CS21SI: AI for Social Good

Lecture 3: Training Deep Neural Networks

Deep Learning

Plan for Today

- Motivation: COMPAS
- Review of neural networks
- Activation functions
- Implementing deep neural networks
- Regularization and hyperparameter tuning

1. Recidivism Score

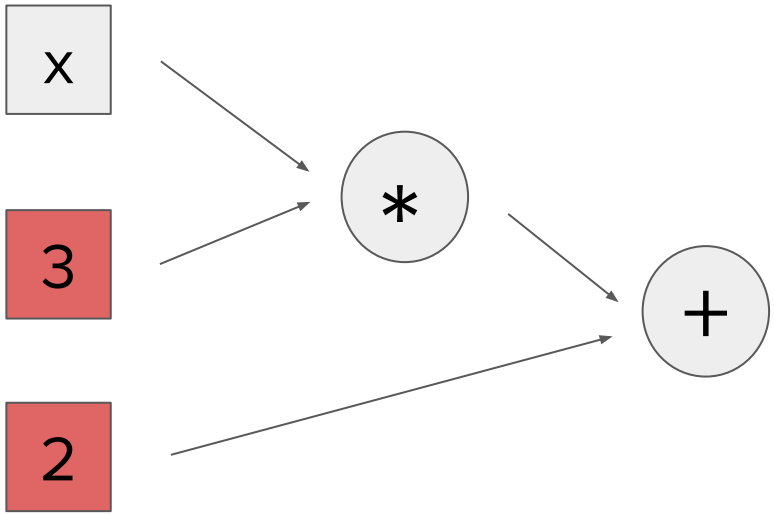
2. Violence Score

Jupyter Exercise 1: Visualize the Data

Task: Recreate COMPAS recidivism scores from data about defendants

Revisiting Computational Graphs

3x + 2:

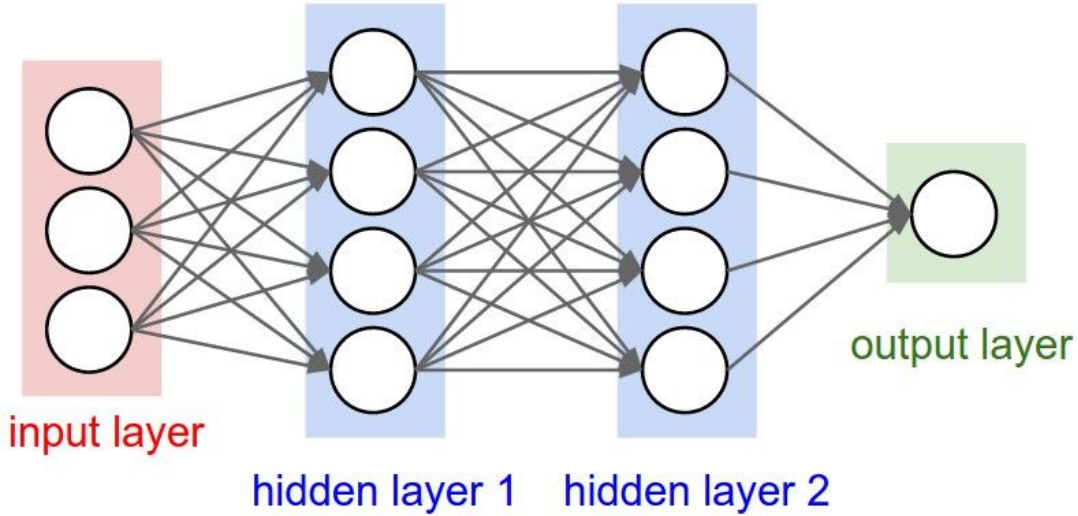


- Computational graphs can represent any function
- “Forward pass” to compute values/losses
- “Backward pass” to compute gradients

Jumping into Neural Networks

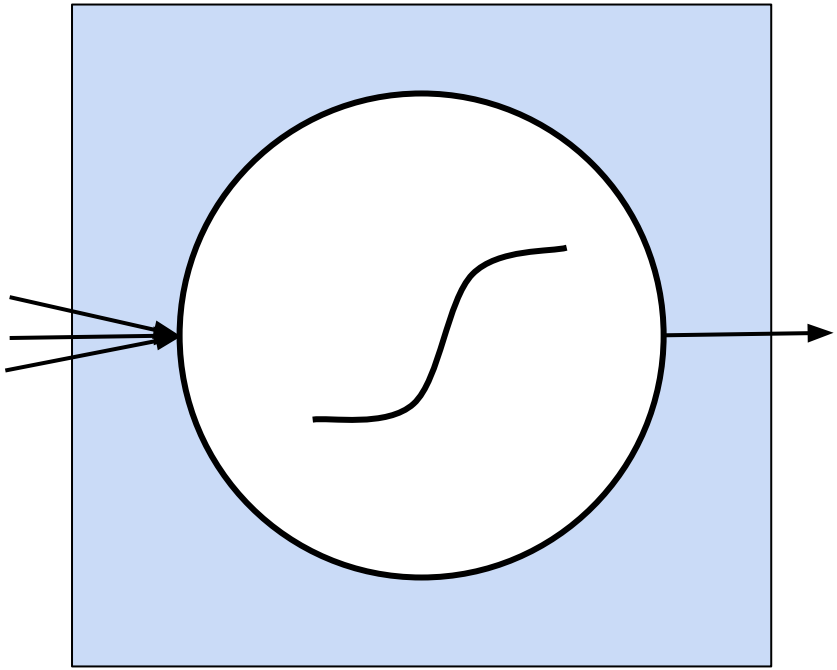
- In the past few weeks we've been discussing logistic regression, which is just a **single-layer neural network**
- A **layer** is the encompassing term for a weight multiplication followed by an activation function (we'll discuss these more today)
- We can stack multiple layers back-to-back to make deeper networks (and this is where the term **"deep learning"** from)

Neural Networks (aka Stacked Layers)



- Linear layers with non-linearities in between.
- Each arrow is a weight
- The nodes in the hidden layers represent operations...

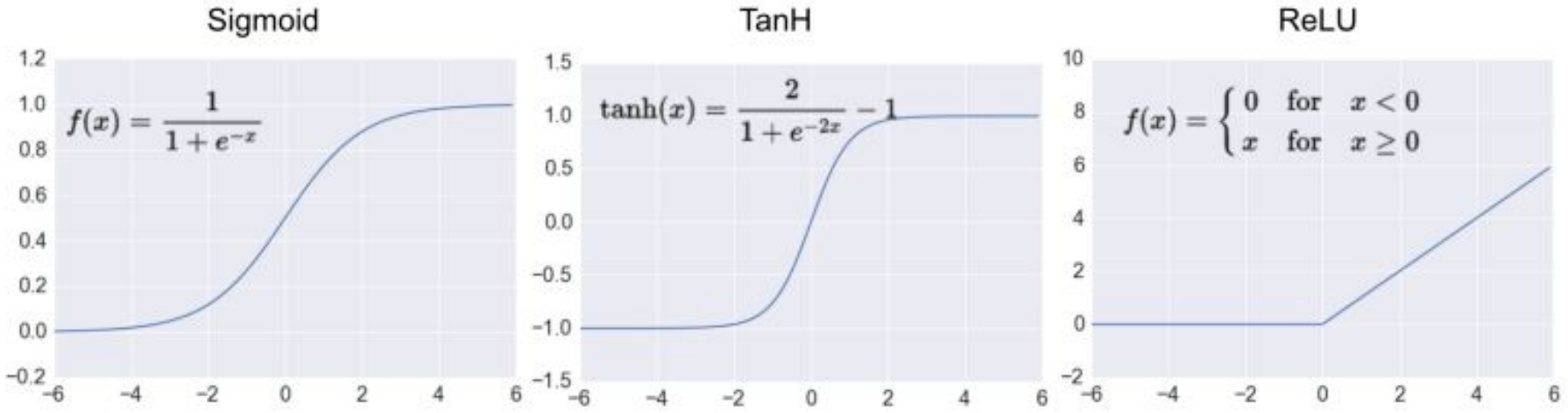
Zooming into the Hidden Layer



Each node encapsulates a non-linearity, or an **activation function**

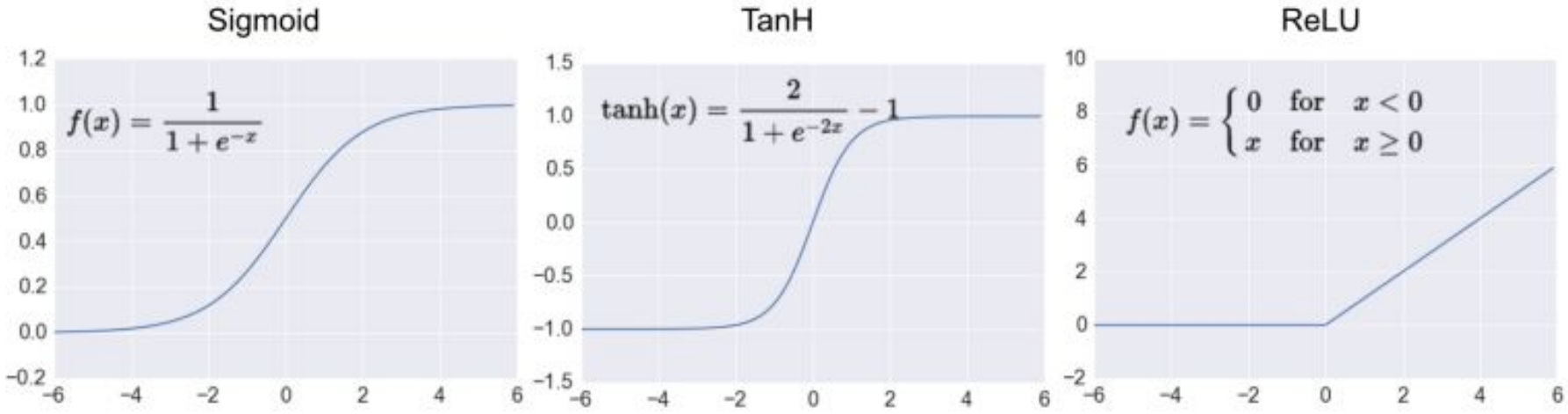
Questions?

Activation Functions



Why non-linear?

Activation Functions

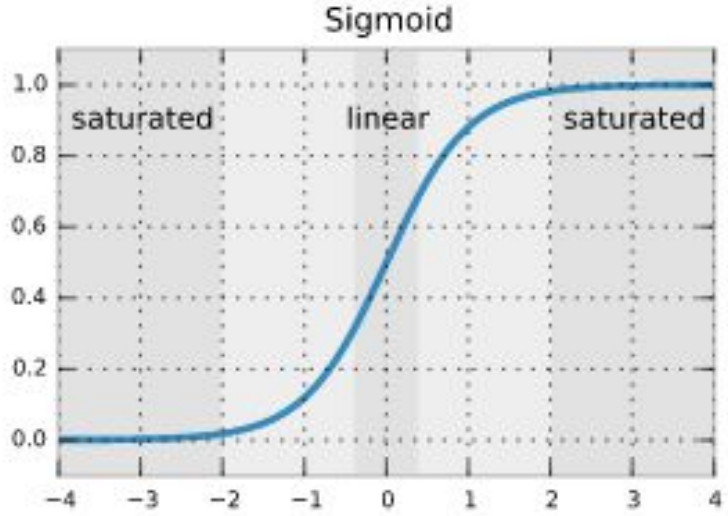


Why non-linear? Creates non-linear decision boundaries!

Activation Functions: **Sigmoid**

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

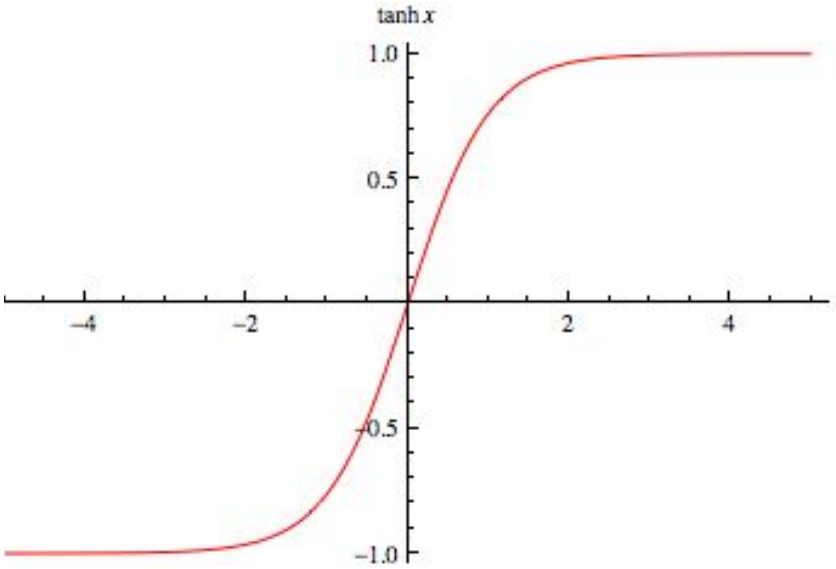
- Range: (0, 1)
- Numbers in the linear regime are good
- Saturation can kill gradients
- Not zero-centered
- exp() can be expensive



Activation Functions: **tanh**

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \\ = \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{1 - e^{-2x}}{1 + e^{-2x}}.$$

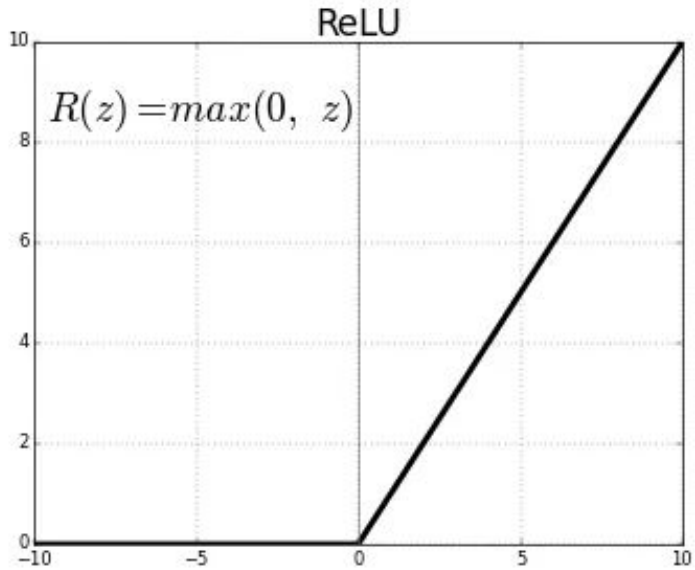
- Range: (-1, 1)
- Scaled sigmoid
- Now zero-centered!
- Saturation still kills gradients...



Activation Functions: **ReLU**

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

- ReLU = **Re**ctified **L**inear **U**nit
- Range: (0, +infty)
- Does not saturate in positive region
- Very efficient + converges fast
- Not zero-centered
- Dying ReLU

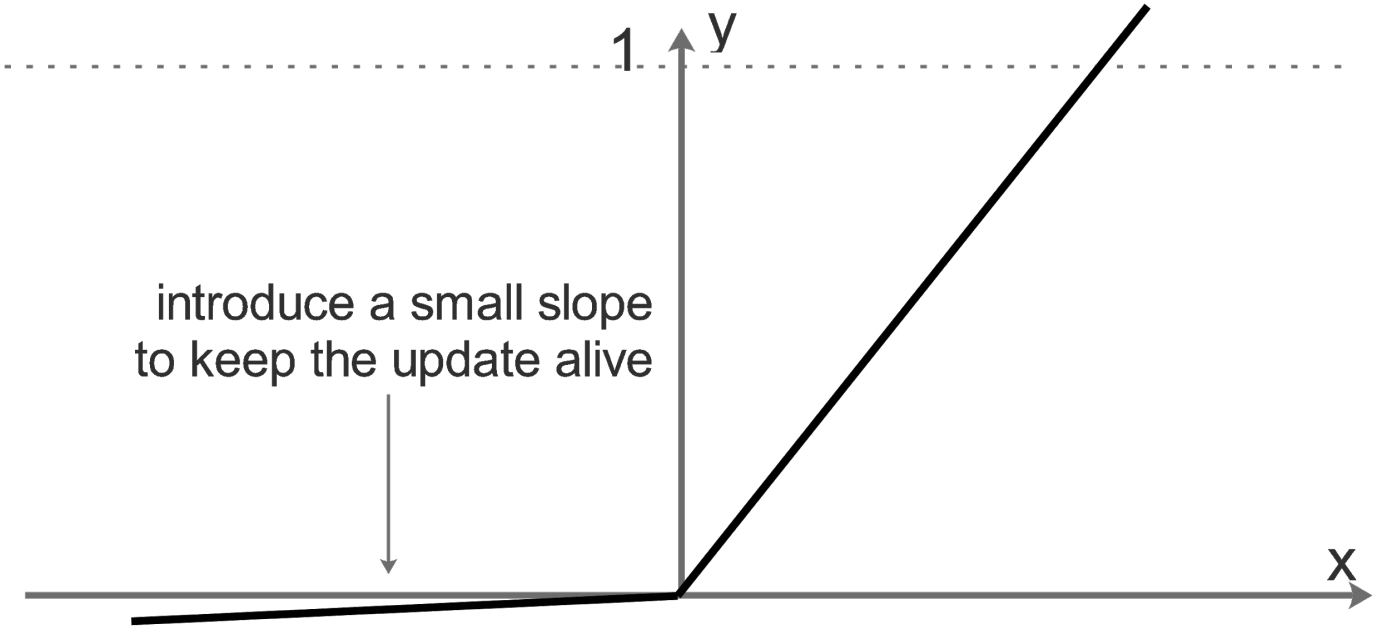


The Dying ReLU Problem

- Always outputs the same value (can't discriminate between inputs)
- Unlikely to recover once dead
- Why?
    a. Poor initialization or high learning rate

Activation Functions: **Leaky ReLU**

$$f(x) = \max(0.01x, x)$$



Questions?

Activation Functions: **Softmax**

- Used in the final layer of network
- Typically networks trained to optimize cross-entropy loss
- For multi-class classification
- Interpretable as probabilities, like sigmoid

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

Which one do I use?

Activation Functions in Practice

- Use ReLU. Be careful with your learning rate!
- Try Leaky ReLU and tanh.
- Don't use Sigmoid.*

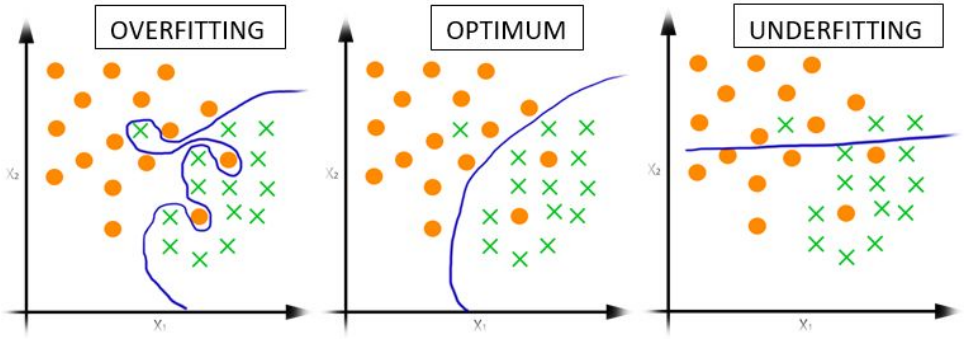
*between hidden layers; okay for output layer

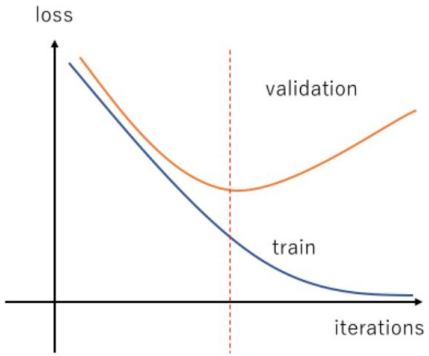
Questions?

Jupyter Exercises 2-4: Baseline NN in Keras

Overfitting

- Performance gap between train and test
  a. Difference in accuracy
  b. Model keeps getting better on train, but plateaus or gets worse on validation set





Regularization: **L2 Weight Decay**

- Artificially discourages complex relationships in data even if they fit training set well. The idea is that such relationships are unlikely to generalize well to the future (e.g. noise).
- Penalize loss function by adding a multiple of the L2 norm of your weight vector *w*. You get the following equation:

$$L(X, Y) + \lambda N(w)$$

Jupyter Exercise 5: L2 Regularization

Summary of Today

- Able to recreate COMPAS recidivism scores around 75% of the time, with a fairly simple deep neural network.
- Learned about activation functions, overfitting, and regularization.
- Learned how to use Keras.

Questions?

Bonus Slides

Regularization: **Dropout**

- Regularization technique where randomly selected neurons are ignored during training.
- Contributions during forward/backward pass are ignored.
- Network becomes less sensitive to the specific weights of neurons ➜ better generalization and less overfitting.

Jupyter Exercise 6: Dropout

Batch Normalization

- Used to help remedy “covariate shifts”, or changes in distribution of function’s domain.
- Keep activations in the zero-mean unit-variance range.

Jupyter Exercise 7: Batch Normalization