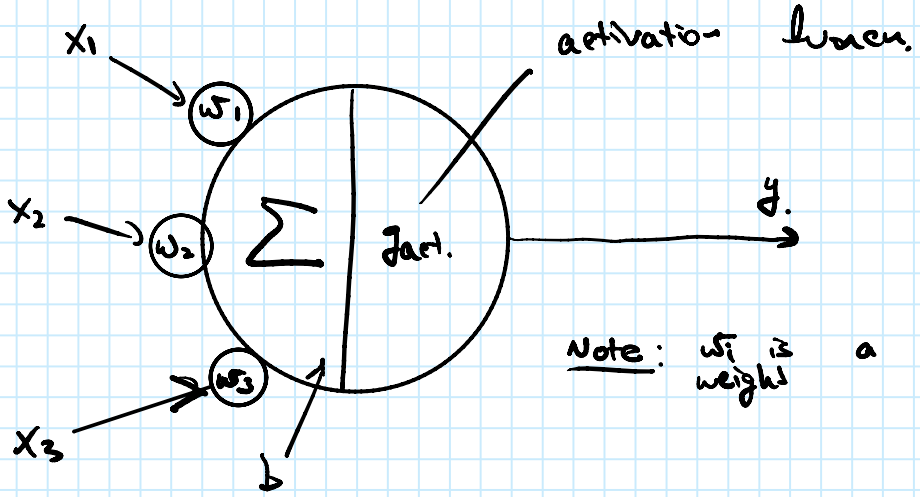
Lecture 17

Wednesday, March 16, 2022 9:11 AM

Admin
① HW#3 due Friday.
② Exam Mar 30th.

Nonlinear Modeling - Part II

Last time: "perceptron"



Note: w_i is a tunable input weight

Key principle: - Given some dataset $\{x_1, \dots x_n\}$ and a desired output y, the weights w_i and bias b can be chosen to create a "perceptron" that produces the same input-out behavior.

- The perceptron works well as a classifier for linearly seperable data.
- The bias "b" allows the input threshold for the activation funcn. to be adjusted for better fit the data.

- Activation functions for Artificial neurons:
  - The activation can be written as:

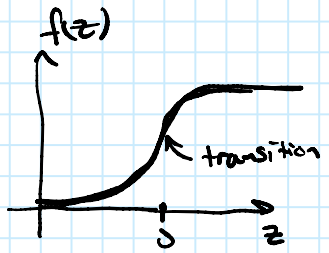
$$g_{Act}(z) \quad \text{where} \quad z = \sum_{i=1}^{\text{\# of inputs}} x_i w_i + b$$

  - $g_{Act}(z)$ can take on several different forms, depending on the application.

$f(z)$

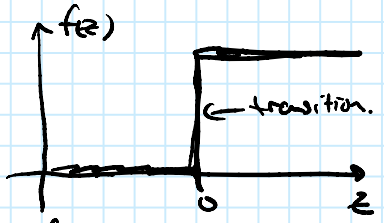
depending on the application.

1) Sigmoid function: $f(z) = \frac{1}{1+e^{z}}$

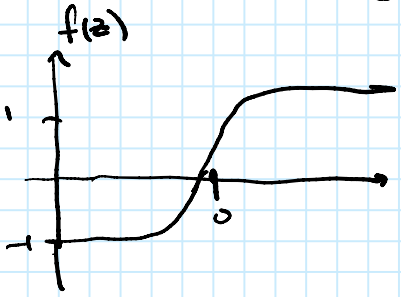


- generally outputs ≥ 0, ≤ 1, but can have smaller outputs @ transition.

2) Threshold func: $f(z) = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ if } z < 0 \end{cases}$

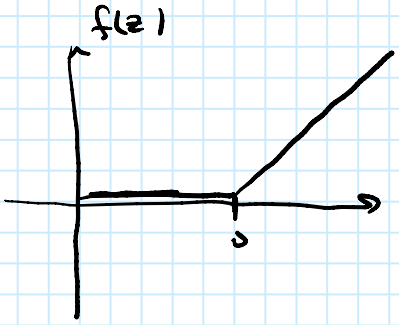


3) Hyperbolic tangent: $f(z) = \frac{1-e^{-2z}}{1+e^{-2z}}$

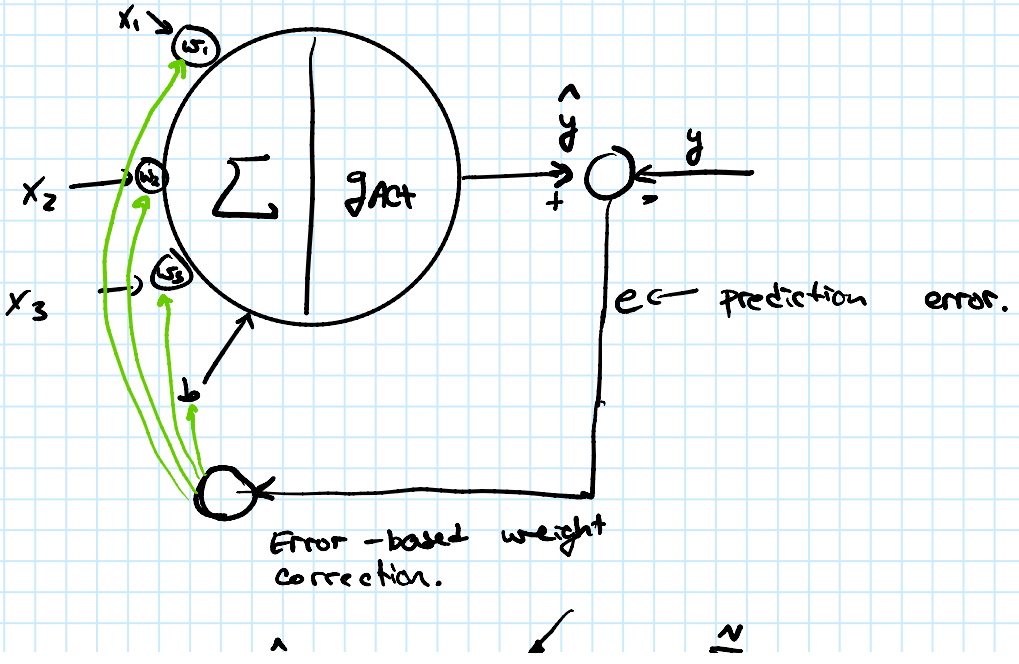


4) Rectified linear unit: $f(x) = \max(x, 0)$

- Enables positive, proportional outputs
- Also known as "ReLu"



- "Training" a perceptron by adjusting its input weights.



correction.

- Error term: $e = \hat{y} - y = g(z) - y$; $z = \sum_{i=1}^{N} w_i x_i + b$.

- one way to make corrections is to use gradient descent

$$\Delta w_i = -\rho \, \text{grad}_{w_i} e^2 = -\rho 2e \frac{de}{dw_i}$$

"learning rate" — change in e w.r.t w_i

- The weight correction can be written as:

$$\Delta w_i = -2\rho \frac{de}{dw_i} e \left(\frac{dz}{dw_i}\right) = x_i$$

↑ Assumes $g(z)$ is differentiable.

- This type of learning is referred to as "supervised learning"
  - used when you have a specific way you want the perceptron to respond to input data.
  - For each data sample, j, provide (x_1^j, x_2^j, x_n^j) & y^j

  => Supervised learning rule: Δw_i = (Inputs x_i) (Error).

- There is also a type of training called "unsupervised learning.
  - when you want the perceptron to create its own classification rule.

  => Unsupervised learning rule: $\Delta w_i \propto$ (Input x_i) (output y_i)

  => see self organizing maps.

- procedures for training a perceptron:
  - There are generally two ways to train: one is a batch processing.

If we assume a linear activation funcn:

$\hat{y} = g(z) = \sum_{i=1}^{n} w_i x_i + b$

given sample data:

n # samples.

given sample data:

$$(y^j, x_1^j, \dots x_n^j) \mid j = 1, \dots, N$$ ← # samples.

we want to find the weights to minimize

$$J_N = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}^i - y^i)$$

⇒ each weight can be computed by.

$$\Delta w_i = -\rho \, \text{grad}_{w_i} J_N = -\rho \frac{2}{N} \sum_j (\hat{y}^j - y^j) \frac{\partial \hat{y}^j}{\partial w_i}$$

⇒ Requires that we store the gradient for all sample data $j = 1, \dots, N$.

- The other way we can process is to update weights based on each new piece of data

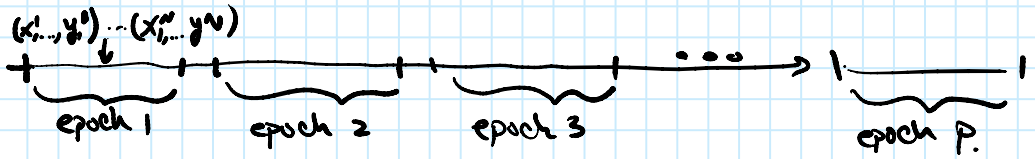
For each new sample, k

$$\Delta^k w_i = \rho \, \delta^k x_i^k \text{, where}$$

$$\delta^k = y^k - \underbrace{\sum w_i^k x_i^k - b}$$

y^k: correct output; $\sum w_i^k x_i^k - b$: predicted output $\hat{y}$ @ k^{th} presentation based on inputs of data.

Learning procedure: Train perceptron by presenting all N training data in a sequence, called an "epoch"



→ This process of "presenting data multiple times is is called "Recycling"

→ The learning procedure is called the Widrow-Hoff Algorithm.

- Convergence of Widrow-Hoff Algorith.

- Convergence of Widrow-Hoff Algorithm.

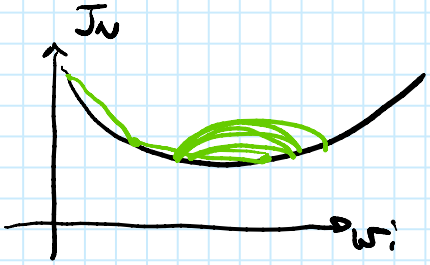
  - If regeling is repeated infinite times, will the optimal weight vector be found?

    => $\arg\min J_N(w_1, w_2, \dots w_n)$

  - Answer: If a constant learning rate $\rho > 0$ is used, the algorithm only converges if $\min J_N = 0$

    If $J_N \neq 0$, Δw_i will also be nonzero for each epoch.

    => J_N will <u>not</u> settle.



-> To prevent this, we can use a variable learning rate:

$$\rho_k = \frac{\text{constant}}{k}$$

=> This dampens oscillations as k increases, improving convergence.