DATASCI207-005/007 Applied Machine Learning

Vilena Livinsky, PhD(c)

School of Information, UC Berkeley

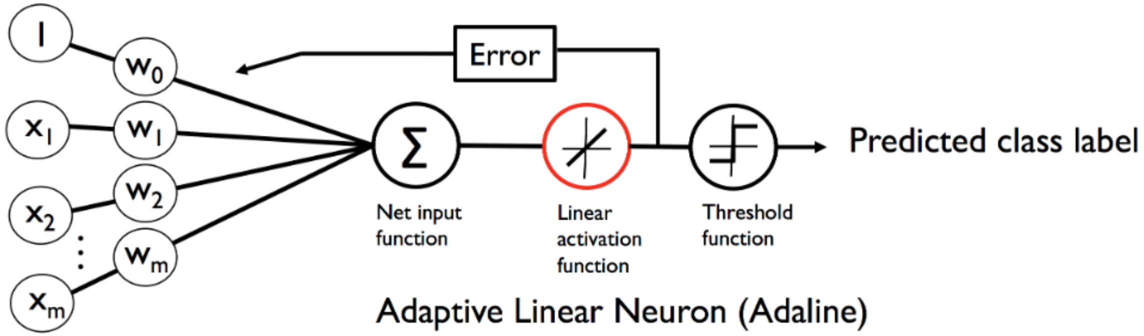
Week 6: 02/12/2024 – 02/13/2024

Today's Agenda

- Feedforward Neural Nets
- Activation Functions
- Regularization
- Walkthroughs:
  - Regularization: L1, L2, Dropout (NNs)
  - XOR
    - TensorFlow: NNs

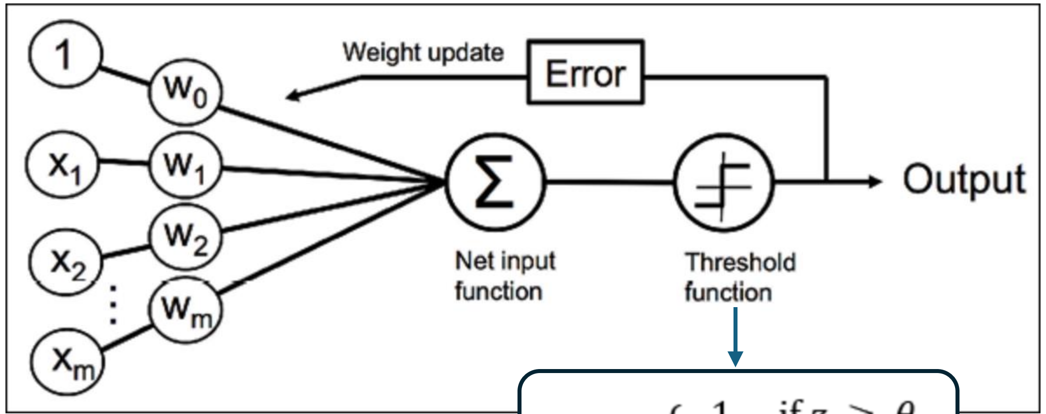
Perceptron: Binary Classification

Review



linear activation function, $\phi(z)$,

$$\phi(\boldsymbol{w}^T\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$$



$$\phi(z) = \begin{cases} 1 & \text{if } z \geq \theta, \\ -1 & \text{otherwise.} \end{cases}$$

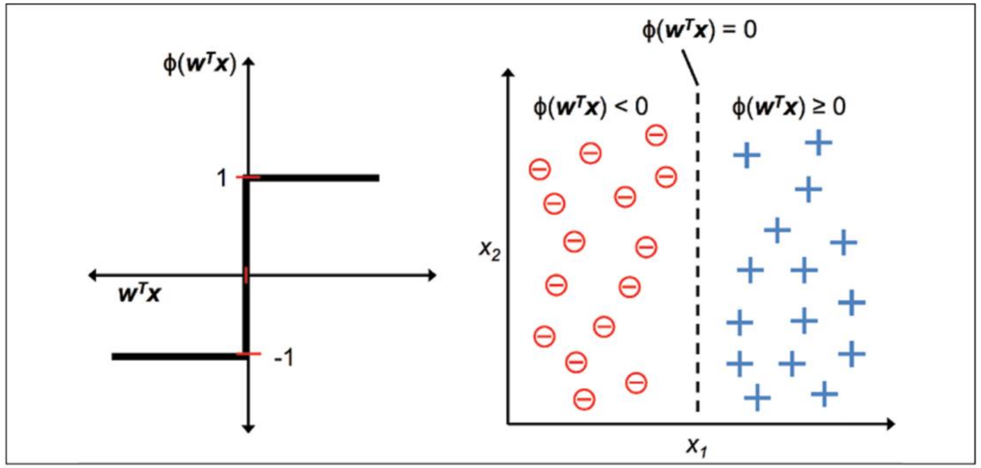
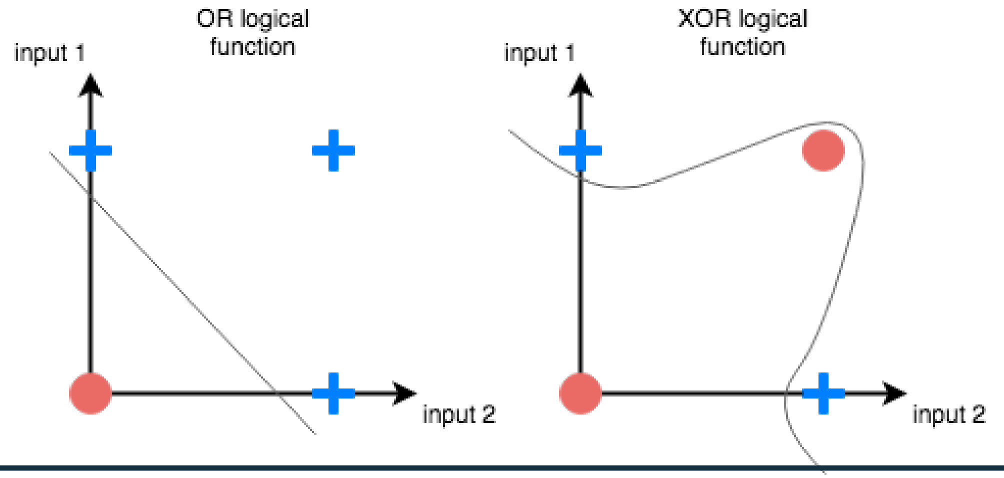
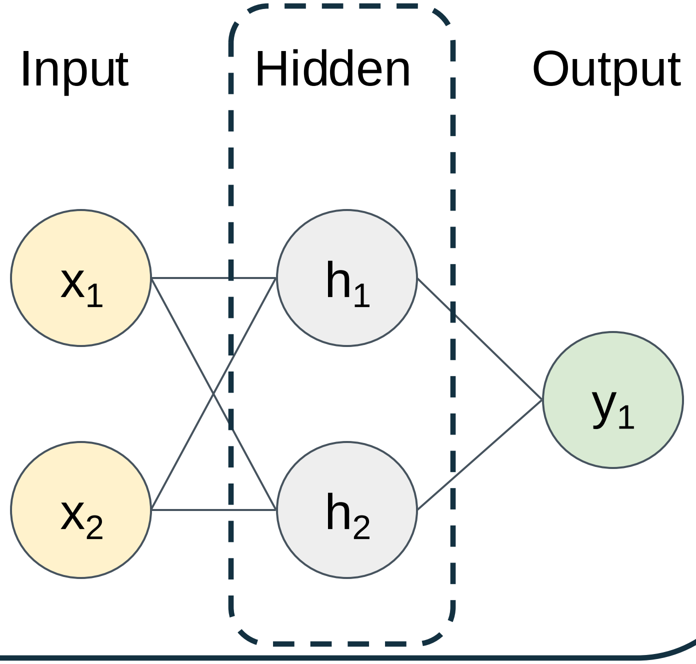


Image (Edited) Ref.: Raschka, S., & Mirjalili, V. (2019). Python Machine Learning, Third Edit.

Constructing XOR



x_1	x_2	h_1=OR(x_1,x_2)	h_2=NAND(x_1,x_2)	y=AND(h_1,h_2)
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0



Neural Nets: Activation Functions & XOR

Animations of ReLU in action:

https://blog.dailydoseofds.com/p/a-visual-and-intuitive-guide-to-what

https://nnfs.io/mvp/

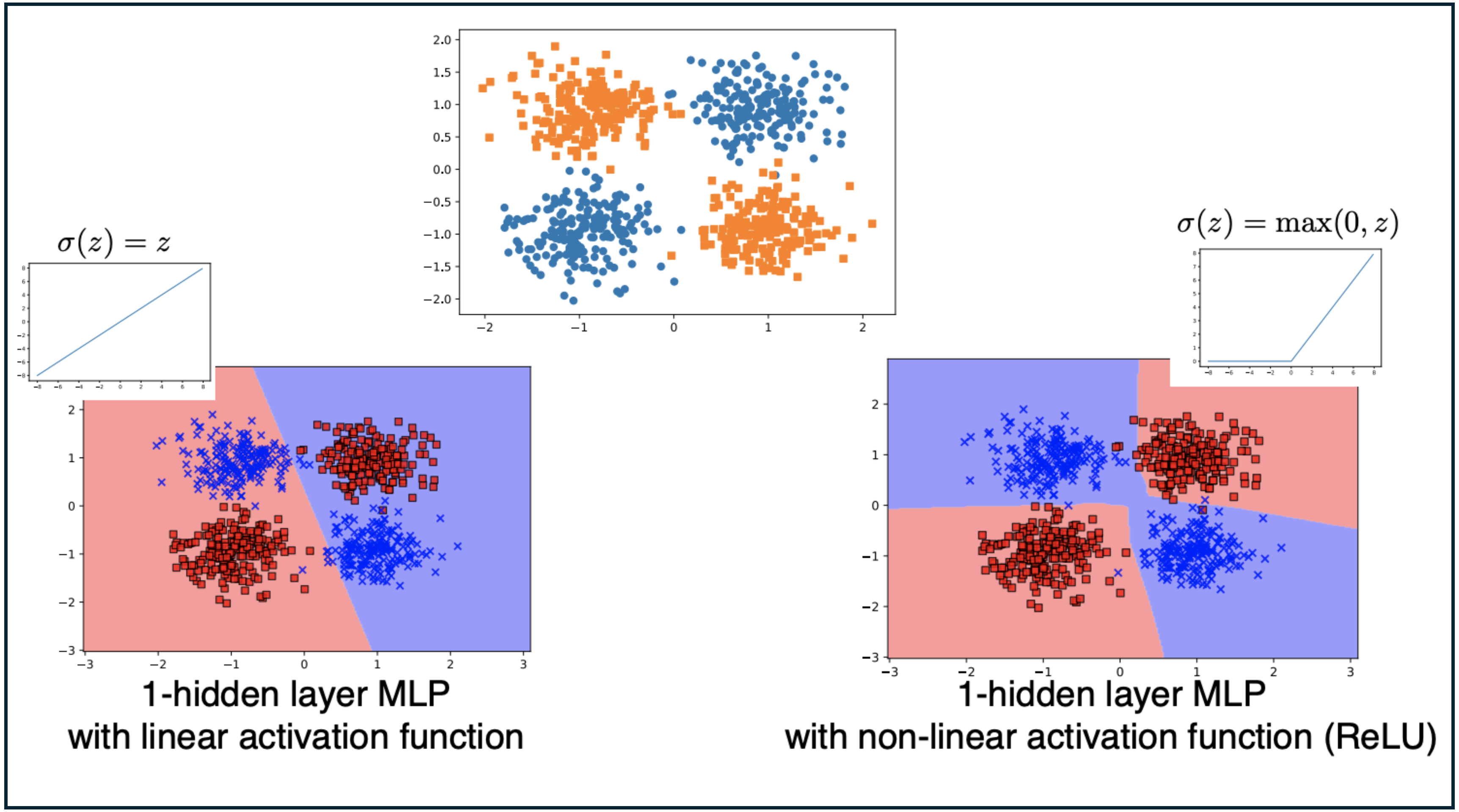


Image Source: Sebastian Raschka
https://github.com/rasbt/stat453-deep-learning-ss21/blob/main/L09/code/xor-problem.ipynb

Multilayer Feedforward Neural Net (or MLP)

- Thus far:
- Ex.: Softmax Regression
  - Input – [Softmax – Output]
- Learning Algorithm
  - Gradient Descent

- Multilayer Perceptron (MLP)
  - Fully connected *feedforward* neural nets with 1(+) **hidden** layers
    - Feedforward: going in one direction, left to right (Input → Output)
    - More hyperparameters:
      - *number of layers*
      - *units*
- Learning Algorithm
  - **Backpropagation**
    - Gradient descent using the chain rule

MLP: Examples, Classifiers

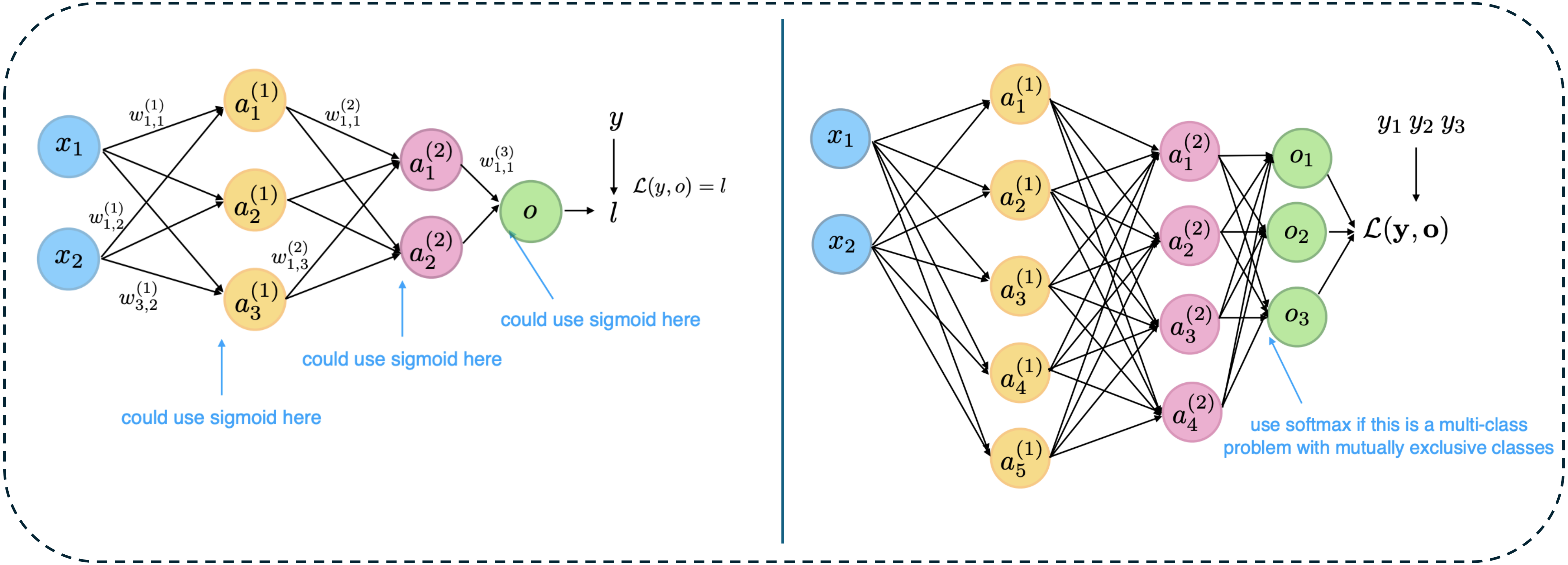


Image Source: Sebastian Raschka, Introduction to Deep Learning and Generative Models

Activation Functions

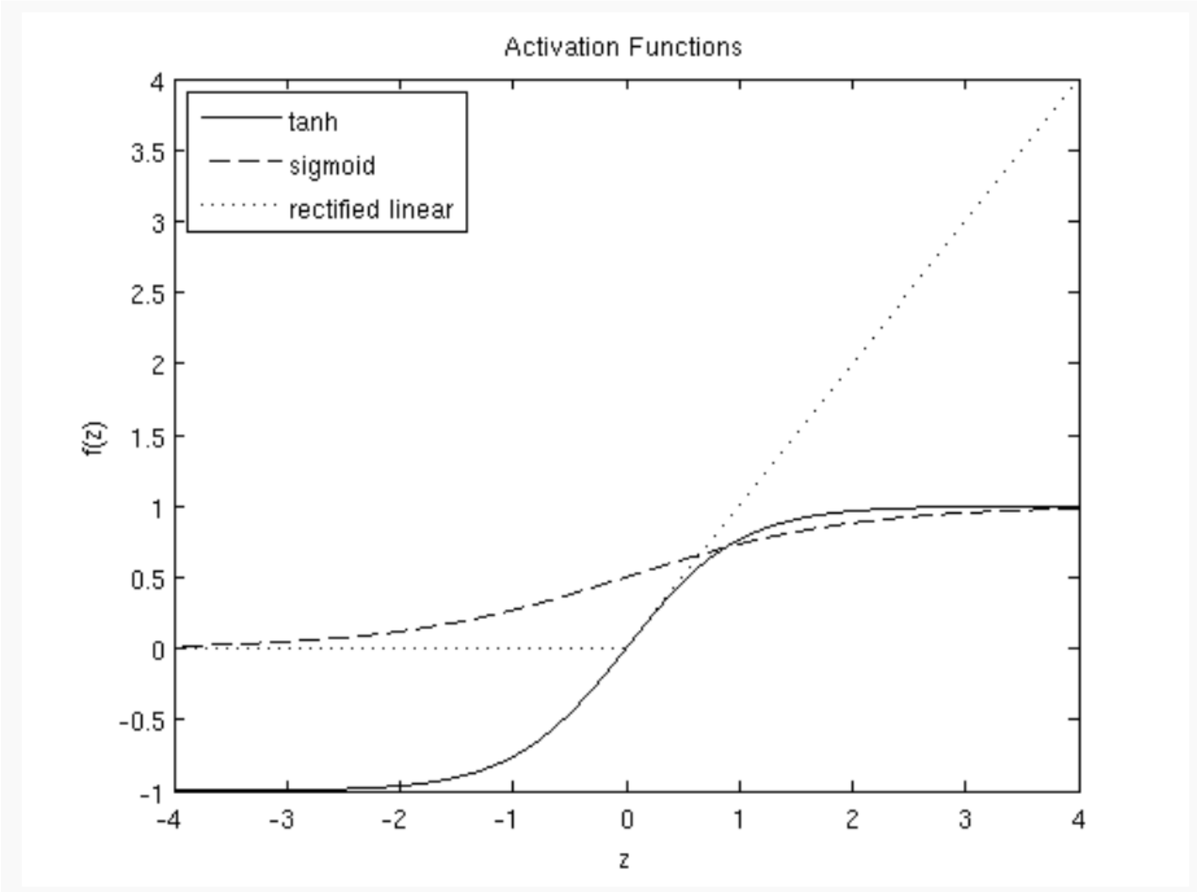
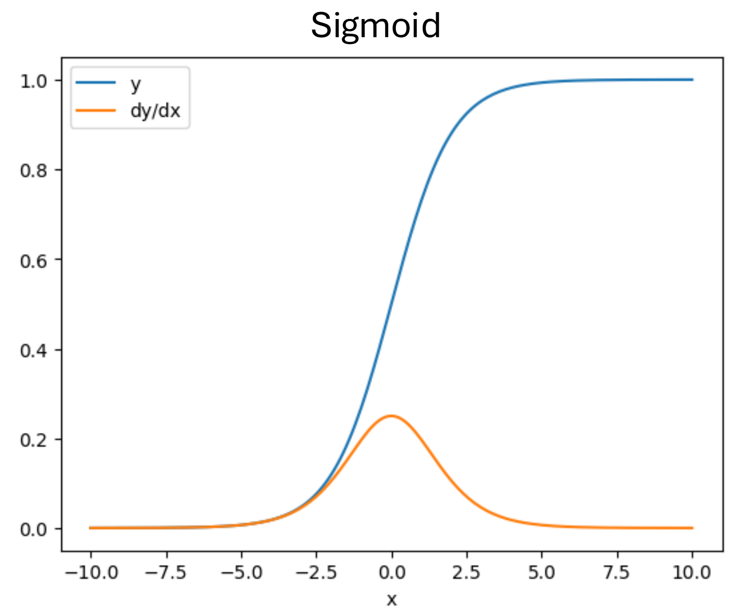
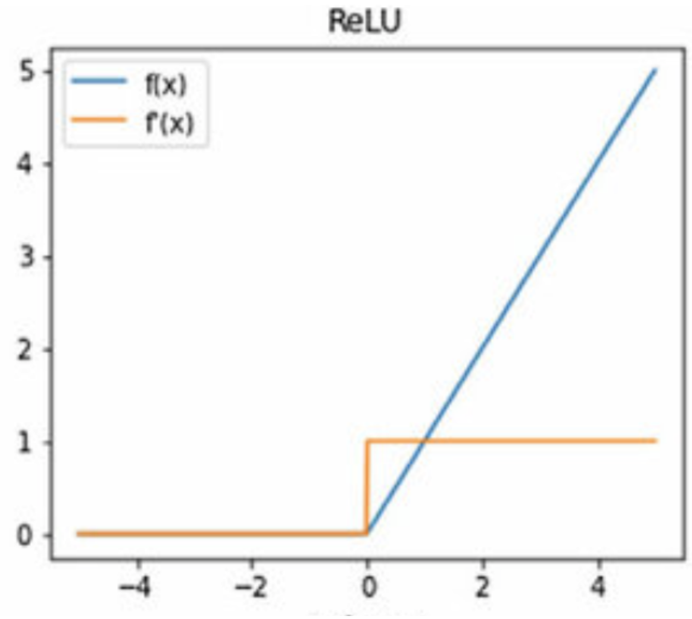


Image Source:
http://deeplearning.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

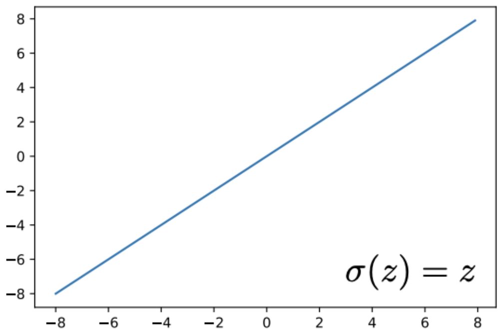
Derivatives



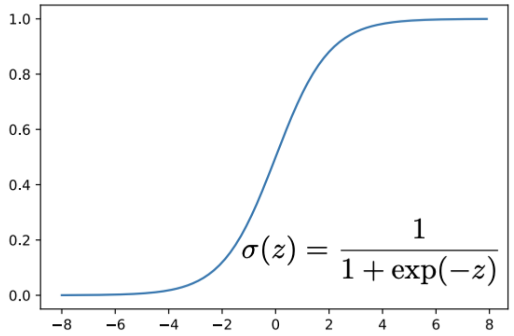


Activation Functions: Some Others

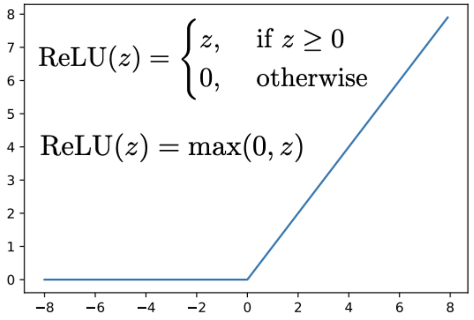
Identity



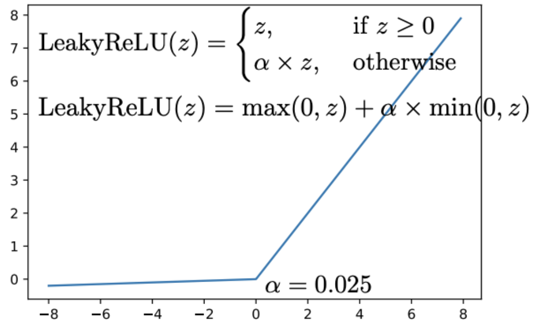
(Logistic) Sigmoid



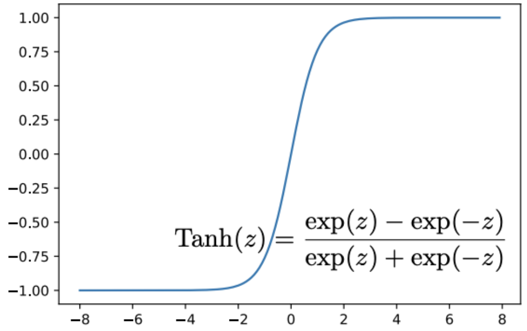
ReLU (Rectified Linear Unit)



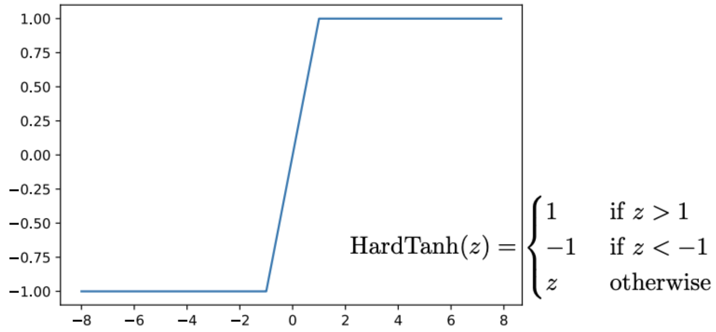
Leaky ReLU



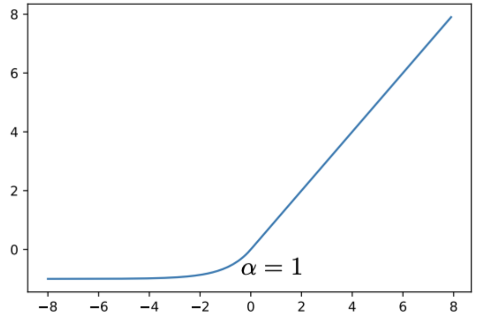
Tanh ("tanH")



Hard Tanh



ELU (Exponential Linear Unit)



$\mathrm{ELU}(z) = \max(0, z) + \min(0, \alpha \times (\exp(z) - 1))$

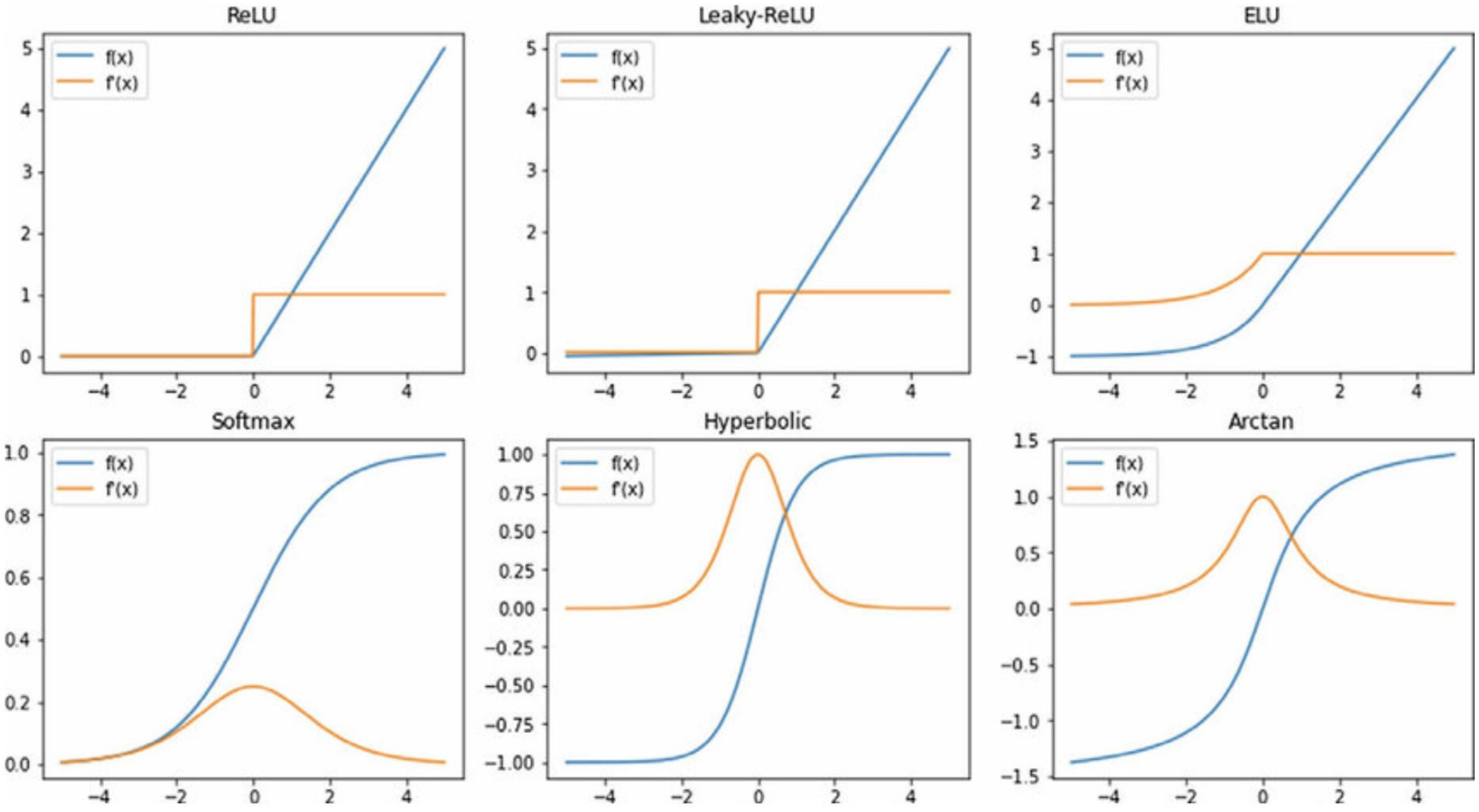
PReLU (Parameterized Rectified Linear Unit)

here, alpha is a trainable parameter

$$\mathrm{PReLU}(z) = \begin{cases} z, & \text{if } z \geq 0 \\ \alpha z, & \text{otherwise} \end{cases}$$

$$\mathrm{PReLU}(z) = \max(0, z) + \alpha \times \min(0, z)$$

Activation Functions: Derivatives

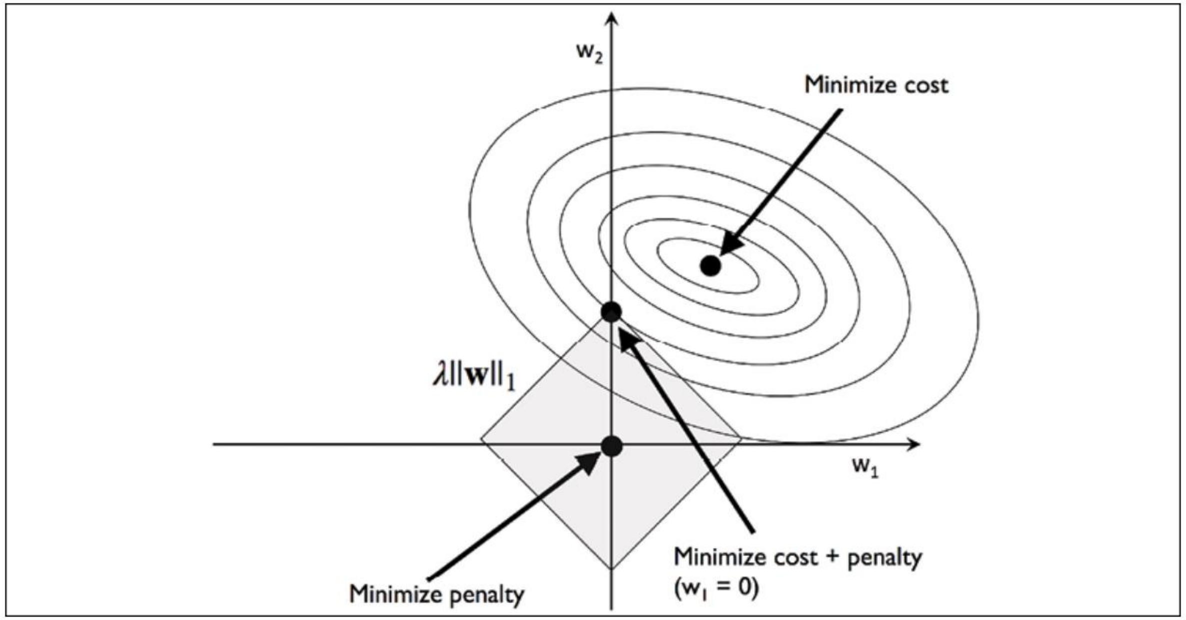


Commonly used activation functions, f(x) and their derivatives, f (x) for x ∈ [− 5, 5]

Image Source: A. Seth, A. Nikou and M. Daoutis. A scalable species-based genetic algorithm for reinforcement learning problems. The Knowledge Engineering Review 37(e9): 1–46.

Regularization: L1 (Lasso) vs. L2 (Ridge)

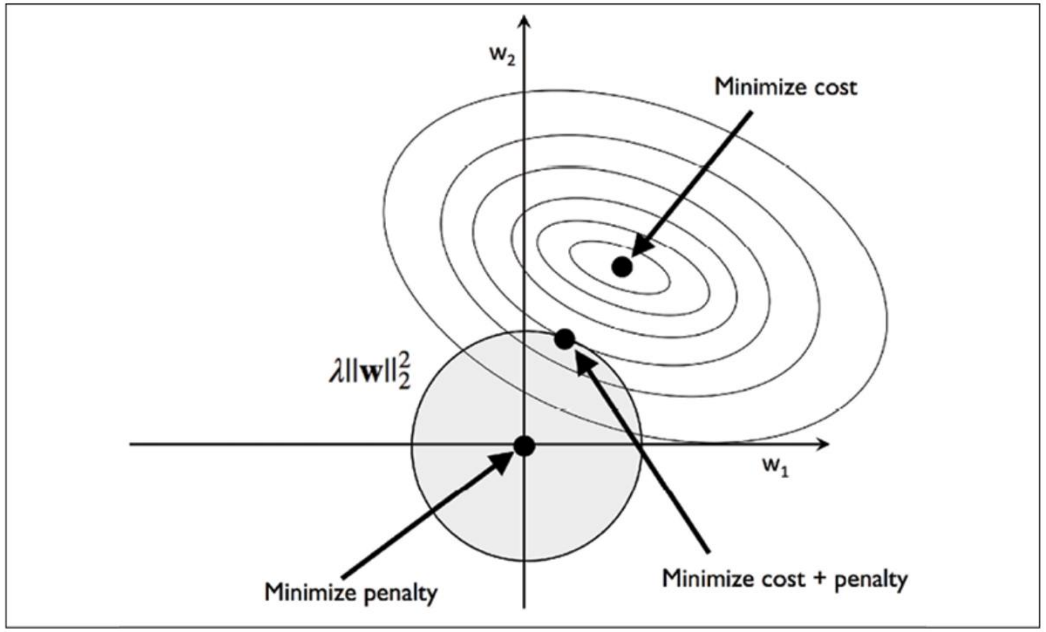
$$L1: \quad \|\mathbf{w}\|_1 = \sum_{j=1}^{m} |w_j|$$



$$L2: \quad \|\mathbf{w}\|_2^2 = \sum_{j=1}^{m} w_j^2$$

$$\text{Cost}_{\mathbf{w},\mathbf{b}} = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(y^{[i]}, \hat{y}^{[i]})$$

$$\text{L2-Regularized-Cost}_{\mathbf{w},\mathbf{b}} = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(y^{[i]}, \hat{y}^{[i]}) + \boxed{\frac{\lambda}{n}\sum_{j} w_j^2}$$

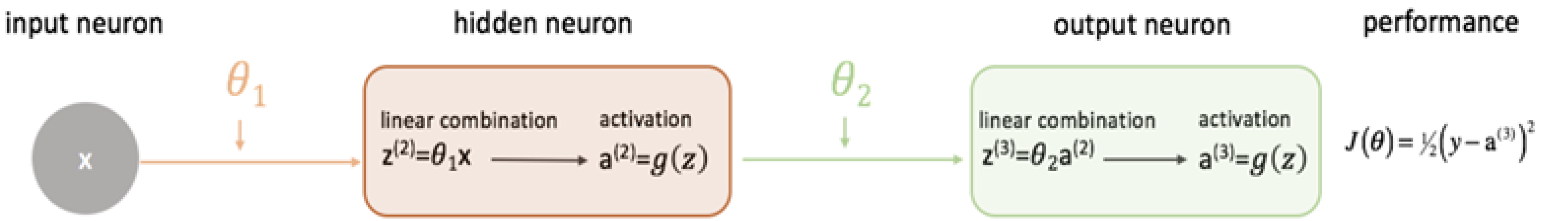


Keras API info on regularizers:
https://keras.io/api/layers/regularizers/

Example Implementation:
https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/first_edition/4.4-overfitting-and-underfitting.ipynb

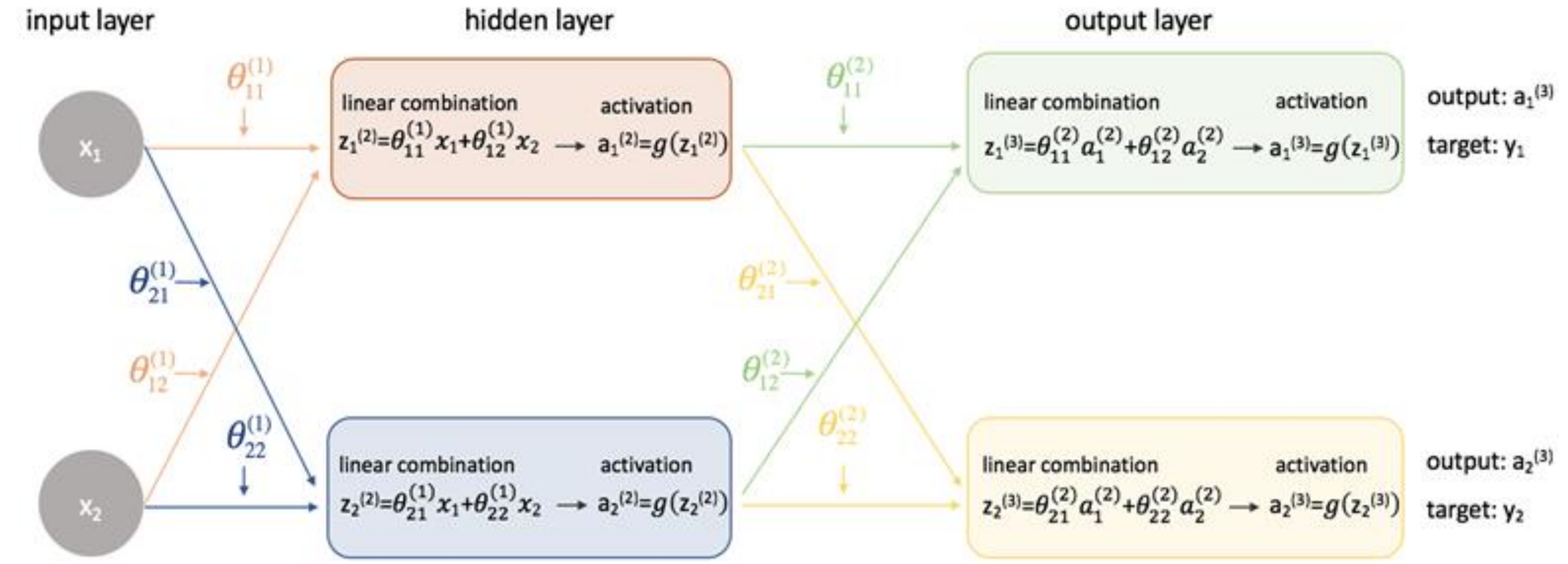
Image Ref.: Raschka, S., & Mirjalili, V. (2019). Python Machine Learning, Third Edit.

Neural Nets: Backpropagation



$$\frac{\partial J(\theta)}{\partial \theta_2} = \left(\frac{\partial J(\theta)}{\partial a^{(3)}}\right)\left(\frac{\partial a^{(3)}}{\partial z}\right)\left(\frac{\partial z}{\partial \theta_2}\right)$$

$$\frac{\partial J(\theta)}{\partial \theta_1} = \left(\frac{\partial J(\theta)}{\partial a^{(3)}}\right)\left(\frac{\partial a^{(3)}}{\partial z^{(3)}}\right)\left(\frac{\partial z^{(3)}}{\partial a^{(2)}}\right)\left(\frac{\partial a^{(2)}}{\partial z^{(2)}}\right)\left(\frac{\partial z^{(2)}}{\partial \theta_1}\right)$$

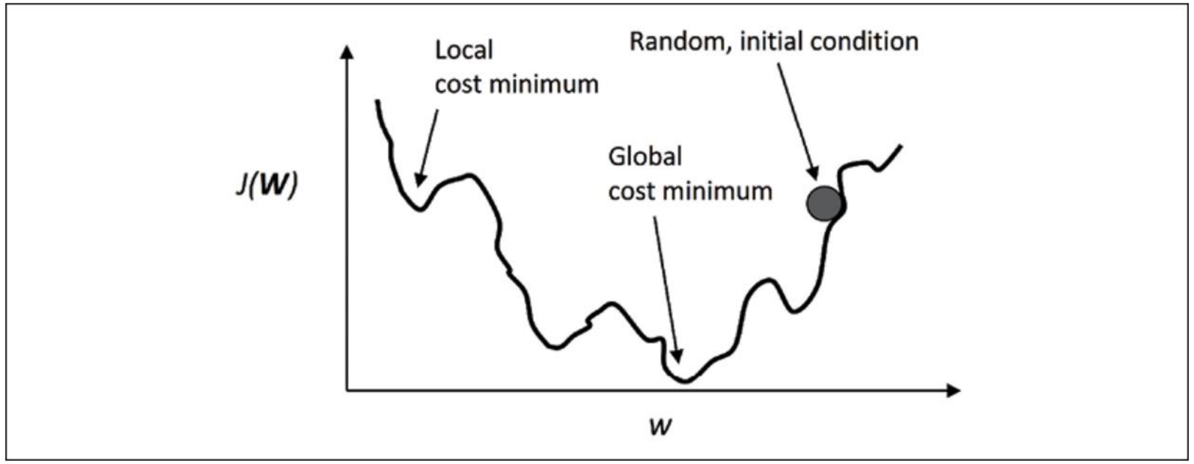


Non-linear Functions

If *f* is a linear function and *g* is a linear function, then their composition *f(g(x))* is a linear function

Without non-linear activations, all neural network functions could be reduced to a single layer

Multilayer NNs: Cost Functions



Ref.: Raschka, S., & Mirjalili, V. (2019). Python Machine Learning, Third Edit.

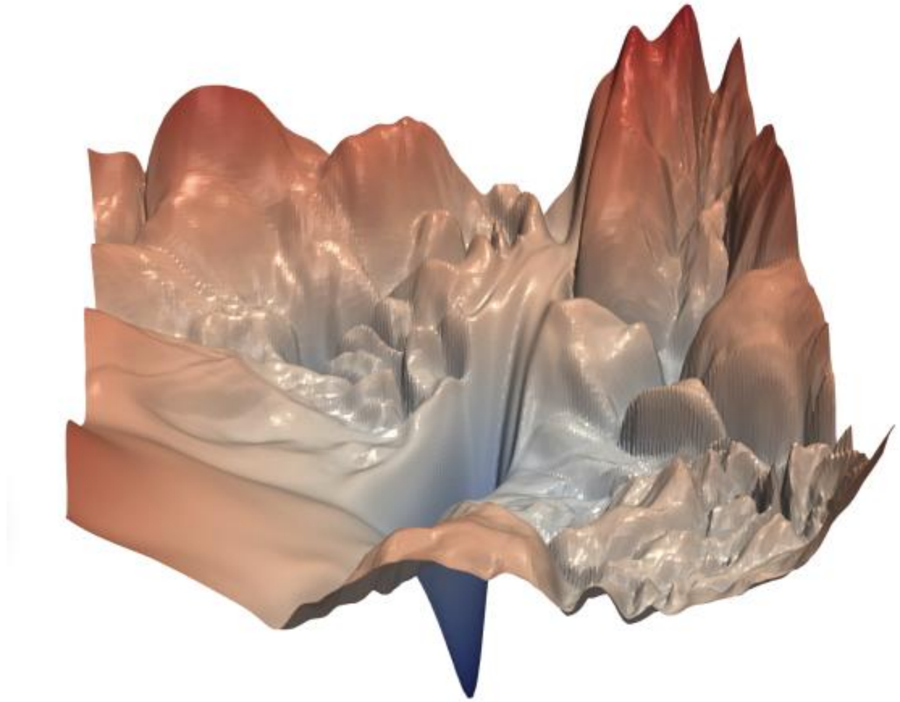


Image Source: Li, H., Xu, Z., Taylor, G., Studer, C., & Goldstein, T. (2018). Visualizing the loss landscape of neural nets. Advances in neural information processing systems, 31.