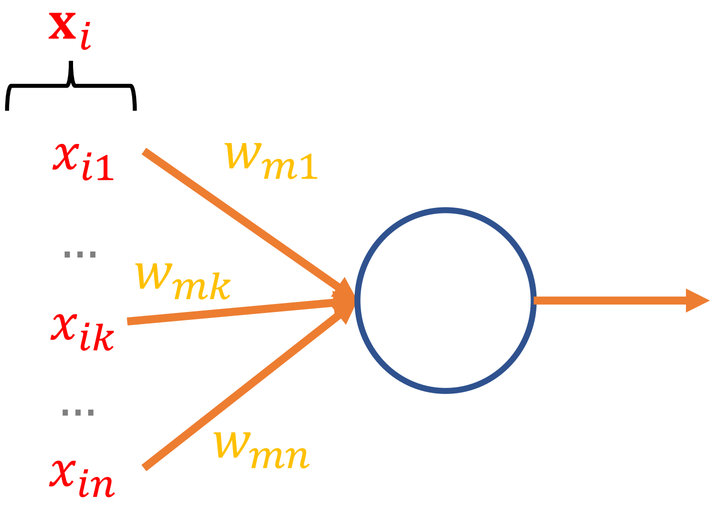
$$\frac{\partial L_i}{\partial w_{mk}} = \frac{\partial L_i}{\partial e_m}\frac{\partial e_m}{\partial s_m}\frac{\partial s_m}{\partial w_{mk}}$$

1

$\mathbb{I}(s_m - s_{y_i} + \Delta > 0)$

x_{ik}

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



$$s_m = \mathbf{w}_m \mathbf{x}_i = \sum_k w_{mk}\, x_{ik}$$

$$L_i = \sum_{j \neq y_i} \underbrace{\max(0, s_j - s_{y_i} + \Delta)}_{e_j}$$

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

Previously on CENG501!

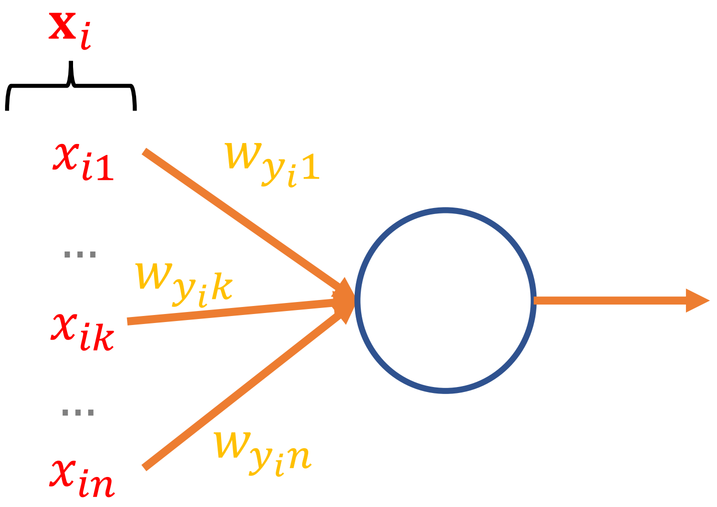
$$\frac{\partial L_i}{\partial w_{y_i k}} = ?$$

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

$$1$$

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

$$x_{ik}$$



$$s_{y_i} = \mathbf{w}_{y_i} \mathbf{x}_i$$

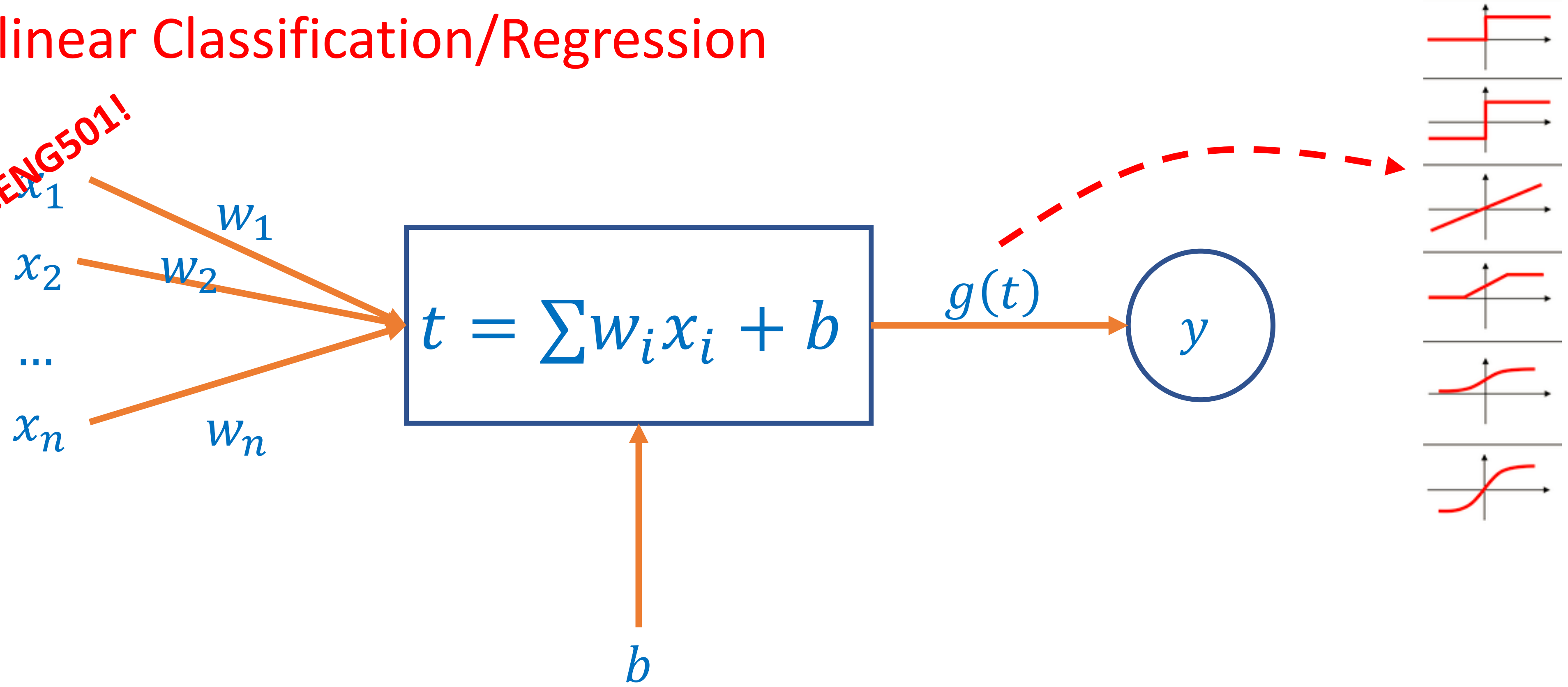
$$= \sum_k w_{y_i k}\, x_{ik}$$

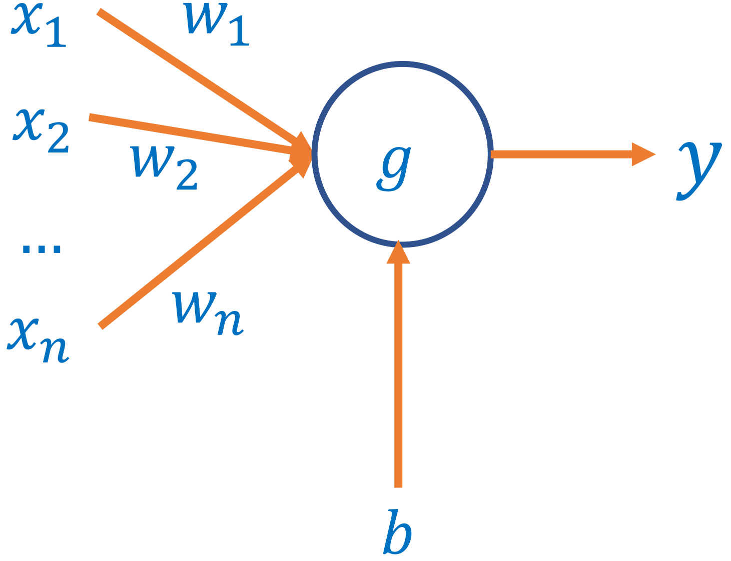
$$L_i = \sum_{j \neq y_i} \underbrace{\max(0, s_j - s_{y_i} + \Delta)}_{e_j}$$

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

Non-linear Classification/Regression

Previously on CENG501!

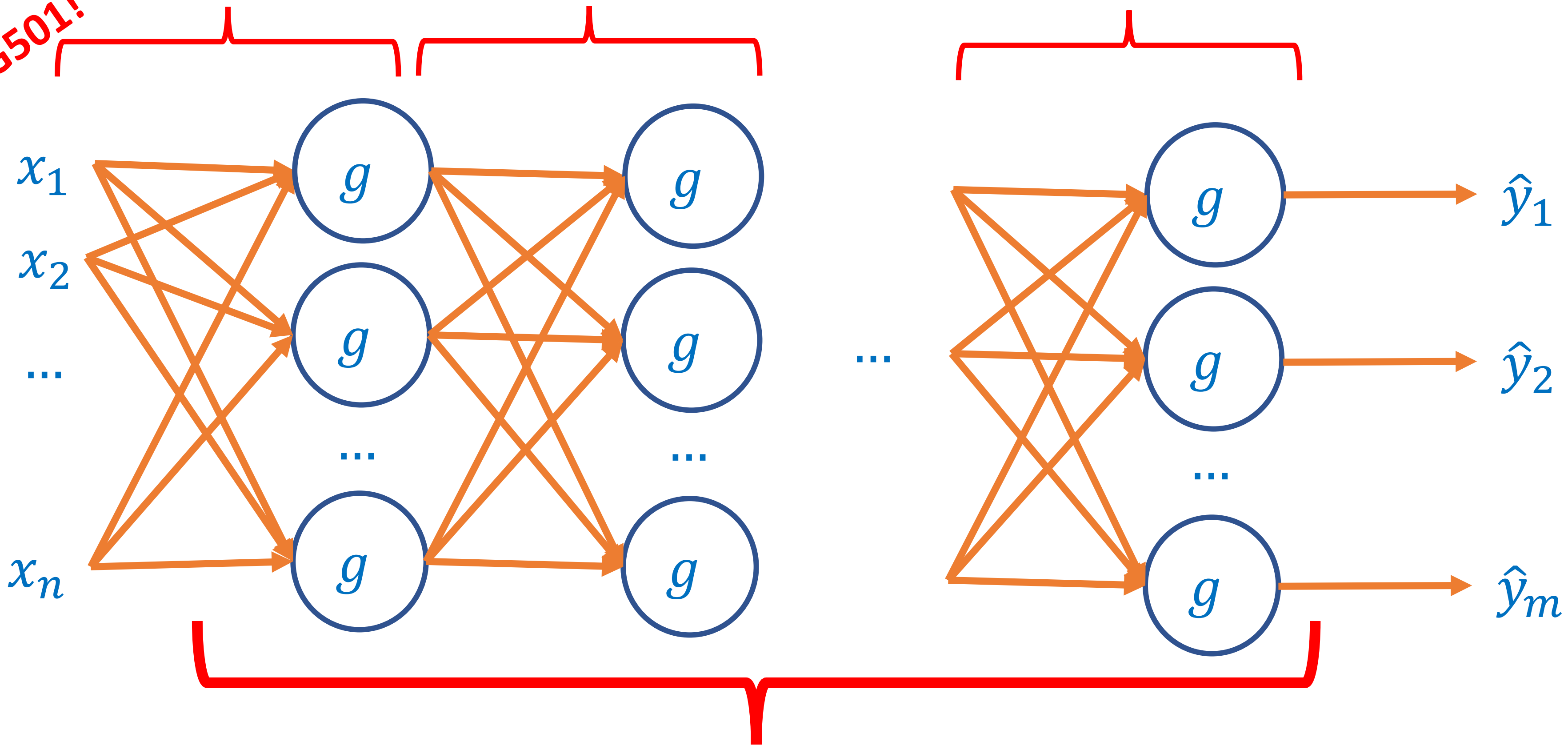




$$\boldsymbol{y} = f(\mathbf{x}) = f(\mathbf{x};\boldsymbol{\theta})$$

$$= g\left(\sum_i w_i x_i + b\right) = g(\mathbf{w}\cdot\mathbf{x})$$

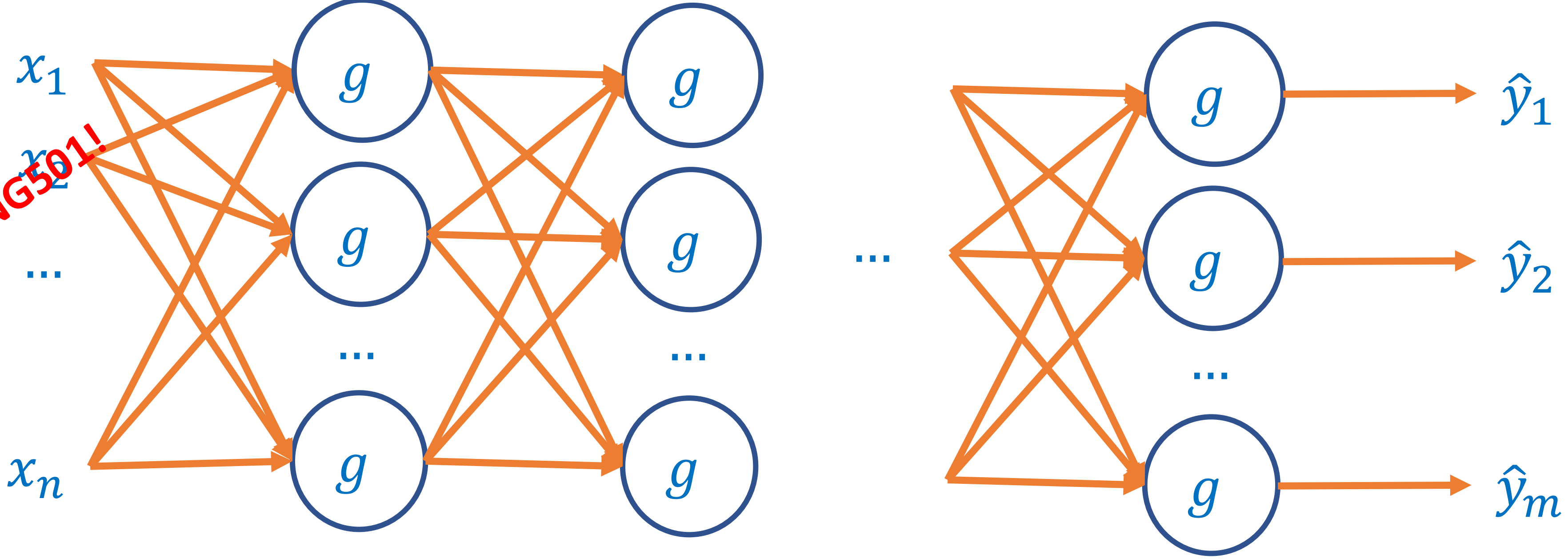
$\mathbf{a}_1 = g(\mathbf{W}_0 \mathbf{x}^T)$ $\mathbf{a}_2 = g(\mathbf{W}_1 \mathbf{a}_1^T)$ $\mathbf{a}_l = g(\mathbf{W}_{l-1} \mathbf{a}_{l-1}^T)$



Previously on CENG501!

$$f() = g\left(g\left(g(g(\dots))\right)\right)$$

Previously on CENG501!



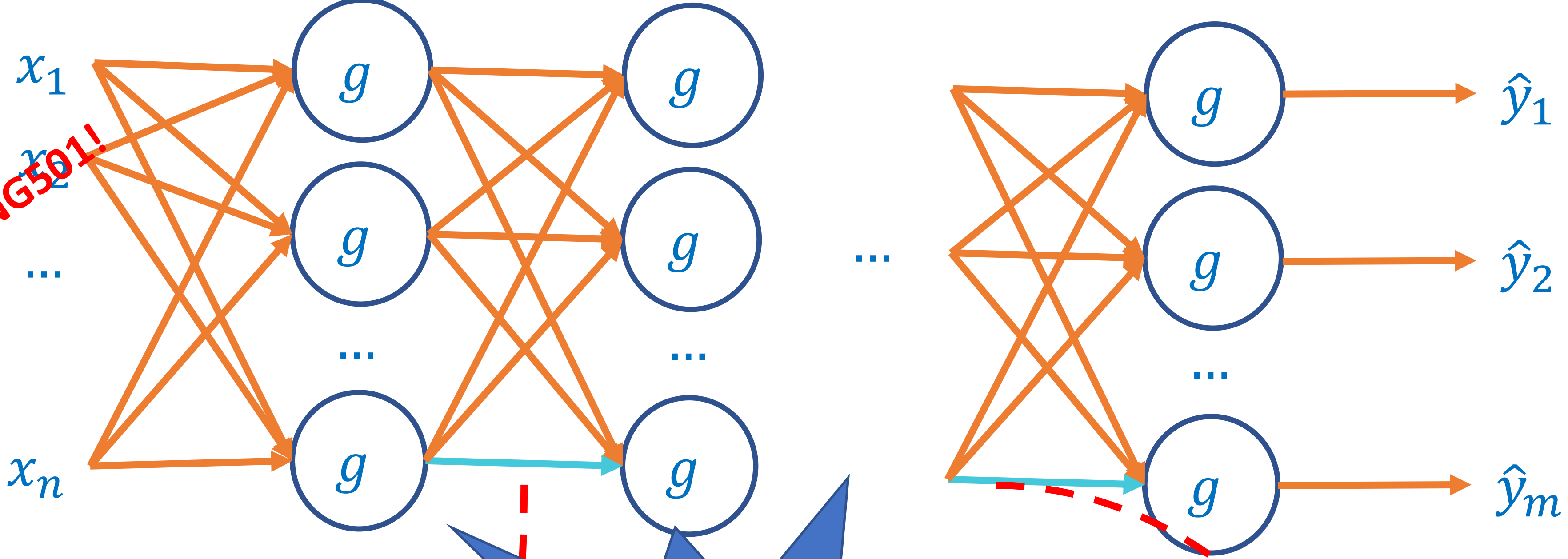
$$L(\mathbf{x};\boldsymbol{\theta}) = \mathrm{d}\big(\mathbf{y}, f(\mathbf{x};\boldsymbol{\theta})\big)$$

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta\, \nabla_{\boldsymbol{\theta}} L(\mathbf{x};\boldsymbol{\theta})$$

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\mathbf{x};\boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{\theta}} L(\mathbf{x};\boldsymbol{\theta}) = \frac{\partial L(\mathbf{x};\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

Previously on CENG501!

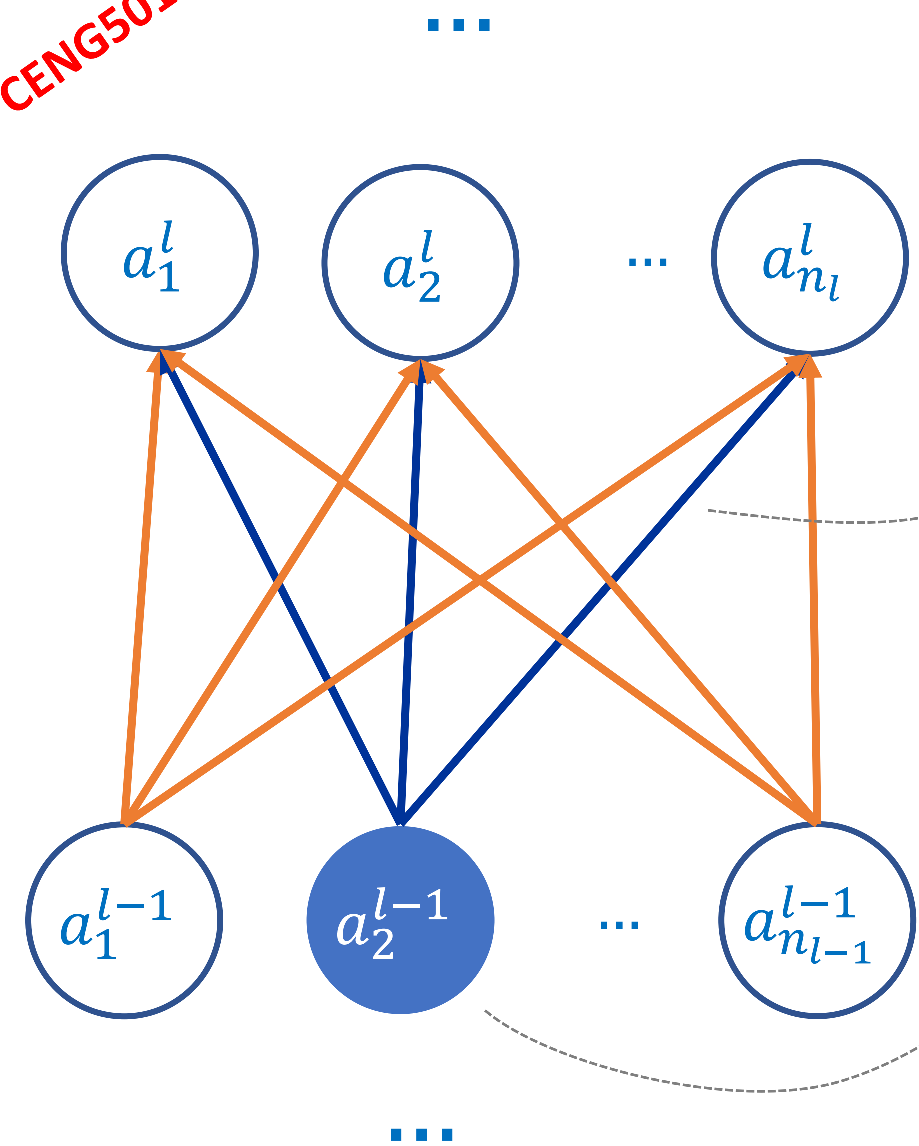


Backpropagation

$$\frac{\partial L(\mathbf{x};\boldsymbol{\theta})}{\partial w_{mi}} = \frac{\partial L}{\partial \hat{y}_m}\frac{\partial \hat{y}_m}{\partial g_m^l}\frac{\partial g_m^l}{\partial t_m}\frac{\partial t_m}{\partial w_{mi}}$$

$$\frac{\partial L(\mathbf{x};\boldsymbol{\theta})}{\partial w_j} = \frac{\partial L}{\partial \hat{y}_m}\frac{\partial \hat{y}_m}{\partial g_m^l}\frac{\partial g_m^l}{\partial t_m}\frac{\partial t_m}{\partial g_k^{l-1}}\frac{\partial g_k^{l-1}}{\partial t_k}\cdots$$

Previously on CENG501!



$$\frac{\partial L(\mathbf{x};\boldsymbol{\theta})}{\partial w_{ij}} = \frac{\partial L(\mathbf{x};\boldsymbol{\theta})}{\partial a_i^l} \frac{\partial a_i^l}{\partial w_{ij}}$$

$$\frac{\partial L(\mathbf{x};\boldsymbol{\theta})}{\partial a_i^{l-1}} = \sum_j \frac{\partial L(\mathbf{x};\boldsymbol{\theta})}{\partial a_j^l} \frac{\partial a_j^l}{\partial a_i^{l-1}}$$

Previously on CENG501!

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.

Cybenko, G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2 (4), 303-314
Kurt Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks", Neural Networks, 4(2), 251257.
Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." Neural networks 2.5 (1989): 359-366.

Multi-layered Fully-connected Networks

Previously on CENG501!

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

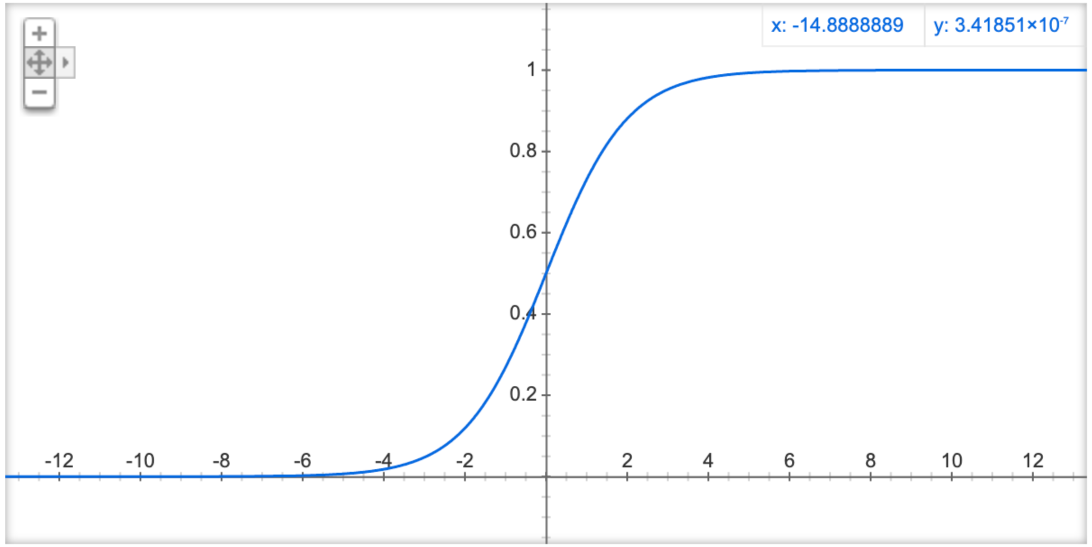
$$\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x})$$

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*

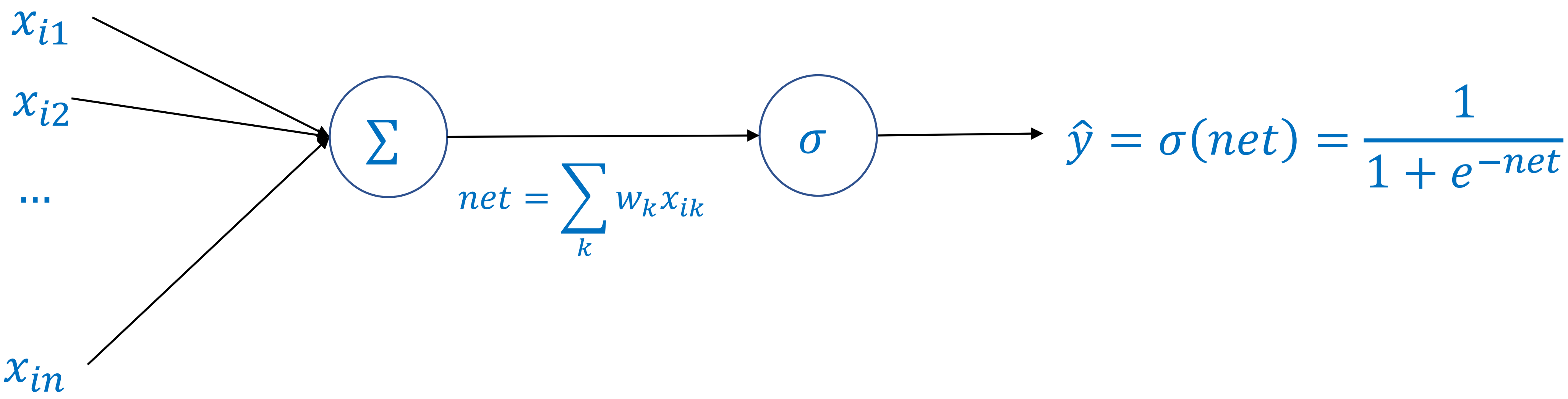
Graph for 1/(1+exp(-x))



More info

A neuron with sigmoid function

Previously on CENG501!



$$net = \sum_k w_k x_{ik}$$

$$\hat{y} = \sigma(net) = \frac{1}{1 + e^{-net}}$$

Backpropagation

Previously on CENG501!

The Model

Hidden activations:

$$h_{ij} = \sigma\left(\mathbf{w}_j^h \cdot \mathbf{x}_i\right) = \sigma\left(net_{ij}^h\right)$$

Output layer:

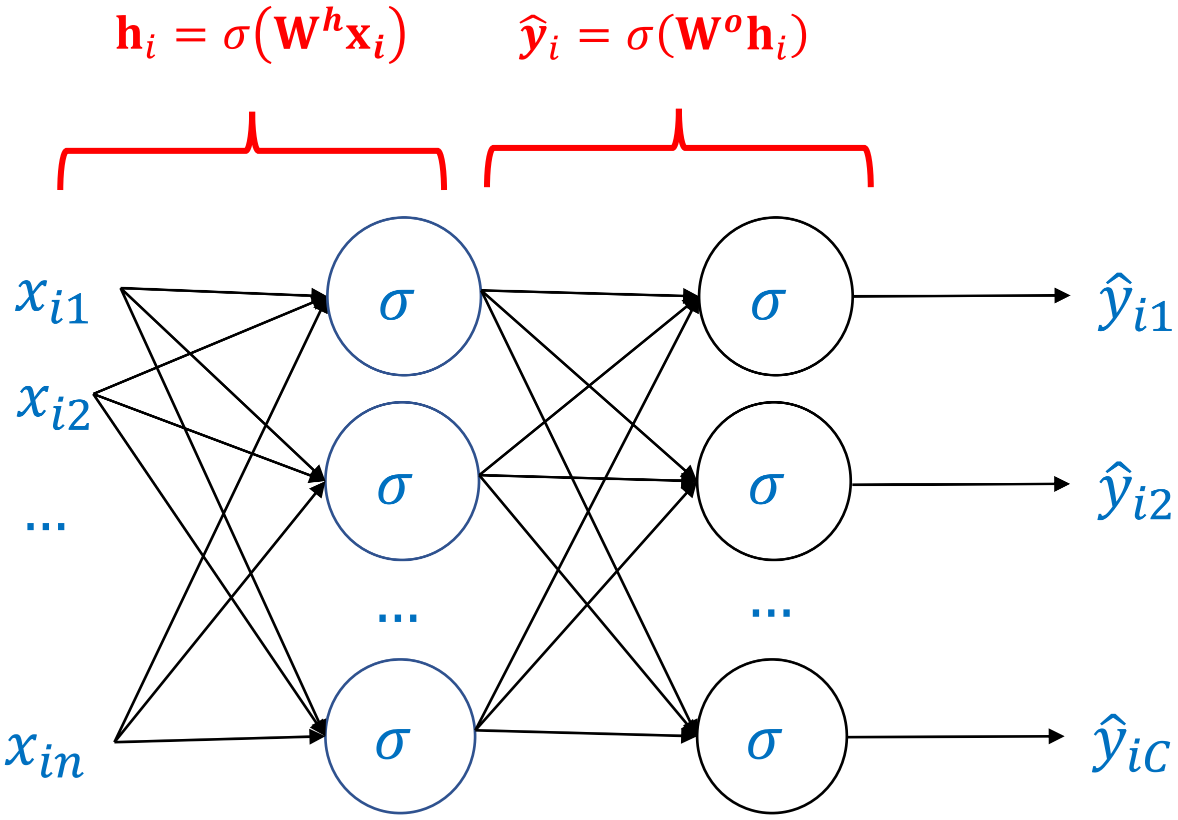
$$\hat{y}_{ic} = \sigma(\mathbf{w}_c^o \cdot \mathbf{h}_i) = \sigma(net_{ic}^o)$$

The loss function:

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

- For one sample:

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



Backpropagation

Previously on CENG501!

For each output unit c, calculate its grad term δ_c^o:

$$\delta_{ic}^o = \frac{\partial L_i}{\partial net_{ic}^o} = \frac{\partial L_i}{\partial \hat{y}_{ic}} \frac{\partial \hat{y}_{ic}}{\partial net_{ic}^o} = (\hat{y}_{ic} - y_{ic})\hat{y}_{ic}(1 - \hat{y}_{ic})$$

For each hidden unit j, calculate its grad term δ_j^h:

$$\delta_{ij}^h = \frac{\partial L_i}{\partial net_{ij}^h} = \frac{\partial L_i}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial net_{ij}^h} = \left(\sum_{c \in C} \frac{\partial L_i}{\partial net_{ic}^o} \frac{\partial net_{ic}^o}{\partial h_{ij}}\right) h_{ij}(1 - h_{ij})$$

$$= \left(\sum_{c \in C} \delta_{ic}^o w_{cj}\right) h_{ij}(1 - h_{ij})$$

Update weight w_{jk}^o in the output layer:

$$w_{jk}^o = w_{jk}^o - \eta \delta_{ij}^o h_{ik}$$

Update weight w_{jk}^h in the hidden layer:

$$w_{jk}^h = w_{jk}^h - \eta \delta_{ij}^h x_{ik}$$

The Model

Hidden activations: $h_{ij} = \sigma(\mathbf{w}_j^h \cdot \mathbf{x}_i) = \sigma(net_{ij}^h)$

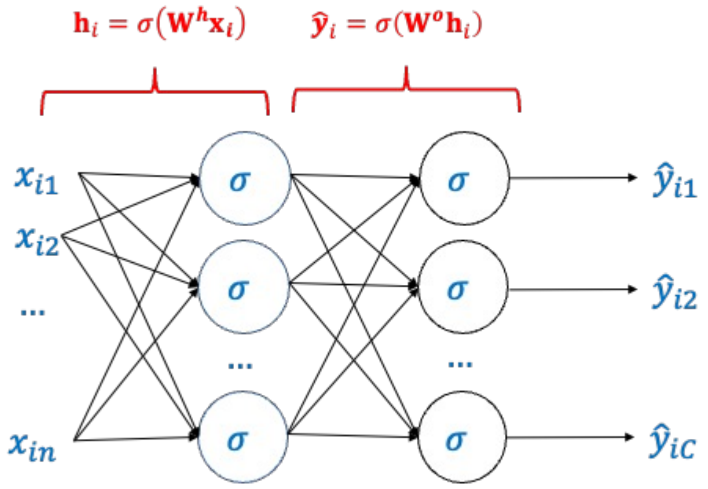
Output layer: $\hat{y}_{ic} = \sigma(\mathbf{w}_c^o \cdot \mathbf{h}_i) = \sigma(net_{ic}^o)$

The loss function:

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

- For one sample:

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


$\mathbf{h}_i = \sigma(\mathbf{W}^h \mathbf{x}_i)$
$\hat{\mathbf{y}}_i = \sigma(\mathbf{W}^o \mathbf{h}_i)$
x_{i1}
x_{i2}
...
x_{in}
σ
...
$\hat{y}_{i1}$
$\hat{y}_{i2}$
$\hat{y}_{iC}$

Backpropagation vs. numerical differentiation

Previously on CENG501!

What are their complexities?

- Backpropagation:
  - $O(|\theta|)$

- Numerical differentiation
  - $O(|\theta|^2)$

Neural Engineering

Previously on CENG501!

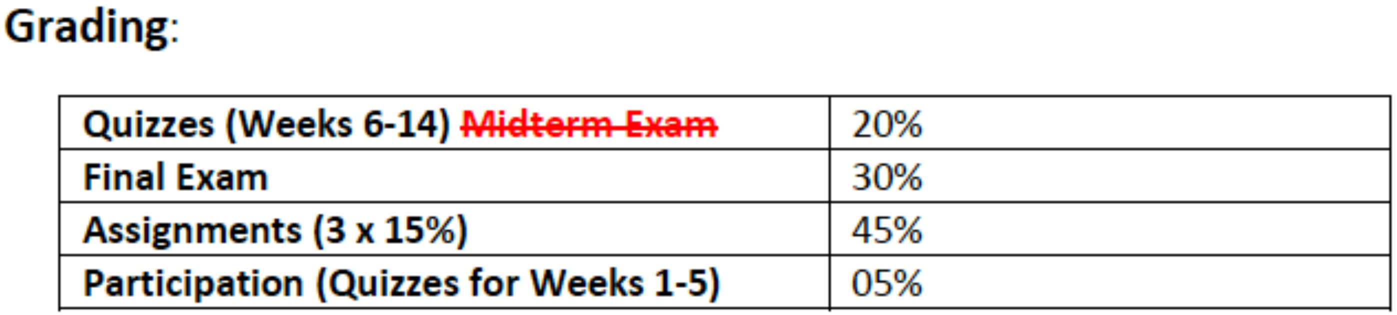
- Loss functions
- On optimization
- Activation functions
- Capacity, convergence
- Preprocessing
- …

Today

- More on loss functions
- Activation functions

Administrative issues

- Lectures:
  - Face-to-face vs. online?
- Midterm exam:
  - “Merged” with final exam
- Programming Assignment 1 (PA1):
  - Due: 16 April.



Grading:

Quizzes (Weeks 6-14) ~~Midterm Exam~~	20%
Final Exam	30%
Assignments (3 x 15%)	45%
Participation (Quizzes for Weeks 1-5)	05%

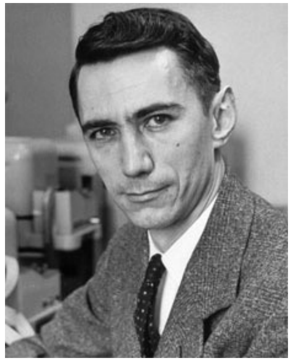
Quiz 3

- On ODTUclass.
- Finish by 23:59 today.

More on loss functions

Softmax or Logistic CLASSIFIERS

Information Entropy



Claude Elwood Shannon
(1916-2001)

("A Mathematical Theory of Communication", 1948)

- Number of bits to represent a coin-pair:

$$\log_2 4 = 2$$

- In fact, this is:

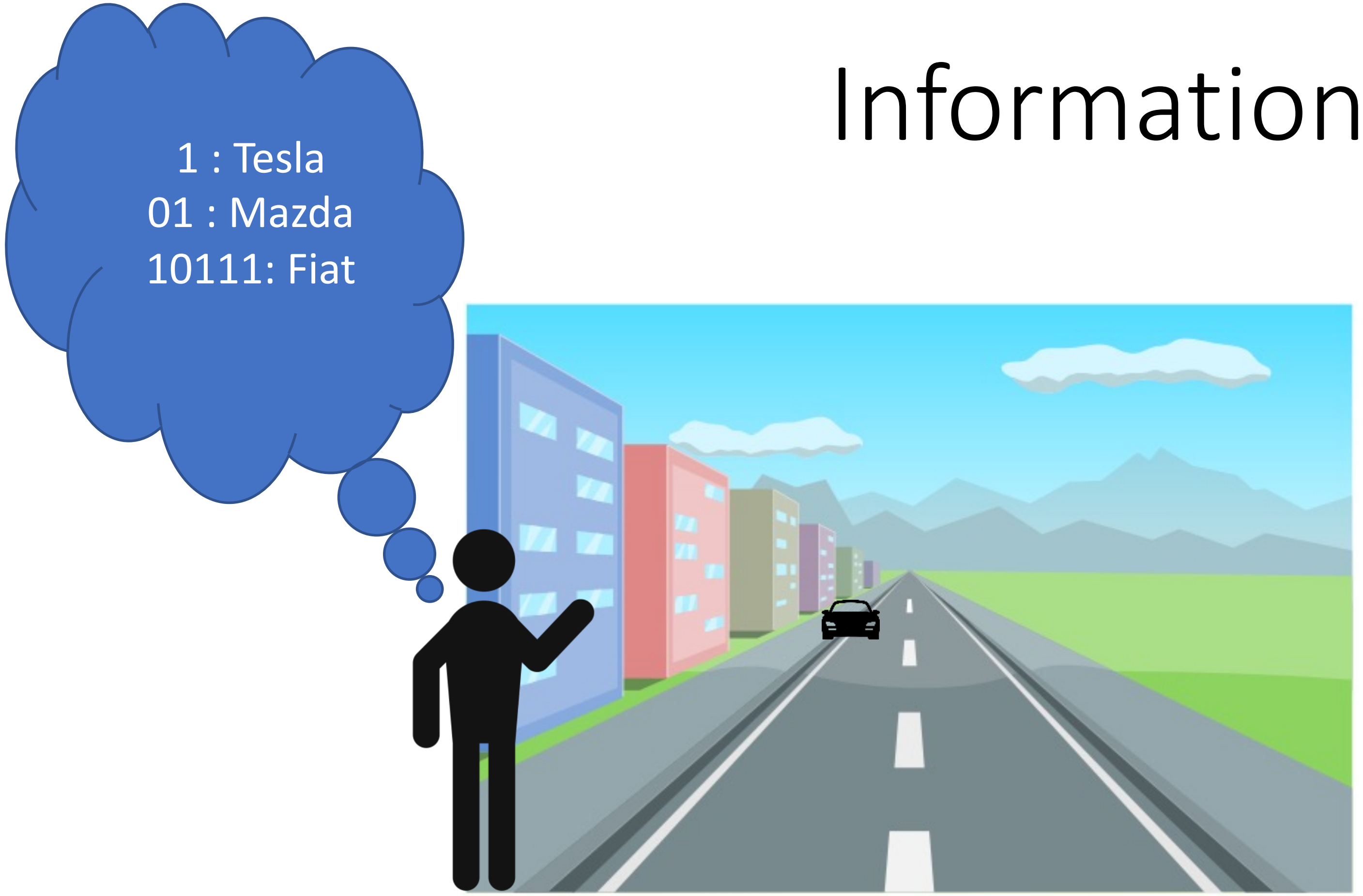
$$\log_2 \frac{1}{p_{coin}} = \log_2 \frac{1}{0.25} = 2$$

- Optimal number of bits to represent an event with probability p:

$$\log_2 \frac{1}{p}$$

H	H
H	T
T	H
T	T

Information Entropy



1 : Tesla
01 : Mazda
10111: Fiat

- Problem:
  - Transmit the label of cars to another person with least number of bits
- Assume that each bit is expensive
  - So, we are interested in the minimal/optimal coding

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

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: $\log_2 1/p$

- Optimal bits to represent fiat cars:

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

- The optimal encoding then requires:

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

Car	Probability	# of bits
Fiat	0.80	0.32
Mazda	0.15	2.74
Tesla	0.05	4.32

Entropy

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:

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

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

Cross-Entropy

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

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

$$L_i = -\log_e\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) = -s_{y_i} + \log_e \sum_j e^{s_j}$$

http://cs231n.github.io/

Derive the gradients of NLL loss

$$\frac{\partial L_i}{\partial w_{jk}} = ?$$

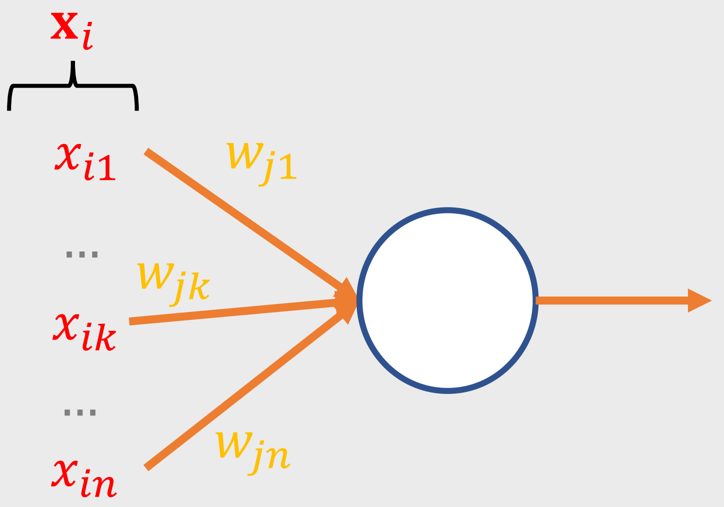
If $j = y_i$:

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

$$-\frac{1}{p_{y_i}} \qquad p_{y_i}(1-p_{y_i}) \qquad x_{ik}$$

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

THE MODEL



$$s_j = \mathbf{w}_j \mathbf{x}_i = \sum_k w_{jk}\, x_{ik}$$

$$L_i = -\log_e p_{y_i}$$

$$p_{y_i} = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}}$$

This assumed that $j = y_i$.
What happens if that's not the case?
See the next page.

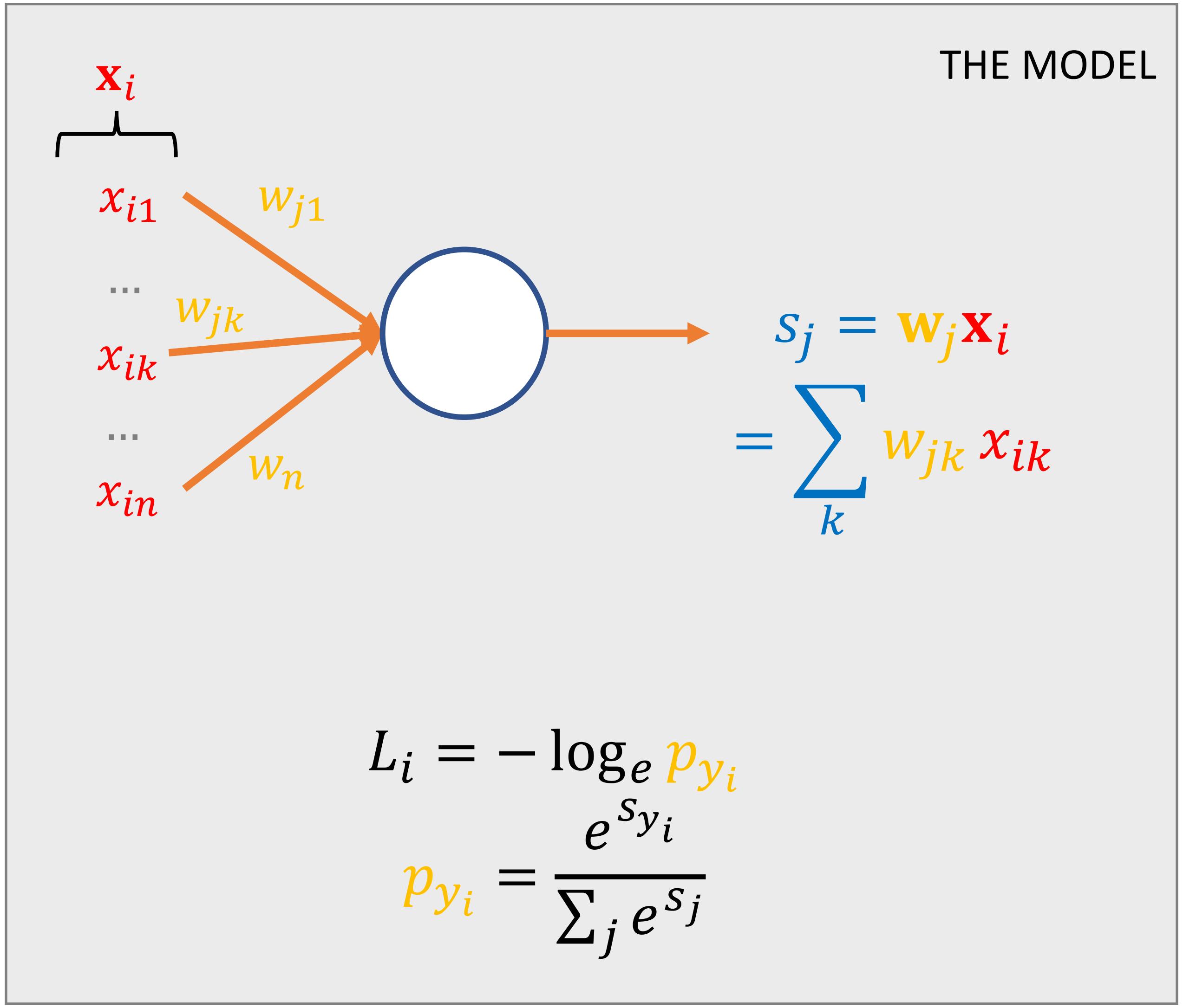
Derive the gradients of NLL loss

$$\frac{\partial L_i}{\partial w_{jk}} = ?$$

If $j \neq y_i$:

$$\frac{\partial L_i}{\partial w_{jk}} = \underbrace{\frac{\partial L_i}{\partial p_{y_i}}}_{-\frac{1}{p_{y_i}}} \underbrace{\frac{\partial p_{y_i}}{\partial s_j}}_{-p_{y_i} p_j} \underbrace{\frac{\partial s_j}{\partial w_{jk}}}_{x_{ik}}$$

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



$$s_j = \mathbf{w}_j \mathbf{x}_i = \sum_k w_{jk} x_{ik}$$

$$L_i = -\log_e p_{y_i}$$

$$p_{y_i} = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}}$$

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):

$$\frac{1}{1+e^{-x}}$$

Logistic regression!
(with a linear model)

[1] http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/

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, $p = [0, \dots, 1{,}0, .. 0]$, 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.

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\,\log\frac{P(i)}{Q(i)}.$$

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/

Hinge loss vs. cross-entropy loss

- Hinge loss is “happy” (= zero) when the classification satisfies the margin
  - Ex: if score values = [10, 9, 9] or [10, -10, -10]
    - Hinge 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:

$$\arg\max\ (s_1, s_2, \ldots, s_n) = (y_1, y_2, \ldots, y_n) = (0,0,..,0,1,0..0)$$

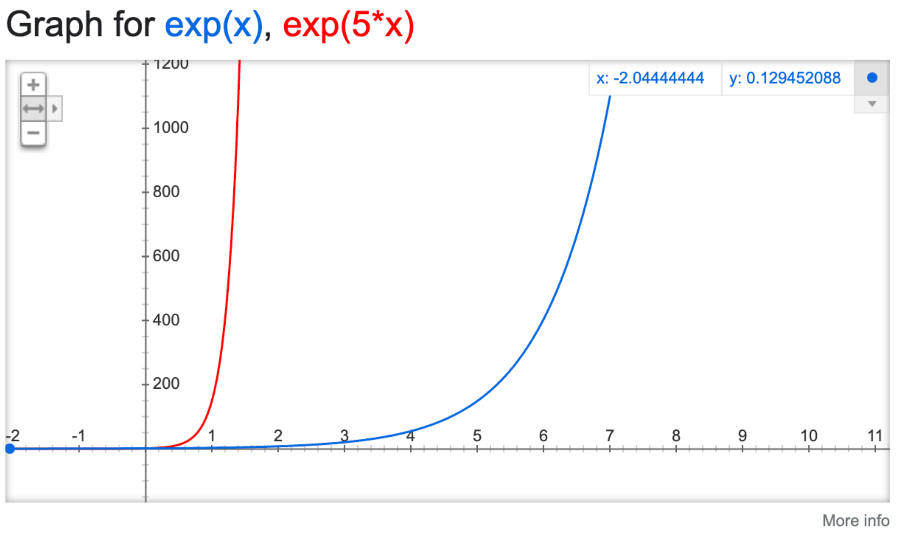
- The base in softmax can be changed to have more “peaky” (or distributed) values for the largest input ($\mathrm{e}^{\beta} = b$):

$$sm_{\beta}(s_i) = \frac{e^{\beta s_i}}{\sum_j e^{\beta s_j}}$$

- When $\beta \rightarrow \infty$, softmax converges to arg max.

e.g.

$$sm_{\beta=1}([1, 1.1]) \quad = [0.475\ 0.524]$$
$$sm_{\beta=2}([1, 1.1]) \quad = [0.451\ 0.550]$$
$$sm_{\beta=5}([1, 1.1]) \quad = [0.378\ 0.622]$$
$$sm_{\beta=100}([1, 1.1]) = [4.5\mathrm{e}{-}05\ 9.9\mathrm{e}{-}01]$$



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 β => 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:
    - $L_i = \sum_{j \neq y_i} \max\left(0, f_j - f_{y_i} + 1\right)$
  - 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:
    - $L_i = \sum_j \max\left(0, 1 - y_{ij} f_j\right)$
    - $y_{ij} = +1$ if example i is labeled with label j; otherwise $y_{ij} = -1.$
  - Alternatively, train logistic loss for each label (0 or 1):

$$L_i = \sum_j y_{ij} \log(\sigma(f_j)) + (1 - y_{ij}) \log(1 - \sigma(f_j))$$

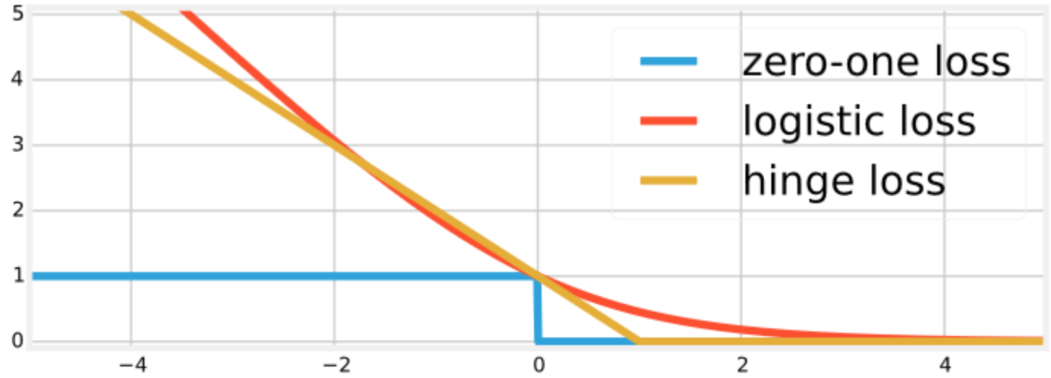
0-1 Loss

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

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

Or equivalently,

$$L = \sum_{\boldsymbol{i}} \mathbf{1}\big(y_i f(x_i; W, b)_{y_i} < 0\big)$$



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.

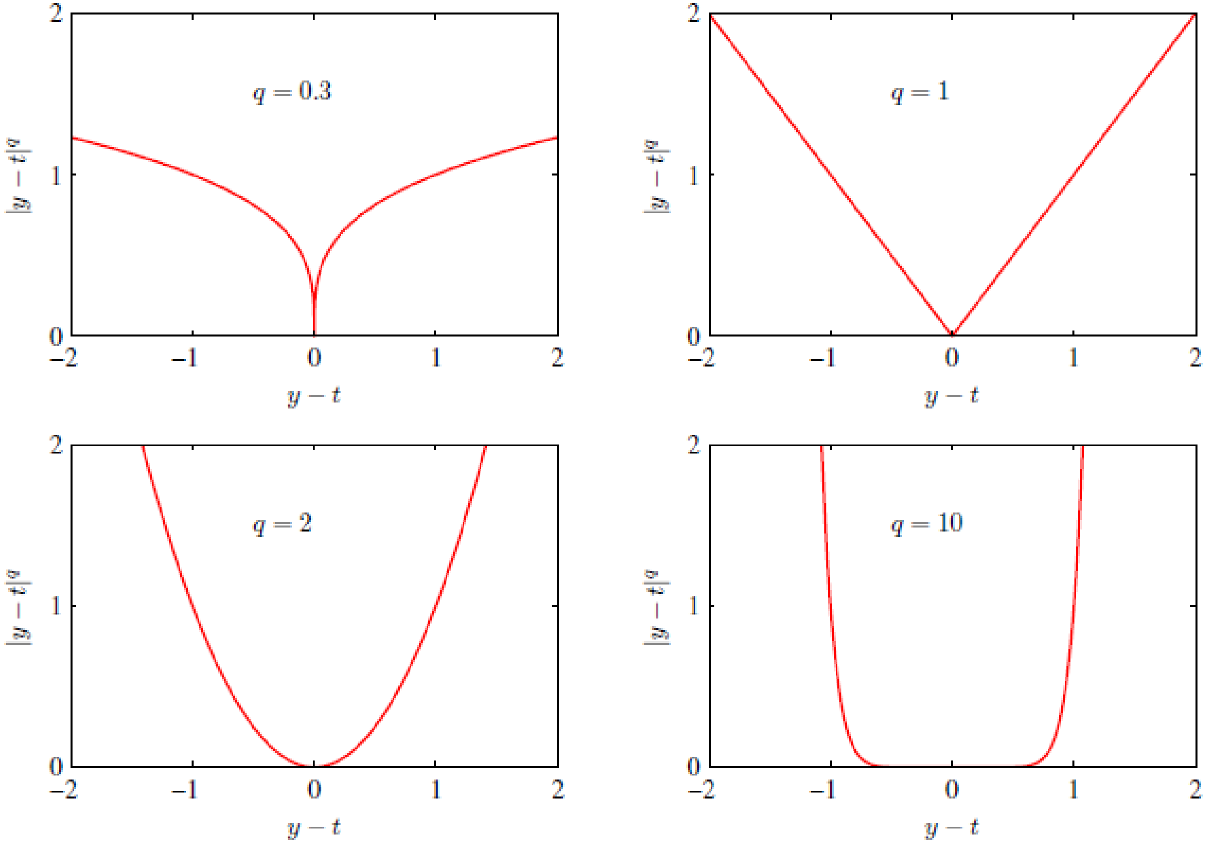


Figure 1.29 Plots of the quantity $L_q = |y - t|^q$ for various values of q.

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

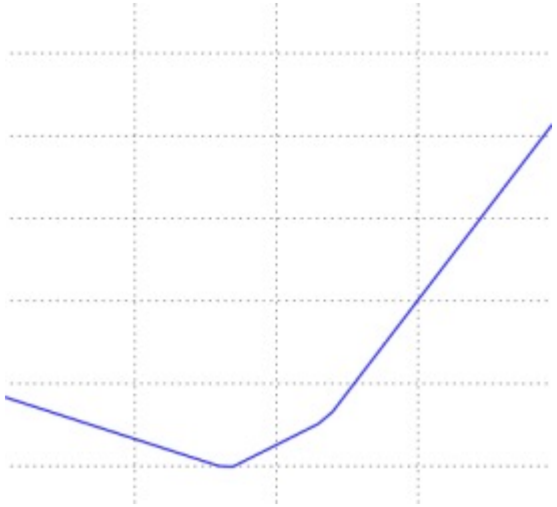
- If you look at one of the example loss functions:

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

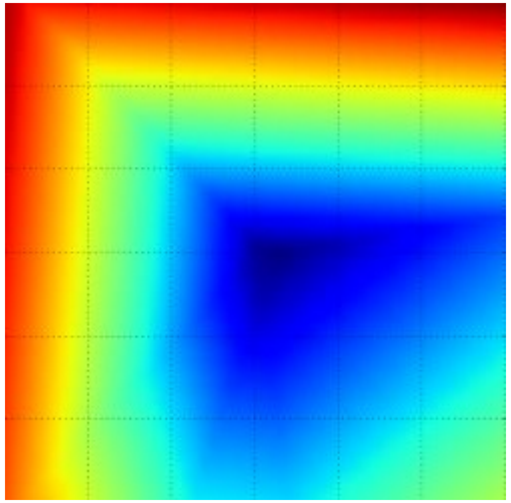
- Since W has too many dimensions, this is difficult to plot.
- We can visualize this for one weight direction though, which can give us some intuition about the shape of the function.
  - E.g., start from an arbitrary W_0, choose a direction W_1 and plot $L(W_0 + \alpha W_1)$ for different values of α.

http://cs231n.github.io/

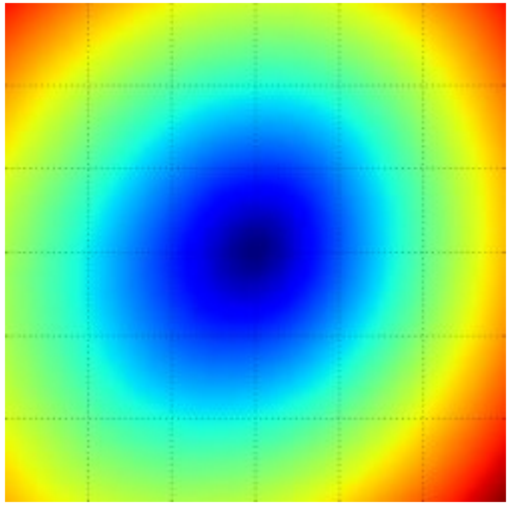
Visualizing Loss Functions



Loss along one direction



Loss along two directions



Loss along two directions (averaged over many samples)

- 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:

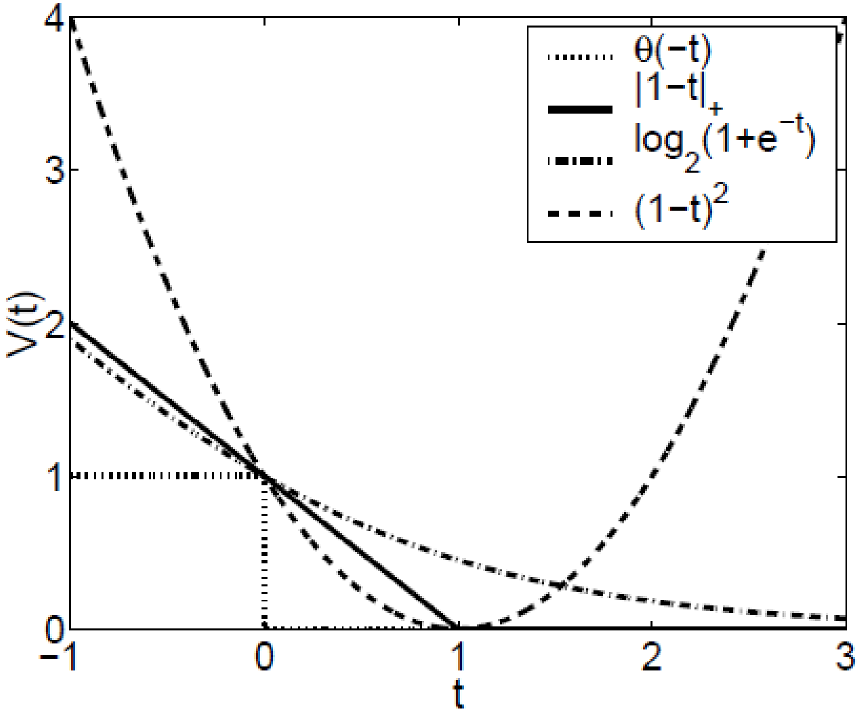
$$L = \left(1 - yf(x)\right)^2$$

- Hinge-loss

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

- Logistic loss (binary Cross Entropy Loss):

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

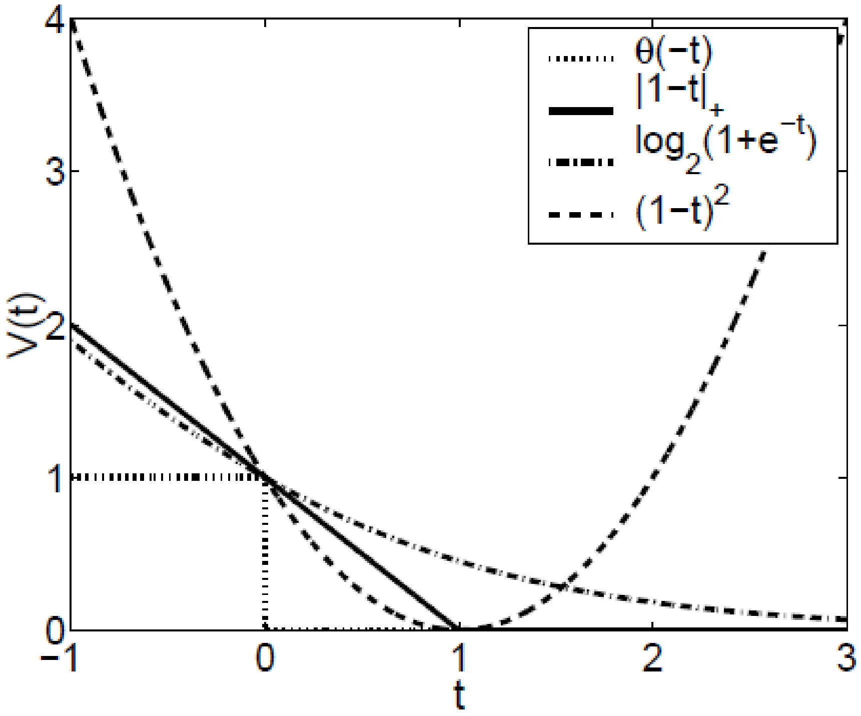


Various loss functions used in classification. Here $t = yf(\mathrm{x})$.

Rosacco et al., 2003

https://web.mit.edu/lrosasco/www/publications/loss.pdf

All losses approximate 0-1 loss



Various loss functions used in classification. Here $t = yf(\mathbf{x})$.

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

$$\frac{d\sigma(x)}{dx} = \sigma(x) \cdot \left(1 - \sigma(x)\right)$$

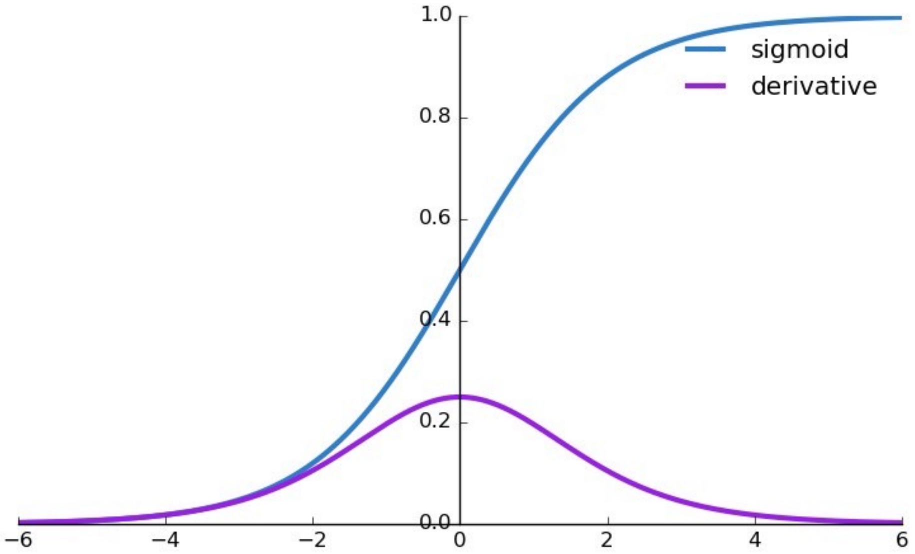


Fig: https://medium.com/@omkar.nallagoni/activation-functions-with-derivative-and-python-code-sigmoid-vs-tanh-vs-relu-44d23915c1f4

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

$$\frac{\mathrm{dtanh}(x)}{dx} = (1 - \tanh^2(x))$$

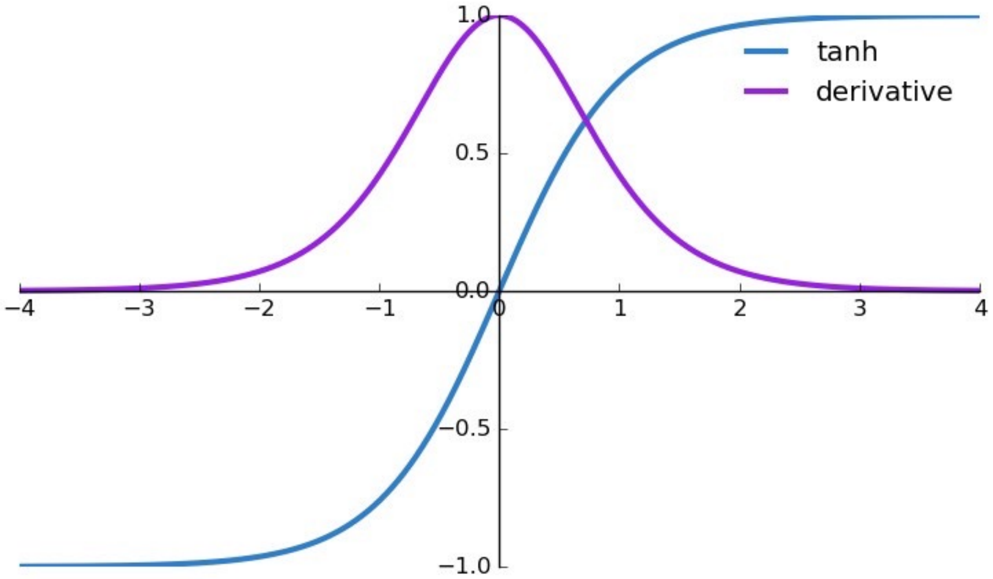


Fig: https://medium.com/@omkar.nallagoni/activation-functions-with-derivative-and-python-code-sigmoid-vs-tanh-vs-relu-44d23915c1f4

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)

They are both non-convex!

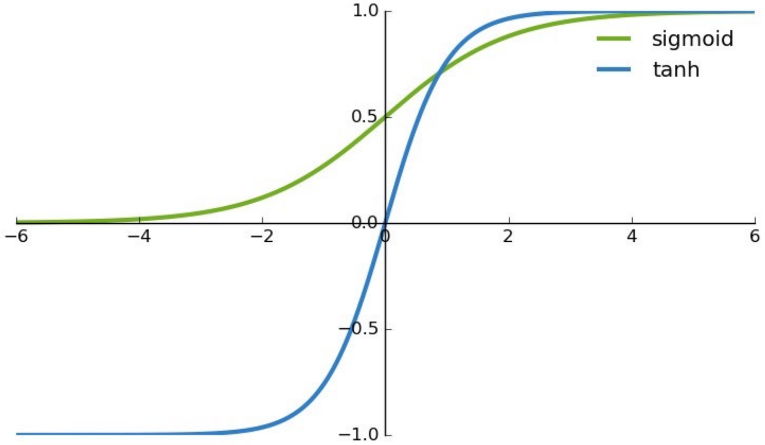


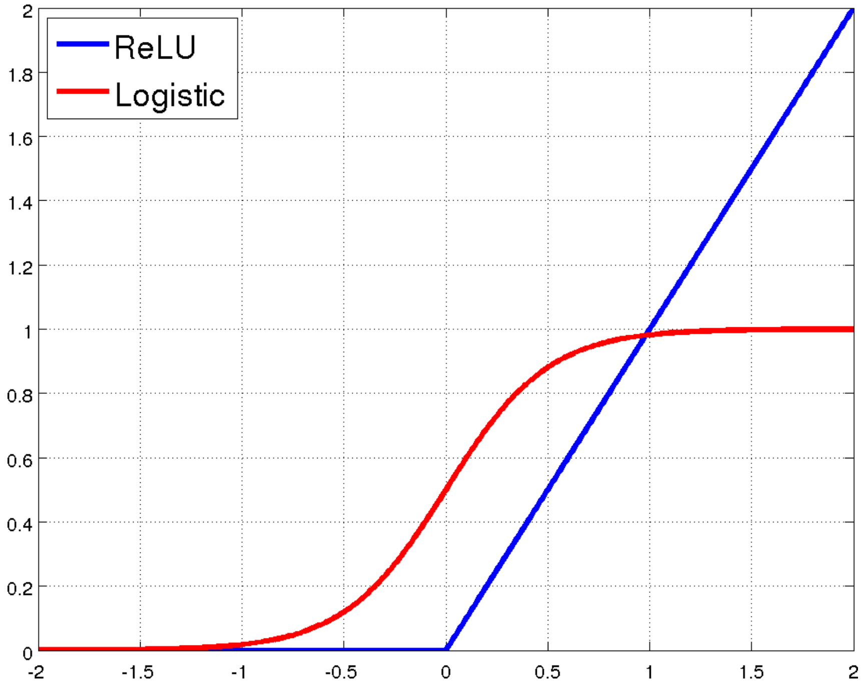
Fig: https://medium.com/@omkar.nallagoni/activation-functions-with-derivative-and-python-code-sigmoid-vs-tanh-vs-relu-44d23915c1f4

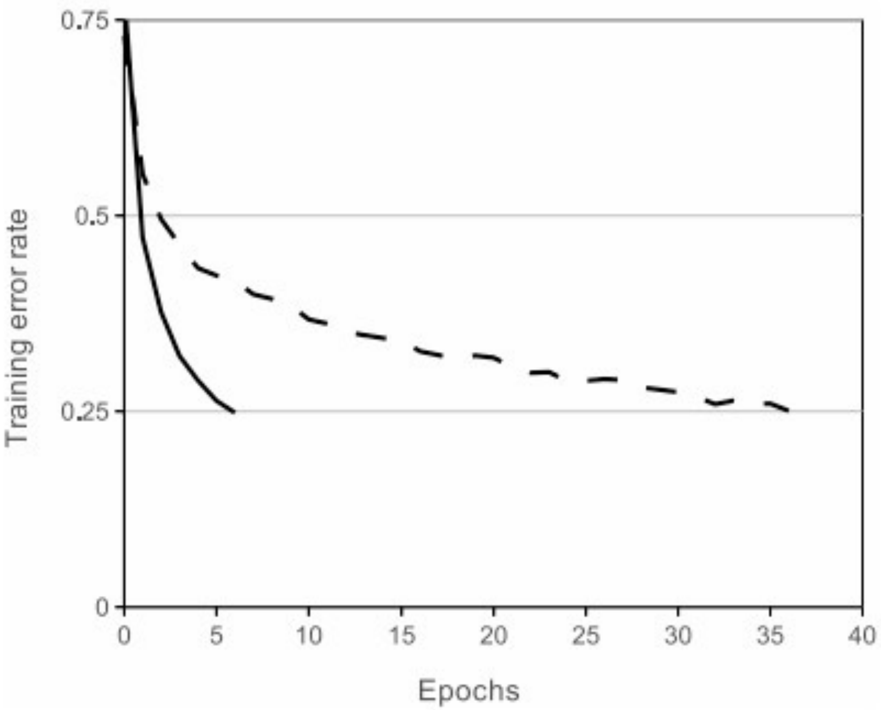
- Tanh
  - Similar to the sigmoid, it saturates
  - However, it is zero-centered.
  - Tanh is generally preferred over sigmod
  - Note: $\tanh(x) = 2\sigma(2x) - 1$

http://cs231n.github.io/neural-networks-1/

Activation Functions: Rectified Linear Units (ReLU)

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





[Krizhevsky et al., NIPS12]

$$f(x) = \max(0, x)$$

Derivative: $\mathbf{1}(x > 0)$

Activation Functions:
ReLU – biological motivation

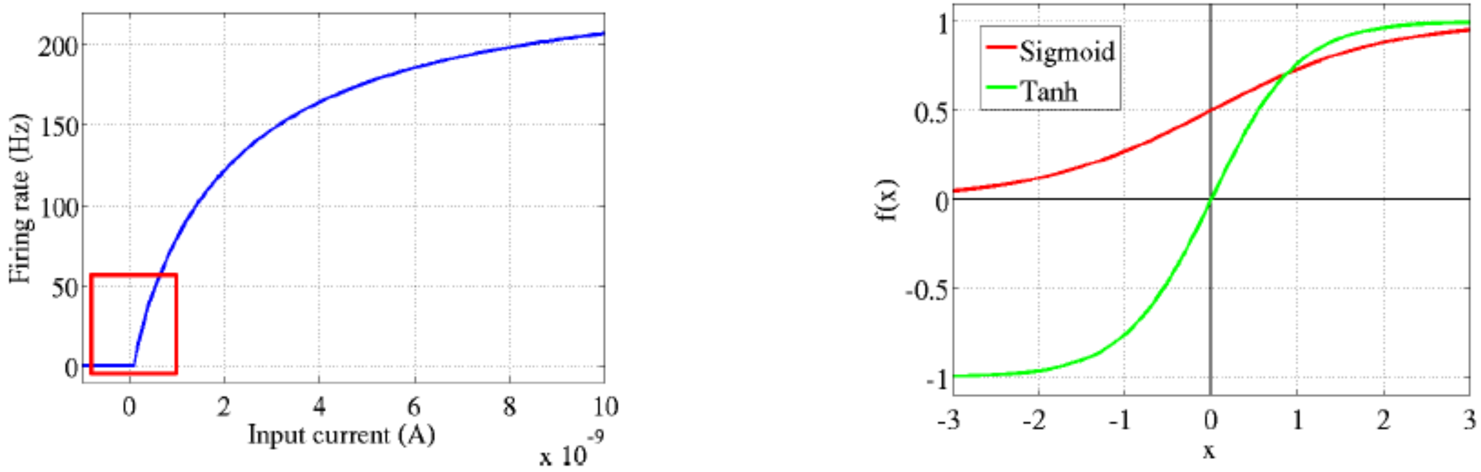
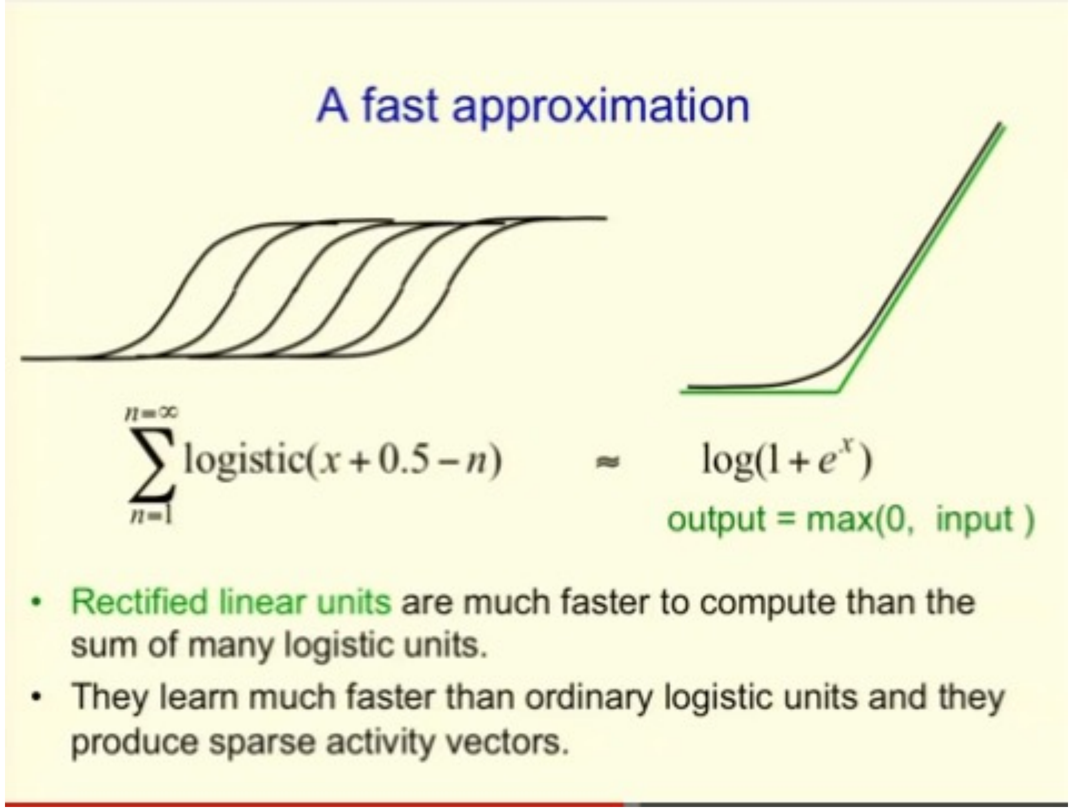


Figure 1: *Left:* **Common neural activation function motivated by biological data.** *Right:* **Commonly used activation functions** in neural networks literature: logistic sigmoid and hyperbolic tangent (*tanh*).

Glorot et al., “Deep Sparse Rectifier Neural Networks”, 2011.

Activation Functions:
ReLU – biological motivation

Hinton argues that this is a form of model averaging



A fast approximation

$$\sum_{n=1}^{n=\infty} \text{logistic}(x+0.5-n) \approx \log(1+e^{x})$$

output = max(0, input)

- 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 as it performs a simple comparison with zero

- 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

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

- $f(x) = \mathbf{1}(x < 0)(\alpha x) + \mathbf{1}(x \geq 0)(x)$
  - When x is negative, Leaky ReLU has a non-zero slope (α)

- If you learn α during training, this is called parametric ReLU (PReLU)

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

Activation Functions: maxout

"Maxout Networks" by Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, Yoshua Bengio, 2013.

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

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

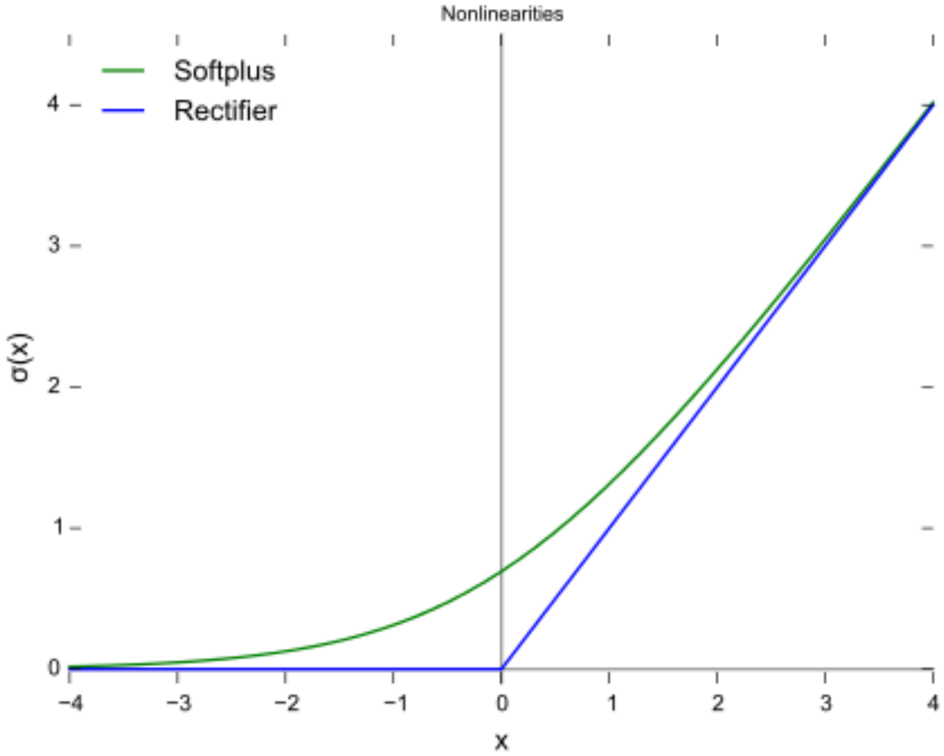
Activation Functions: Softplus

- A smooth approximation to the ReLU unit:

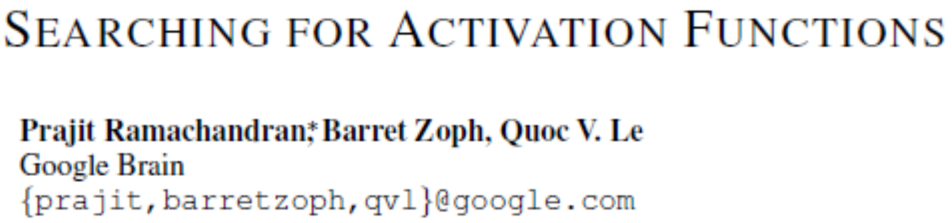
$$f(x) = \ln(1 + e^x)$$

- Its derivative is the sigmoid function:

$$f'(x) = 1/(1 + e^{-x})$$

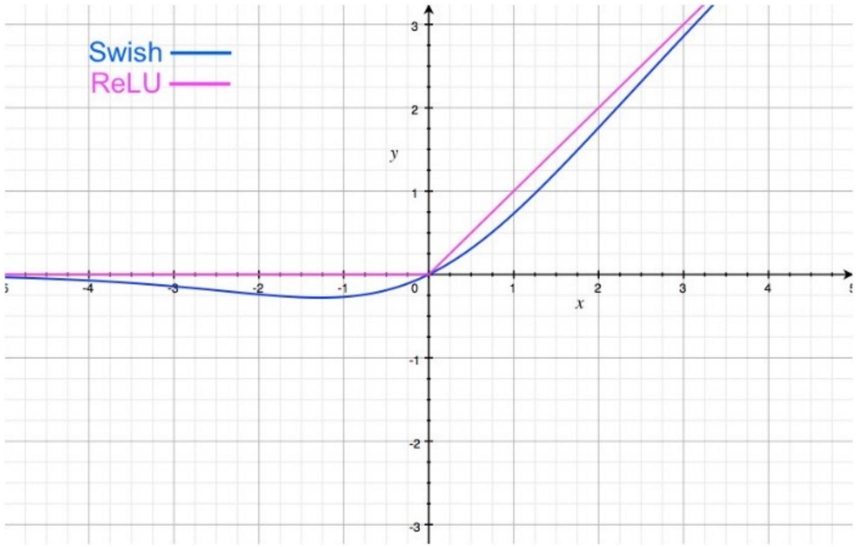


Activation Functions:
Swish: A Self-Gated Activation Function



SEARCHING FOR ACTIVATION FUNCTIONS

Prajit Ramachandran*, Barret Zoph, Quoc V. Le
Google Brain
{prajit,barretzoph,qvl}@google.com



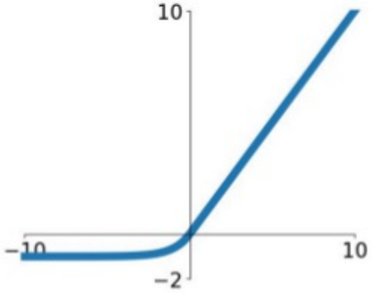
“The choice of activation functions in deep networks has a significant effect on the training dynamics and task performance. Currently, the most successful and widely-used activation function is the Rectified Linear Unit (ReLU). Although various alternatives to ReLU have been proposed, none have managed to replace it due to inconsistent gains. In this work, we propose a new activation function, named Swish, which is simply f(x)=x·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

ELU

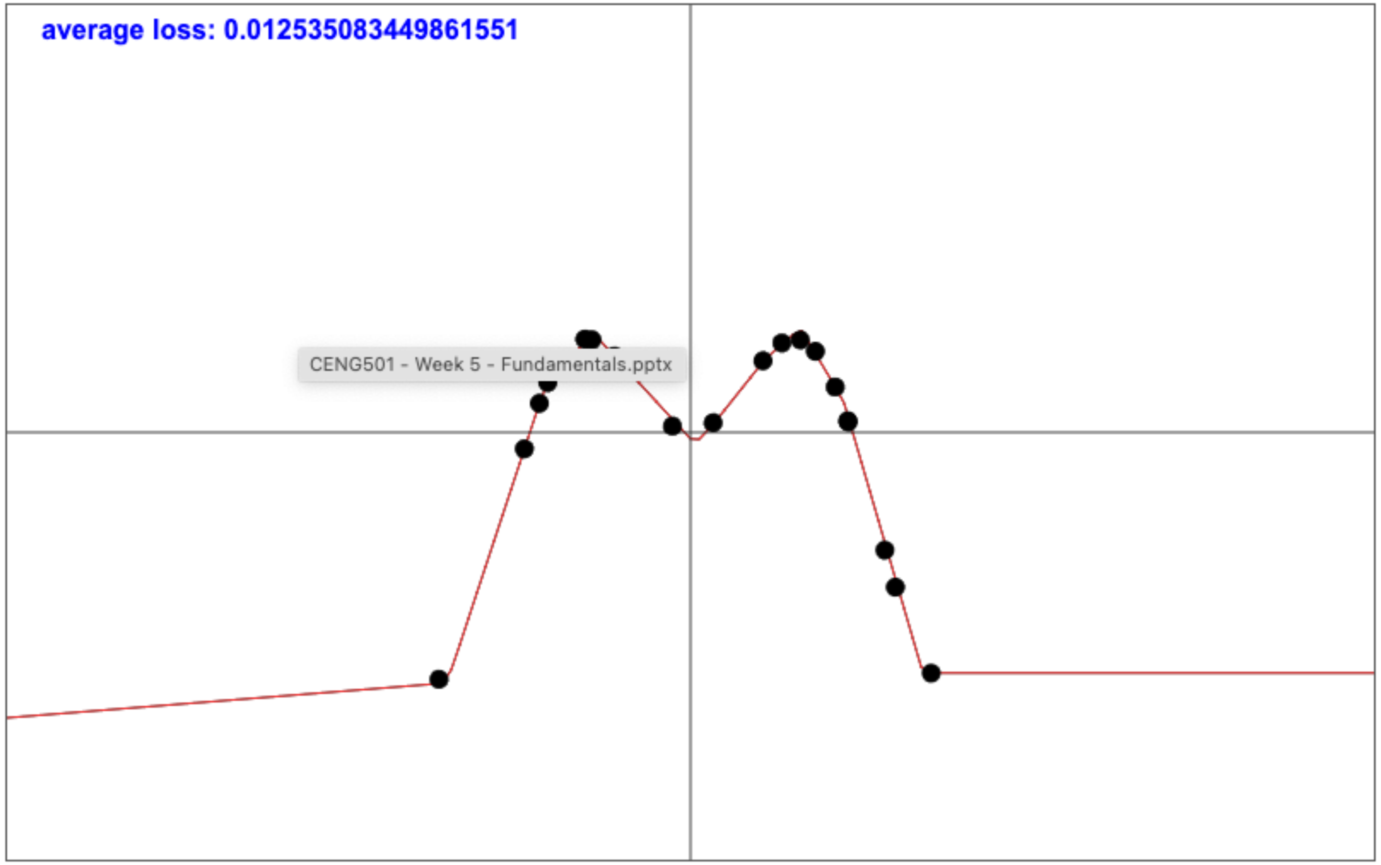
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



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

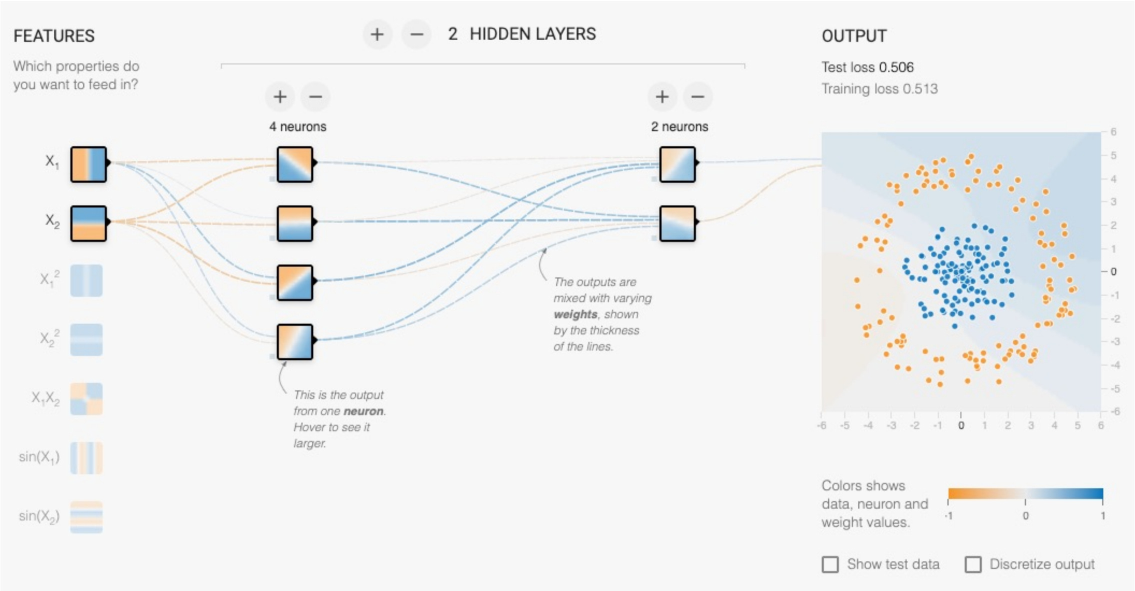
DEMO 1



average loss: 0.012535083449861551
CENG501 - Week 5 - Fundamentals.pptx

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

DEMO 2



Test loss 0.506
Training loss 0.513

The outputs are mixed with varying **weights**, shown by the thickness of the lines.

This is the output from one **neuron**. Hover to see it larger.

Colors shows data, neuron and weight values.

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

https://jalammar.github.io/visual-interactive-guide-basics-neural-networks/