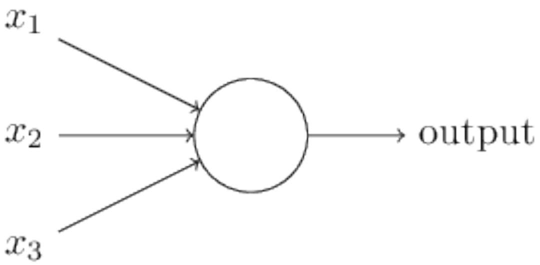
UNIVERSITY OF
SAN FRANCISCO
CHANGE THE WORLD FROM HERE

Neural Networks

Machine Learning

Historically...

- Early attempts to model brain (perceptron)
    - Inputs (x) to a function: 0, 1
    - Output (also 0, 1) depends on f(x) based on:
        - Internal weights
        - Internal threshold for activation

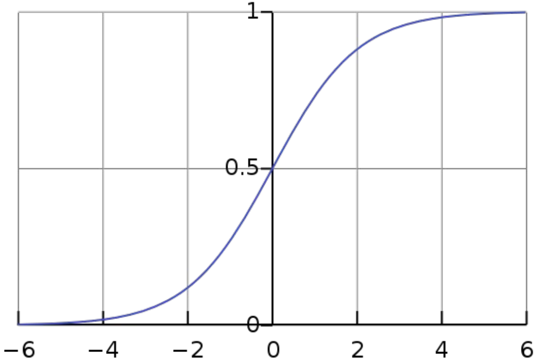


- Other attempts: understand Artificial Intelligence possibilities and limits
- Metaphors:
    - Deciding on what to eat at Bon Appetit (considering budget, preferences, etc.)
    - Computer circuits
- Many names for similar items
    - Basic: (multi-layer) perceptron, (artificial) neural network
    - Closely related: dendritic computation?

Perceptrons to Neurons

- Input is multiple vectors of Boolean propositions
- Output = 0 or 1:
  - $\Sigma_j w_j x_j$
  - Sum passed through an *activation function, f,* (eg. sigmoid)

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



  - Vector product assumed and threshold rewritten as bias (b), so:

$$\text{output} = f\,[\,(w \cdot x) + b\,]$$

- Training a perceptron involves setting weights
- Perceptron to neuron: input and output values do not have to be Boolean

Activation Functions

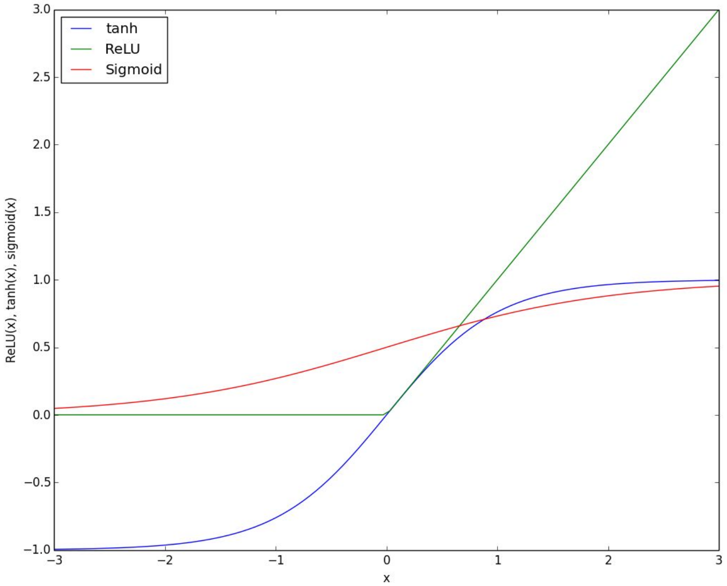
- Sigmoid may be the most common activation function
- Others:
  - Tanh

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

  - ReLU (“Rectifier” / “soft plus”)

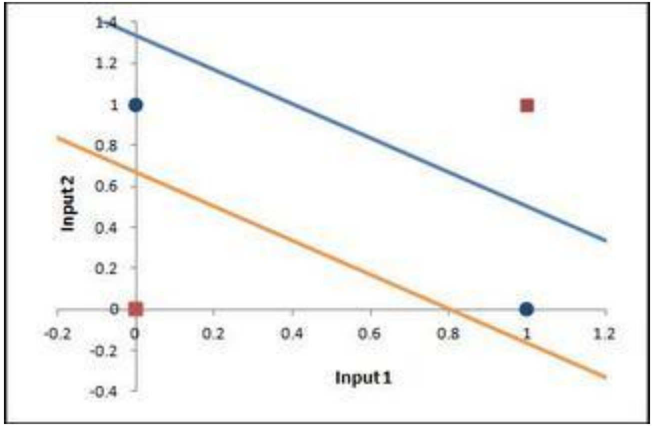
$$f(x) = \log(1 + \exp x)$$

  - Others (maxout, leaky, etc.)



The XOR problem

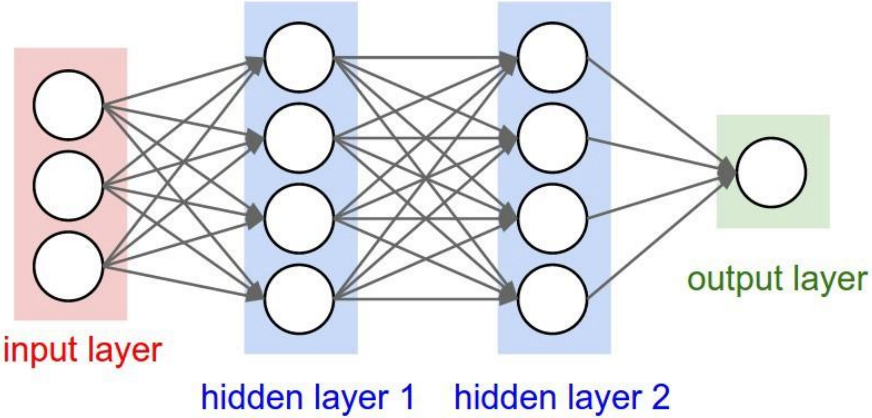
- Early problem in neural networks
- Consider the XOR circuit:
  - Two inputs: x_0, x_1
  - Output:
    - 1 if EITHER x_0 or x_1 has the value 1
    - 0 if x_0 and x_1 have the same value (either 1 or 0)
- Need multiple decisions to model this



http://toritris.weebly.com/perceptron-5-xor-how--why-neurons-work-together.html

Designing Networks

- A typical neural network contains
  - Input layer (exactly 1)
  - Hidden layer(s): 0, 1 or more layers
  - Output layer (exactly 1)
- Layer contents
  - Input: As many neurons as features
  - Output: As many neurons as classes*
  - Hidden: No limits
- Feed forward network
  - Output from one layer becomes input to next
  - Network is *fully connected*: all outputs from one layer are inputs to all neurons in next
  - Weights for each neuron provide differentiation among neurons



https://hackernoon.com/challenges-in-deep-learning-57bbf6e73bb

Advantages & Disadvantages

- Advantages
  - In theory: some network can learn any function
  - No need to derive features… just feed the network the “raw” signal
  - Unbelievable performance
- Disadvantages
  - No one REALLY understands how they work, so drive statisticians crazy
  - Require a lot of computational resources (GPU?) to train
  - Can overfit

A Neural Network Implementation

```
class Network(Classifier):

    def __init__(self, sizes, epochs, batch_size, learning_rate):
        self.num_layers = len(sizes)
        self.sizes = sizes            # Not shown: setting other class data

        # Randomly set biases for all non-input neurons
        self.biases = [np.random.randn(y, 1) for y in sizes[1:]]

        # Randomly set weights for each input/output pair
        self.weights = [np.random.randn(y, x) for x, y in
                zip(sizes[:-1], sizes[1:])]
```

- Example use:

```
dnn = Network([2, 2, 1])
```

Training: SGD, Backpropagation

- [Stochastic] gradient descent — recall:
    - Calculate *cost function* = some difference between targets and hypotheses
    - Cost function is often SSE
    - Gradient descent minimises cost function by (slowly) moving the decision boundary in the direction of the falling gradient
    - Stochastic: make changes with part of the training data (“batch”)
- Backpropagation (concept)
    - Propagate errors to all layers of the network for each batch
    - Each neuron updates its decision boundary

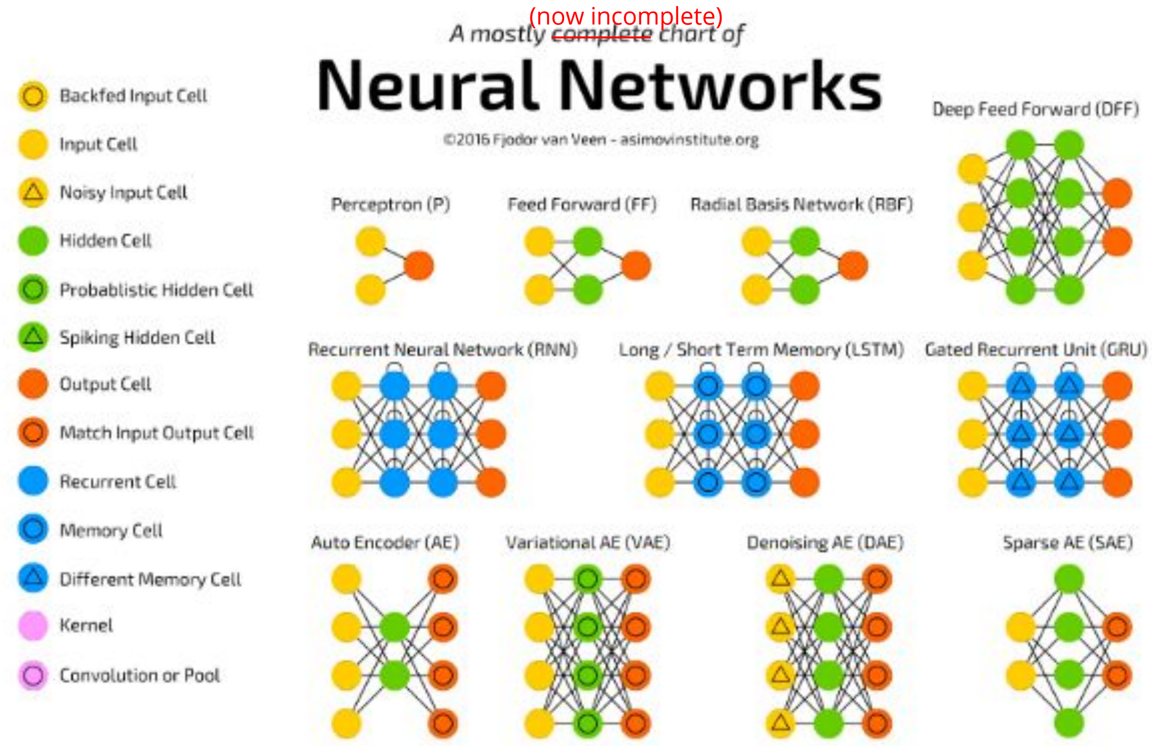
Backpropagation: Partial Code

```
def backprop(self, x, y):

    # Not shown = declaration of layer-by-layer errors
    # Not shown = call to get individual neuron activations & hypotheses
    delta = self.cost_detivative(activations[-1], y) *
sigmoid_prime(hyp[-1])
    bias_error[-1] = delta
    weight_error[-1] = np.dot(delta, activations[-2].transpose())

    # Step backward in the network and spread errors to all layers (not
input)
    for layer in xrange(2, self.num_layers):
        z = zs[-layer]
        sigmoid_prime = sigmoid_prime(z)
        bias_error[-layer] = delta
        weight_error[-layer] =
np.dot(delta,activations[-layer-1].transpose())
    return (bias_error, weight_error)
```

Hands-on (Practical)

- Download MNIST dataset from Kaggle
  - https://www.kaggle.com/c/digit-recognizer/data
  - Only need “train.csv” since “test” does not have labels
- Recall model performance from earlier SVM experiments
  - Test set is final 100 items
  - Expected SVM accuracy: 0.78 <= x <= 0.91 (changes with random seed)
- Design a neural network for this
  - Try the following:
    ```
    from sklearn.neural_network import MLPClassifier
    mlp = MLPClassifier(hidden_layer_sizes=(100, 10))
    ```
  - Which items did it get wrong? (Are they also hard for you to determine?
  - Experiment with parameters to raise performance

Going Deeper



From "Up to Speed on Deep Learning: September, Part 2 and October, Part 1"

Tools for Larger Networks

Keras

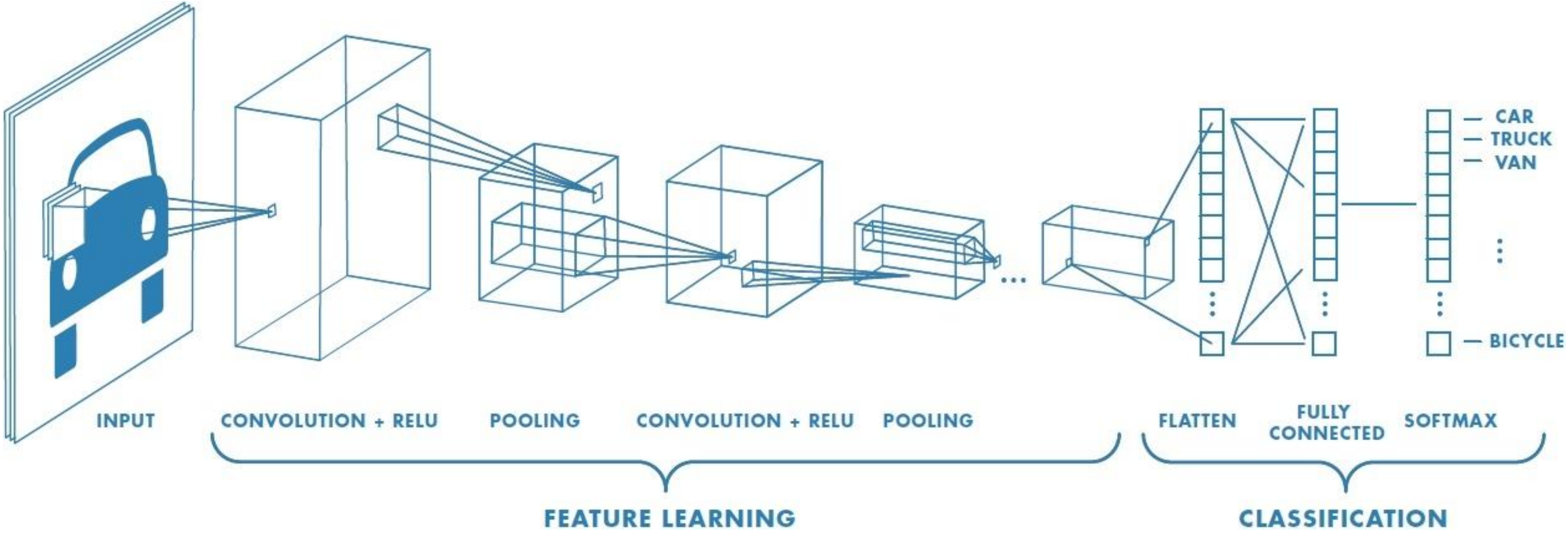
PYTORCH

TensorFlow

theano

torch

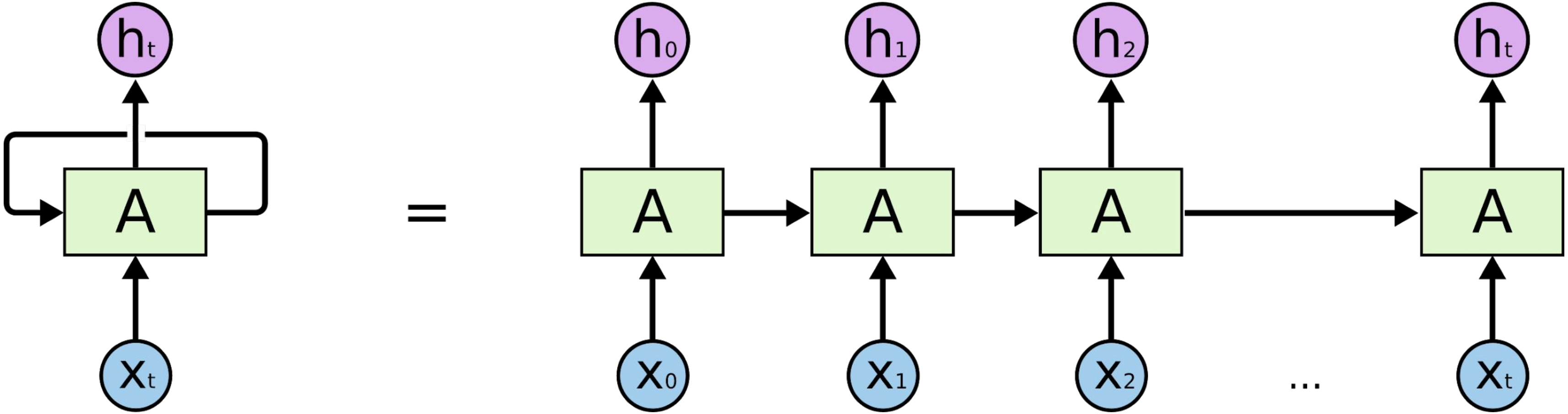
Convolutional Neural Networks



https://www.mathworks.com/discovery/convolutional-neural-network.html

Long Short Term Memory (LSTM)

- Output of a layer becomes (additional) input



- Good for language, speech, video

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Odds and Ends

- Dropout
  - Neurons in the network randomly select some input weights to be 0
  - Forces network to learn more robust features
  - Reduces overfitting — similar to Random Forest, Boosting, etc.
- GPUs are standard equipment for neural networks
  - For each epoch, each connection between two neurons is a matrix calculation
  - CPUs are not optimised to run many tiny calculations
- Data determines outcomes
  - A neural network (any ML algorithm) will only be as good as the data it’s trained on
  - Collect data with care!

Next Time

- Thursday:
  - Classifying smile types (Leon Wang)
  - Classification Summary & Review
- Next week: Clustering (Paul Intrevado)
- Midterm II