CMPS 460 – Spring 2022

MACHINE LEARNING

Tamer Elsayed

9.b

Neural Networks II: Training

Handouts

Roadmap …

- How to train neural networks?
- What are commonly-used loss functions?
- Regularization
- Gradient Descent & variants
- What are commonly-used activation functions?
- Weight initialization
- Demo with Playground!

How to train neural networks?

Cross-Entropy Loss (Binary classif.)

Recall:

- $L_i(\hat{y}_i, y_i) = -y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i)$

- $L_i(\hat{y}_i, y_i) = \begin{cases} -\log \hat{y}_i & y_i = 1 \\ -\log(1 - \hat{y}_i) & y_i = 0 \end{cases}$

- $L_i(\hat{Y}_i, y_i) = \begin{cases} -\log \hat{y}_1 & y_i = 1 \\ -\log \hat{y}_0 & y_i = 0 \end{cases}$ $\quad \hat{Y}_i = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_0 \end{bmatrix}$

Cross-Entropy Loss (MC classif.)

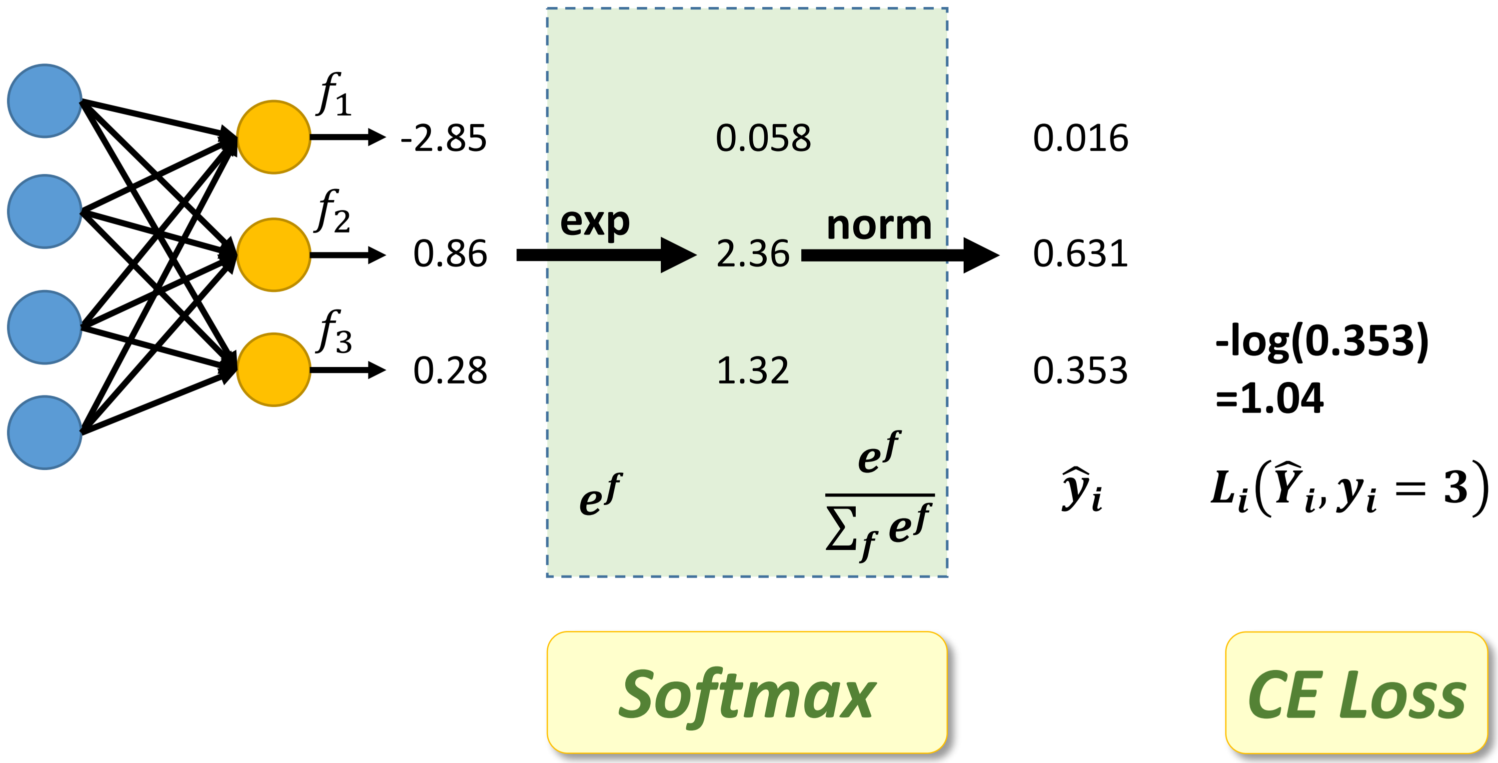
- $L_i(\hat{Y}_i, y_i) = \begin{cases} -\log \hat{y}_n & y_i = n \\ \vdots & \\ -\log \hat{y}_1 & y_i = 1 \end{cases}$ $\qquad \hat{Y}_i = \begin{bmatrix} \hat{y}_n \\ \vdots \\ \hat{y}_1 \end{bmatrix}$

- $L_i(\hat{Y}_i, y_i) = -\log \hat{y}_i$

- $\hat{y}_i = p(y_i|\theta) = \frac{e^{f_i}}{\sum_j e^{f_j}}$

Softmax function

Example



f_1
-2.85
f_2
0.86
f_3
0.28
exp
0.058
2.36
1.32
norm
e^f
$\frac{e^f}{\sum_f e^f}$
0.016
0.631
0.353
$\hat{y}_i$
-log(0.353)
=1.04
$L_i(\hat{Y}_i, y_i = 3)$
Softmax
CE Loss

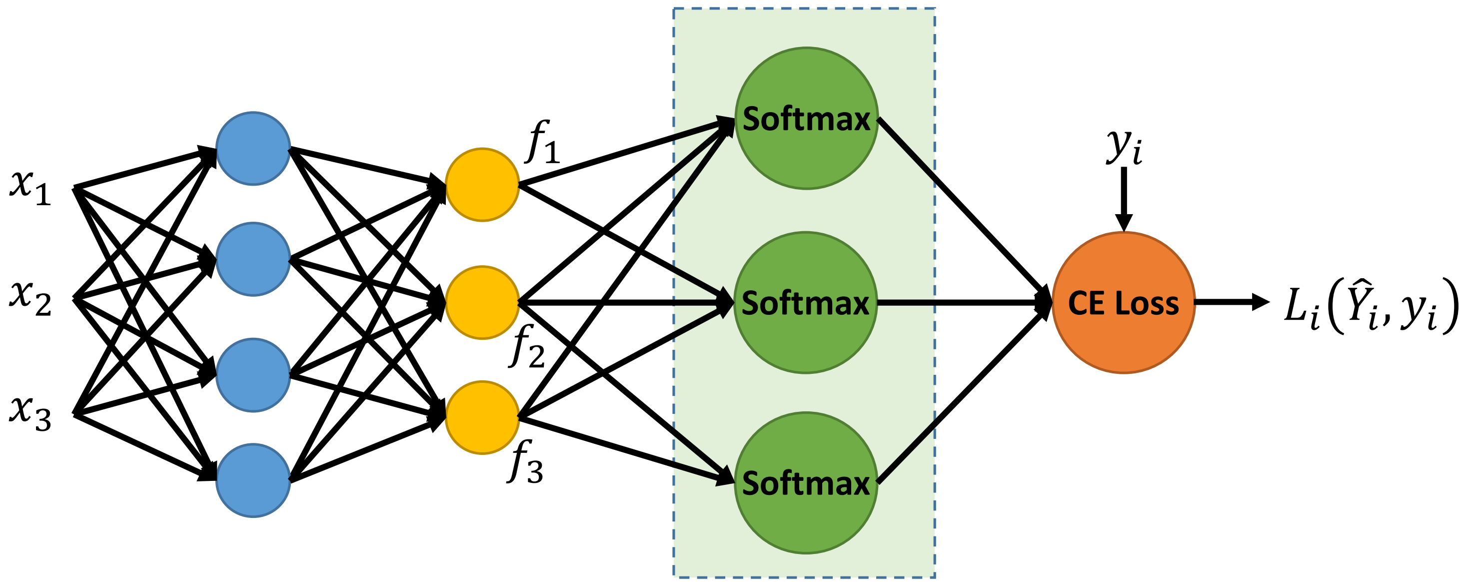
Neural Network Training

Back-propagation

+

Gradient Descent

Computational Graph (w/o Reg.)

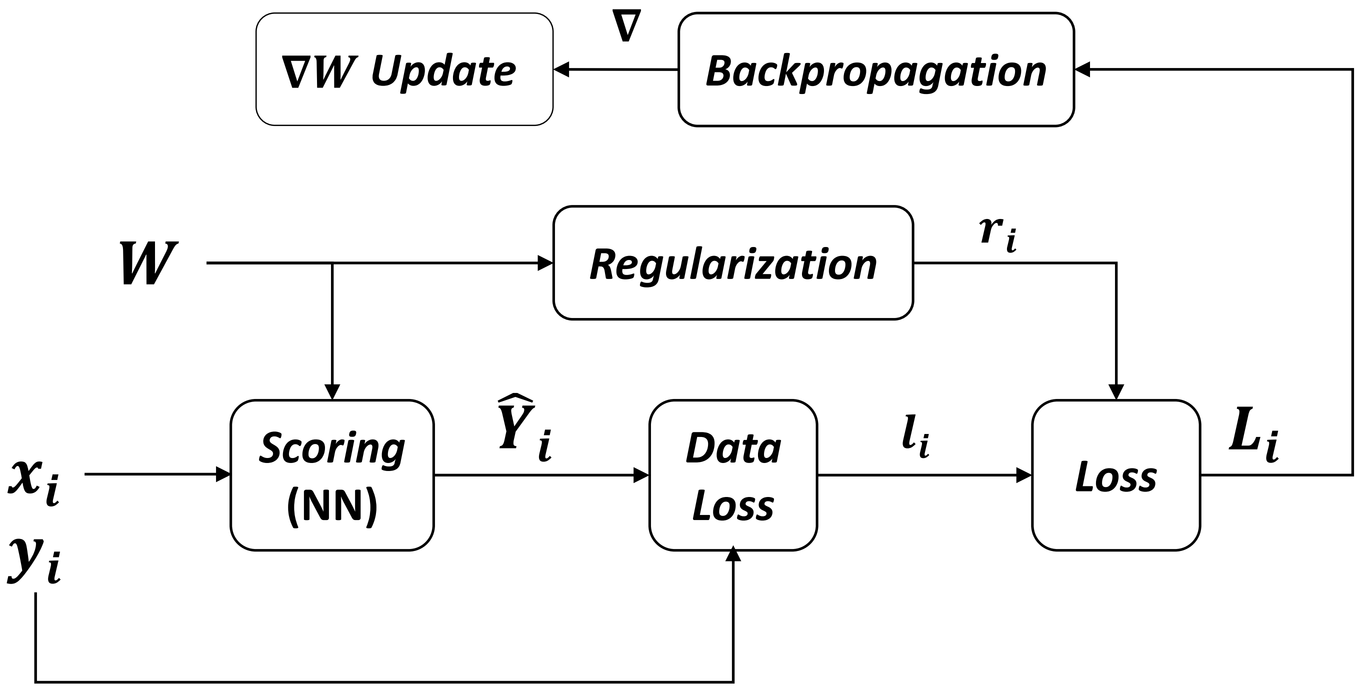


Softmax

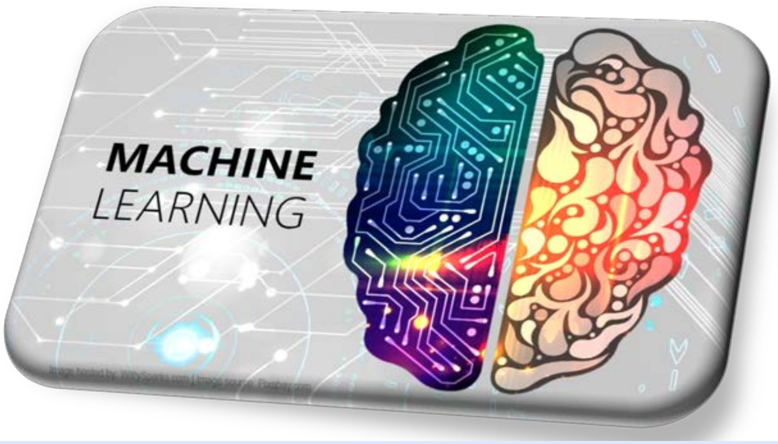
Hands-on Exercise

- Backpropagation for 3-class classification with 2-layer network, 2 input features, 2 hidden units (with sigmoid).

The Big Picture! (for 1 training ex.)



... then next example!



Data Loss Functions

Common Data Loss Functions

Binary Classification

$$L_i(\hat{y}_i, y_i) = -y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i)$$

MC Classification

$$L_i\left(\hat{Y}_i, y_i\right) = -\log \hat{y}_i \qquad , \hat{y}_i = p(y_i|\theta) = \frac{e^{f_i}}{\sum_j e^{f_j}}$$

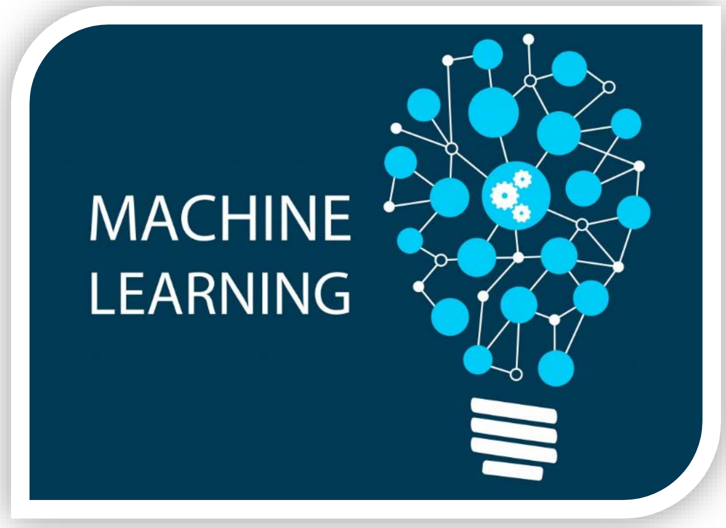
Common Data Loss Functions

Multi-Label Classification

$$L_i(\hat{Y}_i, Y_i) = \sum_j -y_{ij} \log \hat{y}_{ij} - (1 - y_{ij}) \log(1 - \hat{y}_{ij})$$

Regression

$$L_i = \|f - y_i\|^2$$



Regularization

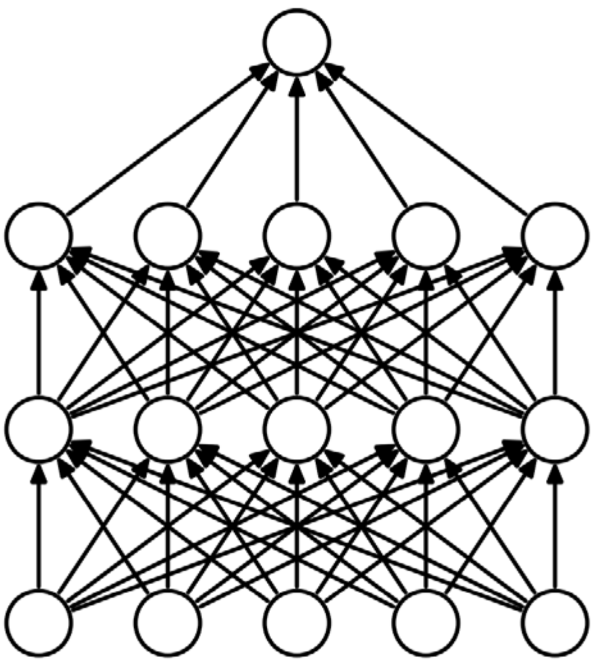
L2 & L1 Regularization

- We can still use them!

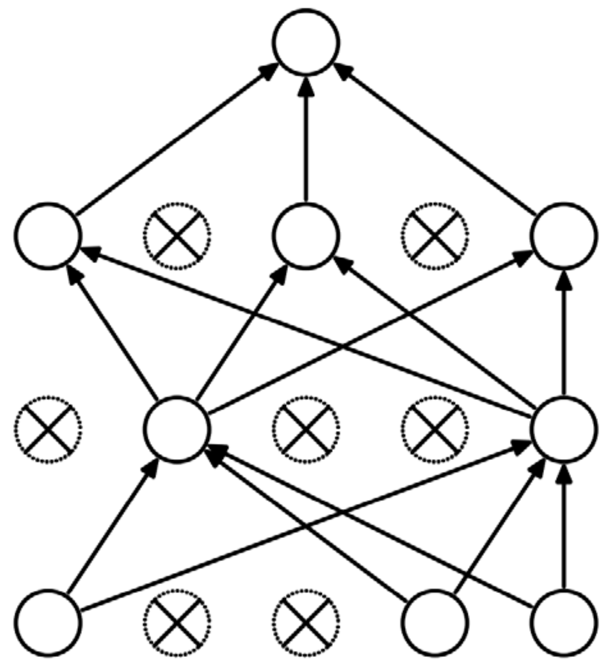
- **Elastic net regularization**: combining L1 regularization with L2 regularization: $\lambda_1 |w| + \frac{1}{2}\lambda_2 w^2$.

Dropout

- Extremely effective, simple and recently introduced.
- Keeping a neuron active with some probability *p* (a hyper-parameter), or setting it to zero otherwise.



(a) Standard Neural Net



(b) After applying dropout.

Dropout

- **Intuition**: By avoiding training all nodes on all training data, dropout decreases overfitting.

- The method also significantly improves training speed.

Prediction?

DropConnect

- Generalization of dropout: each connection, rather than each output unit, can be dropped with probability *p*.
  - i.e., a random set of weights is instead set to zero during forward pass.

Gradient Descent & Variants

Vanilla Gradient Descent

- Compute and accumulate gradients from all training examples before updating the parameters.

What happens with large training data?

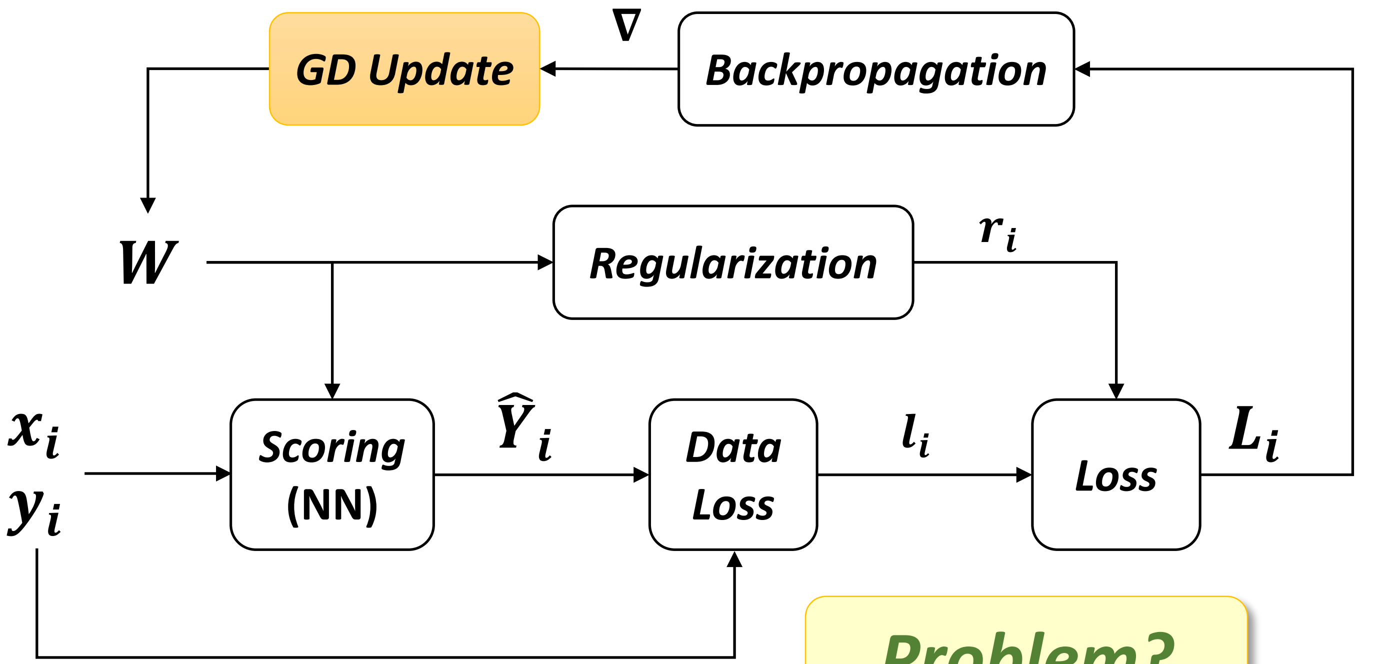
Mini-batch Gradient Descent

- Compute the gradient over ***batches*** of the training data.
- Gradient from a mini-batch is a good approximation of the gradient of the full objective.
  - much faster convergence by more frequent parameter updates.
- **epoch**: one full round over the entire data.
- mini-batch size: a hyper-parameter.
  - usually based on memory constraints (if any), or set to some value, e.g., 32, 64 or 128.

Stochastic Gradient Descent (SGD)

aka ***on-line gradient descent***

- The mini-batch is a single example!



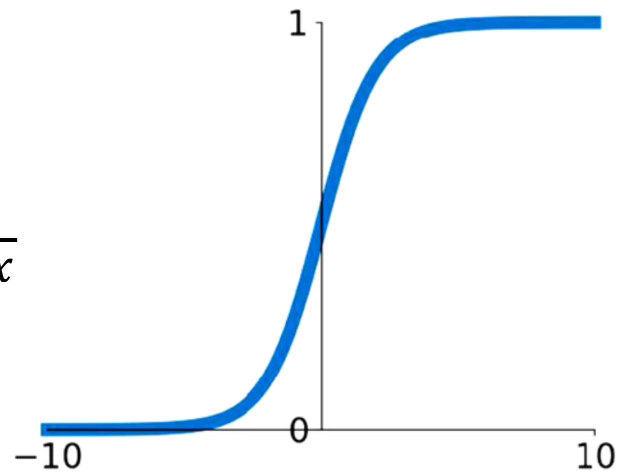
Problem?

When to use?

Activation Functions

Sigmoid

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



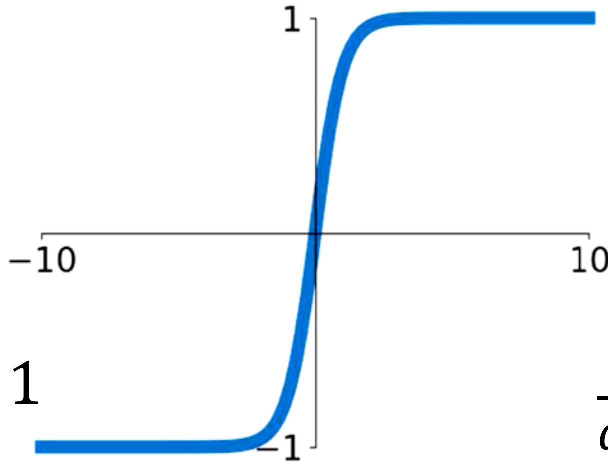
$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

- takes a real-valued number and "squashes" it into range between 0 and 1.
- used historically, but recently fallen out of favor:
  - **saturate (local gradient is very small) ➔ kill gradients**
  - outputs are not zero-centered ➔ inputs are always positive ➔ weight updates are all +ve or –ve ➔ zigzagging behavior in updates.

Tanh

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

$$\tanh(x) = 2\sigma(2x) - 1$$

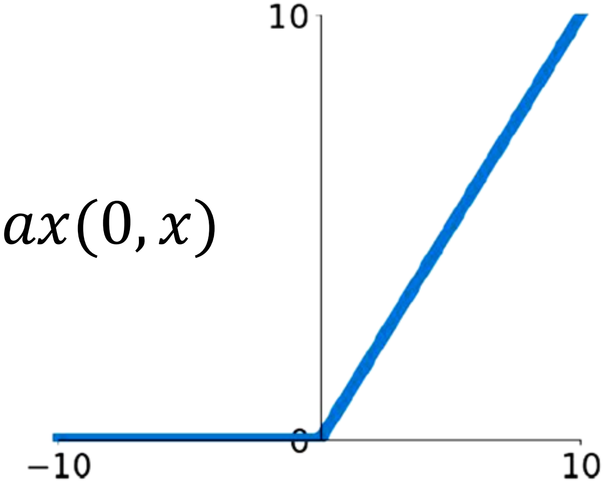


$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$$

- Squashes a real-valued number to the range [-1, 1].
- Saturate, but its output is zero-centered.
- Therefore, in practice the tanh non-linearity is always preferred to the sigmoid nonlinearity.

Rectified Linear Unit (ReLU)

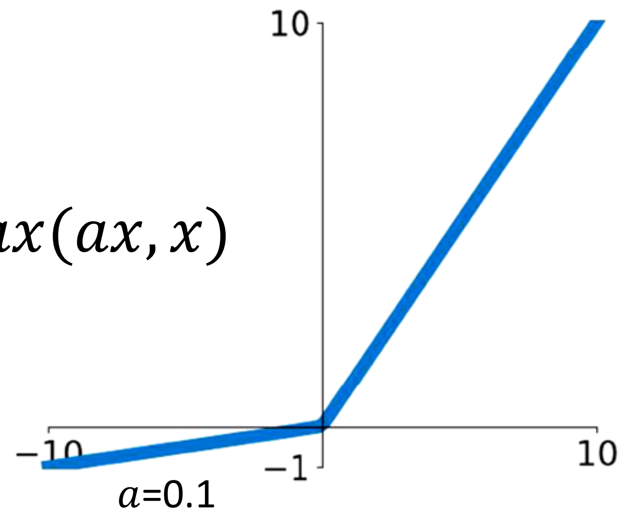
$$\text{ReLU}(x) = x^{+} = max(0, x)$$



- become very popular recently.
- accelerates the convergence.
- can be implemented simply by thresholding at zero.
- can "die" during training (once it outputs zero). Careful learning rate mitigates.

Leaky ReLU

$$\text{leaky ReLU}(x) = max(ax, x)$$



- fixing the "dying ReLU" problem.
- Some success but not always consistent.
- Slope can be a parameter.

Weight Initialization

Why bother?

- Sensitive to initialization
  - Objective is non-convex, many local optima

All-Zero Initialization

- Set all the initial weights to zero.
- Every neuron computes same output ➔ same gradients ➔ same parameter updates!

So what?

- No source of asymmetry between neurons.

Small Random Numbers

- Initialize weights to small numbers ➔ symmetry breaking.

- **Idea**: neurons will compute distinct updates and integrate themselves as diverse parts of the full network.

Sparse Initialization

- Set all weight matrices to zero, but to break symmetry every neuron is randomly connected to a fixed number of neurons below it.
  - e.g., 10.

Initializing the Biases

- More common to simply use 0 bias initialization.

Why not a problem?

- For ReLU, some people use small constant, e.g., 0.01, to ensures all ReLU units fire in the beginning and therefore obtain and propagate some gradient.

Demo: NN Playground ☺

***Play** with different architectures and training parameters here*:
http://playground.tensorflow.org

Our Goal!

جامعة قطر
QATAR UNIVERSITY

اللهم علّمنا ما ينفعنا
وانفعنا بما علمتنا
وزدنا علما

O' Allah, let us learn what would benefit us,
benefit us from what we learned,
and increase us in knowledge

Our Intention!

جامعة قطر
QATAR UNIVERSITY

من سلك طريقاً يلتمس فيه علماً
سهل الله له به طريقاً إلى الجنة

Whoever follows a path to seek knowledge therein, Allah will make easy for him a path to Paradise.