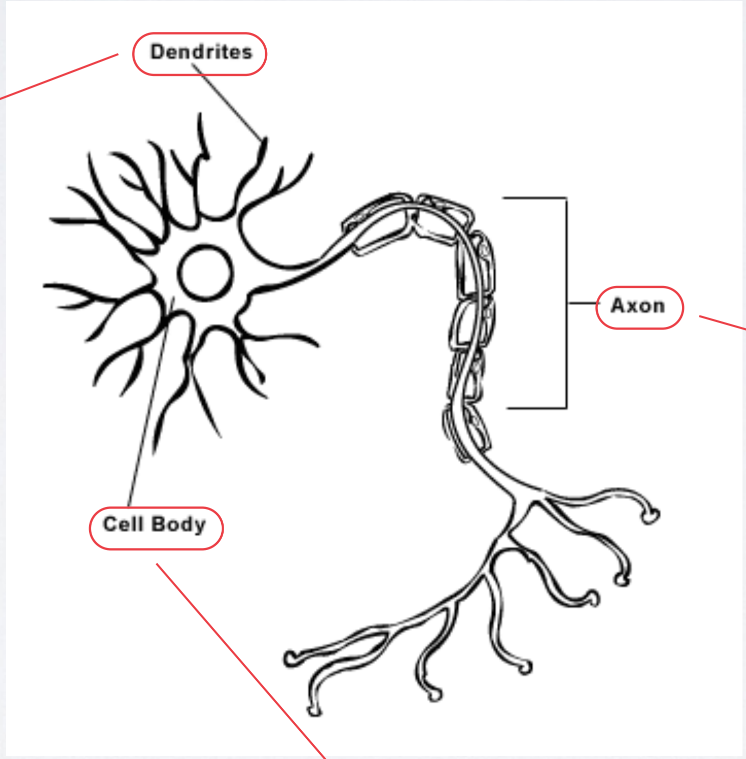
NEURAL NETWORKS

Charlie Bonfield
ASTR 503/703
October 31st, 2016

MAPPING COMPUTATIONAL PROBLEMS WITH NEURONS

How does the human brain process information?

Dendrites receive information and carry it towards the cell body (**input**).

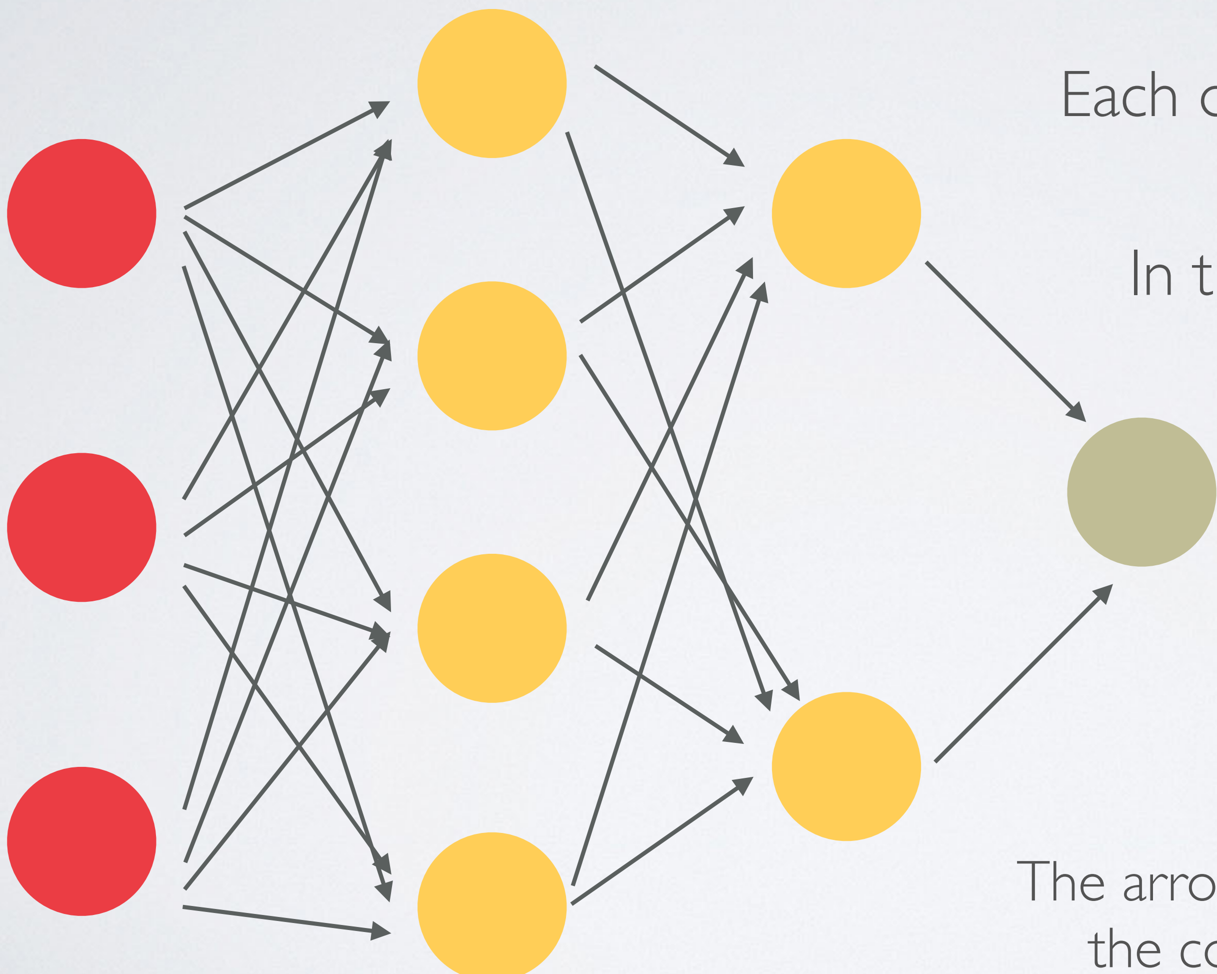


Axons carry information sent from cell body to be transmitted to other neurons (**output**).

Cell body processes input from dendrites (**processing**).

TNC

MAPPING COMPUTATIONAL PROBLEMS WITH NEURONS



Artificial Neural Network

Each dot represents a neuron.

In the business, these can be referred to as "units".

- Red: Input units
- Yellow: Hidden units
- Green: Output units

The arrows between the units denote the connections between them.

BASIC APPLICATIONS OF NEURAL NETWORKS

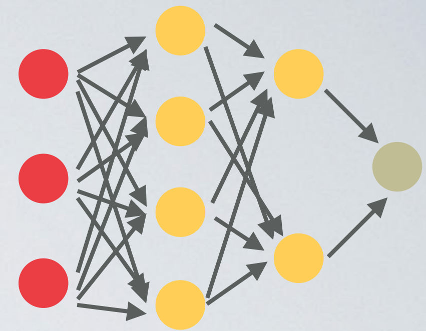
Neural networks are good at identifying trends and picking up on patterns that may exist in a set of data, making them applicable in a wide array of fields.

- Pattern recognition (numbers, letters, faces, etc.)
- Time series analysis (weather patterns, stock prices)
- Signal processing (filter out noise, deliver signal)
- Control (self-driving cars!)
- Soft sensors (data fusion)
- Anomaly detection (deviations from standard behavior)

Often times, neural networks perform tasks that can be best described as "easy for a human, difficult for a machine".

NEURAL NETWORKS

(QUICK FACTS)

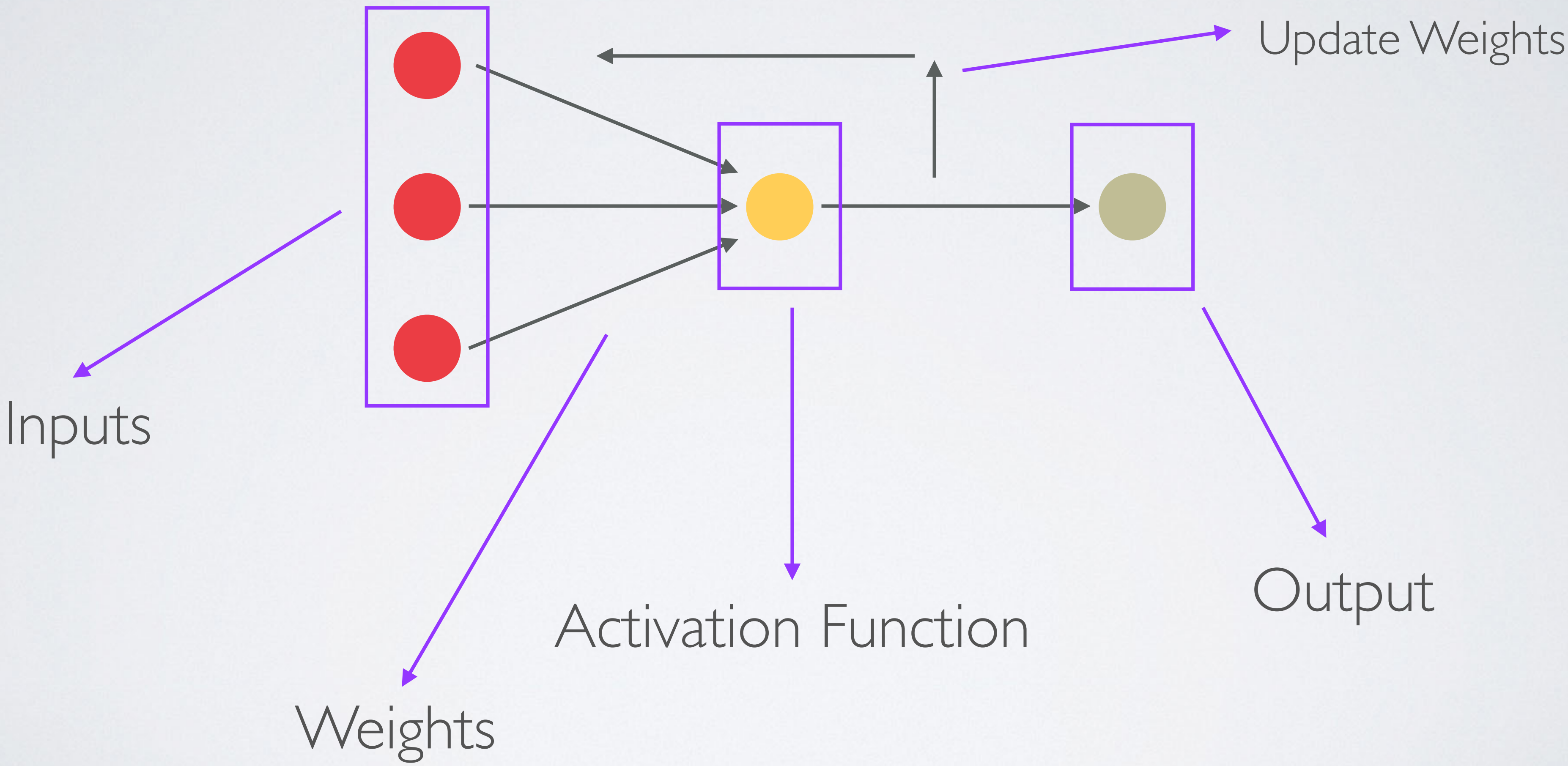


- Like many of the other techniques that we have discussed, neural networks require a *training* set and a *test* set.
- There are many different neural network structures, the simplest of which is called the *multilayer perceptron (MLP).* Others include radial basis function networks (RBF), wavelet neural networks, and self-organizing maps.
- Neural networks may be used for both *supervised* and *unsupervised* learning. (We'll talk about supervised learning, but members of the class are encouraged to explore clustering algorithms such as self-organizing maps (SOM) and adaptive resonance theory (ART).)
- Neural networks can be used for both *classification* and *regression* problems (classification will be the focus of this presentation).

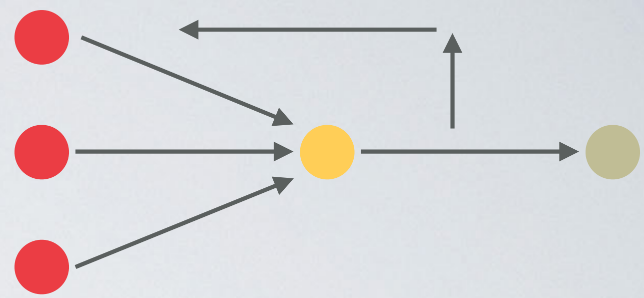
ZNN

PERCEPTRONS

The most fundamental component of an MLP is a single *perceptron*.



PERCEPTRONS



- **Inputs**: data (commonly referred to as features)

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_n \end{bmatrix}$$

- **Weights**: weight associated with each feature

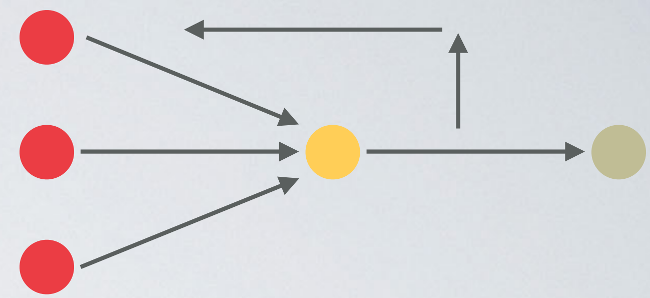
$$\mathbf{w} = \begin{bmatrix} w_1 & w_2 & w_3 & \cdots & w_n \end{bmatrix}$$

- **Activation Function**: determines how we classify our object based on the input features and weights

$$z = x_1 w_1 + x_2 w_2 + x_3 w_3 + \cdots + x_n w_n = \mathbf{w}^T \mathbf{x}$$

Choice of activation function for two linearly separable classes $\longrightarrow$ $\phi(z) = \begin{cases} 1 & z \geq \theta \\ -1 & z \leq \theta \end{cases}$

PERCEPTRONS



- **Update Weights**: use successes/failures to update weights

$$w_i = w_i + \Delta w_i$$

(cost function is sort of wrapped up in here - more on that later!)

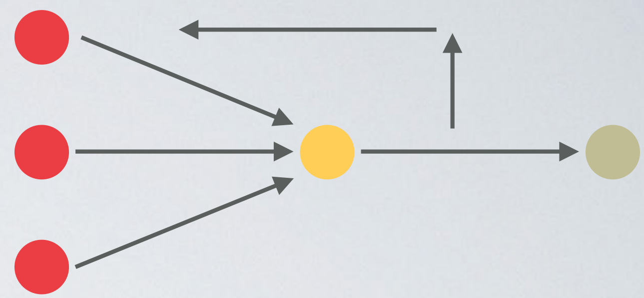
$$\Delta w_j = \eta \left(y^{(i)} - \hat{y}^{(i)} \right) x_j^{(i)}$$

learning rate actual class predicted class

- **Output**: set of final classifications (occurs once we have stepped through the specified number of updates)

Done!

PERCEPTRONS



```
import numpy as np


class Perceptron(object):
    """Perceptron classifier.

    Parameters
    ------------
    eta : float
        Learning rate (between 0.0 and 1.0)
    n_iter : int
        Passes over the training dataset.

    Attributes
    -----------
    w_ : 1d-array
        Weights after fitting.
    errors_ : list
        Number of misclassifications in every epoch.

    """
    def __init__(self, eta=0.01, n_iter=10):
        self.eta = eta
        self.n_iter = n_iter

    def fit(self, X, y):
        """Fit training data.

        Parameters
        ----------
        X : {array-like}, shape = [n_samples, n_features]
            Training vectors, where n_samples is the number of samples an
d
            n_features is the number of features.
        y : array-like, shape = [n_samples]
            Target values.

        Returns
        -------
        self : object

        """
        self.w_ = np.zeros(1 + X.shape[1])
        self.errors_ = []

        for _ in range(self.n_iter):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w_[1:] += update * xi
                self.w_[0] += update
                errors += int(update != 0.0)
            self.errors_.append(errors)
        return self

    def net_input(self, X):
        """Calculate net input"""
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):
        """Return class label after unit step"""
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```

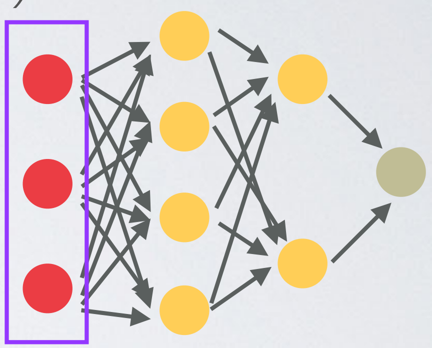
PML

BUILDING A NEURAL NETWORK

Perceptrons may be extended to far more complex models.

Fixed: number of inputs, number of hidden units/layers, data, activation functions

Variable: weights (these need to be learned for classification!)



Data: $$\mathbf{x} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix}$$

m : number of examples

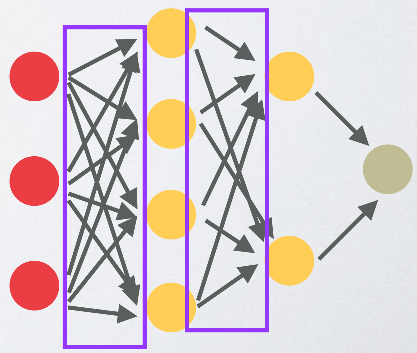
n : number of features per example

p : hidden units (1)

q : hidden units (2)

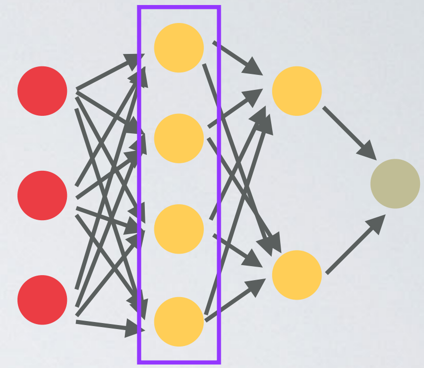
Weights: $$\mathbf{w}^{(1)} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1p} \\ w_{21} & w_{22} & \dots & w_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \dots & w_{np} \end{bmatrix}$$

$$\mathbf{w}^{(2)} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1q} \\ w_{21} & w_{22} & \dots & w_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ w_{p1} & w_{p2} & \dots & w_{nq} \end{bmatrix}$$



BUILDING A NEURAL NETWORK

Activity (Hidden Layer 1): $\mathbf{z}^{(1)} = \mathbf{x}\mathbf{w}^{(1)}$

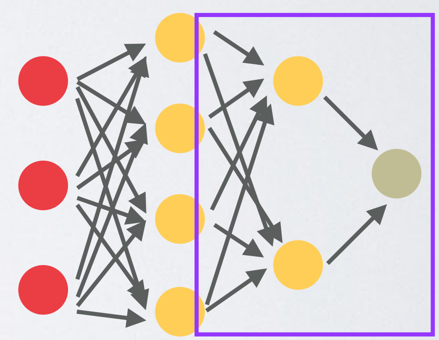


Apply activation function at each hidden unit:

$$\mathbf{f}\left(\mathbf{z}^{(1)}\right) = \phi\left(\mathbf{z}^{(1)}\right)$$

Feed result through next set of weights…

Activity (Hidden Layer 2): $\mathbf{z}^{(2)} = \mathbf{f}\left(\mathbf{z}^{1}\right)\mathbf{w}^{(2)}$



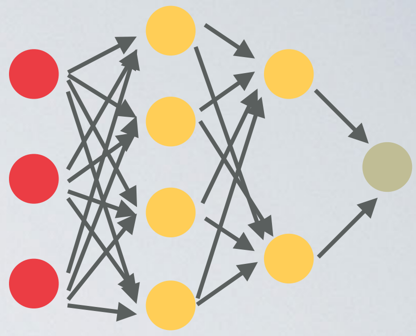
Apply activation function at each hidden unit:

$$g\left(\mathbf{z}^{(2)}\right) = \phi\left(\mathbf{z}^{(2)}\right)$$

Feed result through next set of weights…

Same game… $\mathbf{z}^{(3)} = \mathbf{g}\left(\mathbf{z}^{(2)}\right)\mathbf{w}^{(3)} \longrightarrow \phi\left(\mathbf{z}^{(3)}\right)$

BUILDING A NEURAL NETWORK



How do we assess the accuracy of our network?

What metric do we use to update our weights?

Cost Function: minimize in order to find the optimal weights

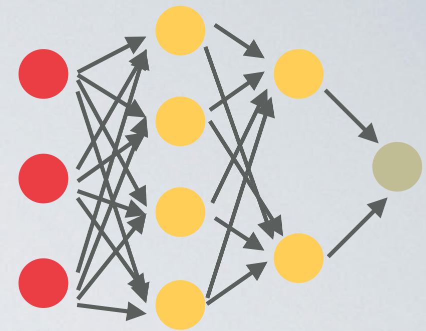
$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}(y_i - \phi(z_i))^2$$

This is the activation function evaluated using the activity found after our final set of weights has been applied.

Thoughts on how to proceed?

WLNN, PML

BUILDING A NEURAL NETWORK



We can use *gradient descent* to shift our weights in the direction that minimizes the cost function.

There are better choices than gradient descent (BFGS), but this serves for illustrative purposes.

$$\frac{\partial J}{\partial \mathbf{w}^{(3)}} = -\left[\mathbf{g}\left(\mathbf{z}^{(2)}\right)\right]^T (y - \hat{y})\, \phi'(\mathbf{z}^{(3)})$$

$$\frac{\partial J}{\partial \mathbf{w}^{(2)}} = -\left[\mathbf{f}\left(\mathbf{z}^{(1)}\right)\right]^T (y - \hat{y})\, \phi'(\mathbf{z}^{(3)}) \left(\mathbf{w}^{(2)}\right)^T \phi'(\mathbf{z}^{(2)})$$

$$\frac{\partial J}{\partial \mathbf{w}^{(1)}} = ?$$

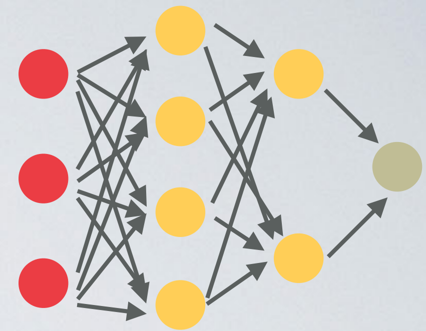
I leave this as an exercise to the class! It also probably wouldn't hurt to check my math on the previous two lines.

Bottom Line: We can minimize our cost function, thereby allowing us to make an informed decision about how to change our weights (*backpropagation*).

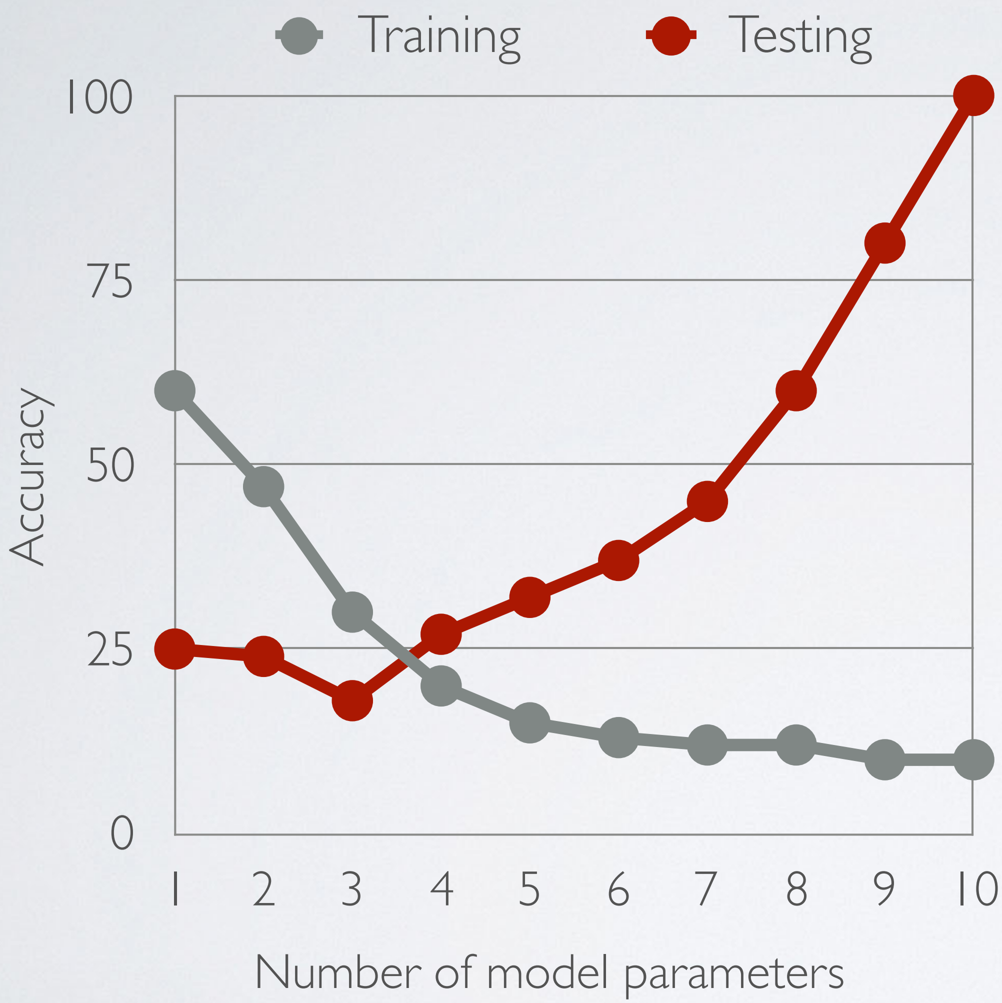
Example of MLP: refer to http://scikit-learn.org/stable/modules/neural_networks_supervised.html

WLNN, PML

BUILDING A NEURAL NETWORK

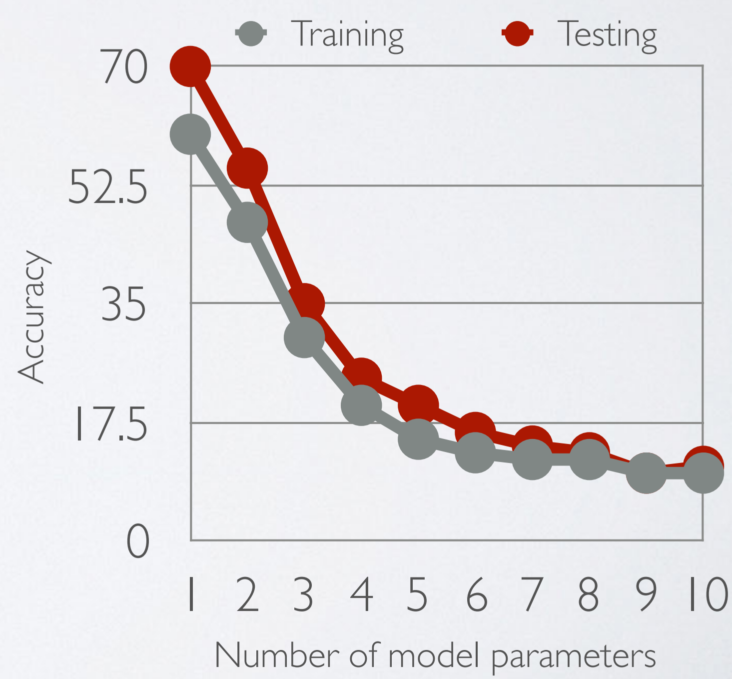


Once we have our result, we have to look out for *overfitting*.



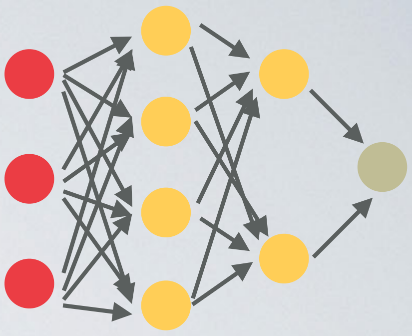
Solutions:

- More data!
- *Regularization*: penalize overly complex models by adding an extra term to cost function



WLNN, PML

BUILDING A NEURAL NETWORK

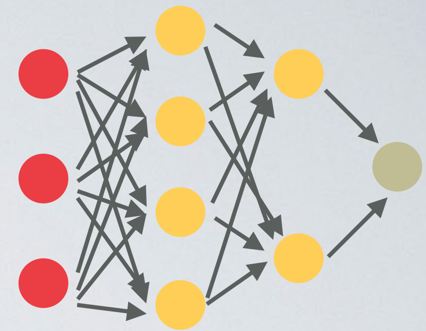


Basic Ingredients for MLP:

1. Data (examples, features)
2. Number of hidden layers
3. Number of units per layer
4. Activation function
5. Cost function (w/ regularization)
6. Smart way to take derivatives

How to determine these?

BUILDING A NEURAL NETWORK



Basic Ingredients for MLP:

1. Data (examples, features)
2. Number of hidden layers
3. Number of units per layer
4. Activation function
5. Cost function (w/ regularization)
6. Smart way to take derivatives

How to determine these?

Short answer: still an open problem!

- There is no clear-cut way of determining these quantities.
- However, one might surmise that dimensionality and the degree of nonlinearity plays a role.
- Potential solutions: trial and error, dynamically changing the number of neurons while training the network, etc.

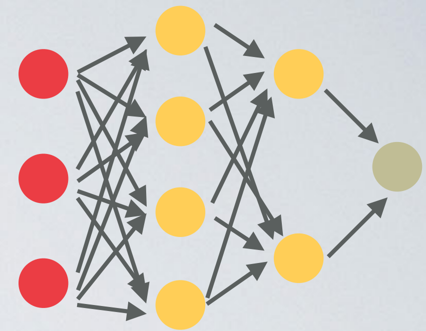
ZNN, EBP

CLASSIFIERS

TABLE 9.1.
Summary of the practical properties of different classifiers.

Method	Accuracy	Interpretability	Simplicity	Speed
Naive Bayes classifier	L	H	H	H
Mixture Bayes classifier	M	H	H	M
Kernel discriminant analysis	H	H	H	M
Neural networks	H	L	L	M
Logistic regression	L	M	H	M
Support vector machines: linear	L	M	M	M
Support vector machines: kernelized	H	L	L	L
K-nearest-neighbor	H	H	H	M
Decision trees	M	H	H	M
Random forests	H	M	M	M
Boosting	H	L	L	L

ACCURACY

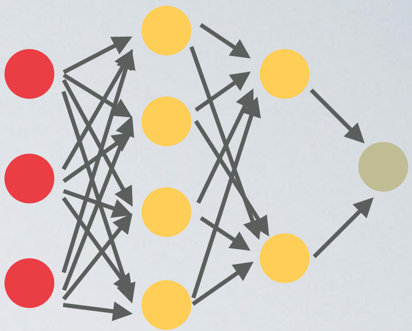


How well can it make accurate predictions or model data?

- Neural networks is an example of a *nonparametric* method.
- Since the number of parameters grows as the number of data points grows, it is reasonable to expect nonparametric methods to perform better than their parametric methods.
- To truly ensure that our neural network is parametric, we would need to incorporate a way of allowing the number of hidden layers/units to vary as we change our sample size.

Regardless, I think the lesson to be learned is that one does not know which method will be most accurate until one tries them all.

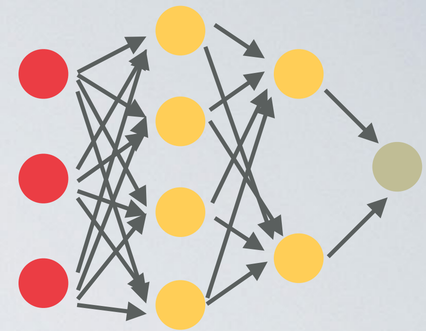
INTERPRETABILITY



How easy is it to understand why the model is making the predictions that it does, or reason about its behavior?

- As you may have surmised, neural networks rate poorly when it comes to interpretability.
- In my opinion, there are many different questions you could ask regarding this issue:
  - If we are only after the results, how much do we care about what goes on underneath the hood?
  - Are we able to leverage whatever knowledge we have about our problem through the initial weights and/or choice of activation function?

SIMPLICITY

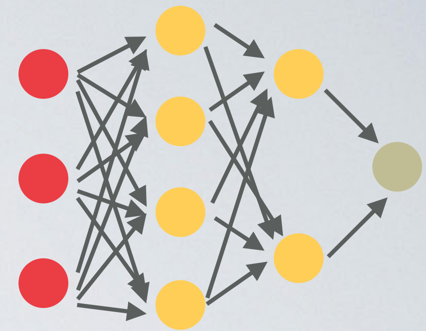


Is the model "fiddly", that is, does it have numerous parameters that must be tuned and tweaked with manual effort? Is it difficult to program?

- The authors rated neural networks poorly here as well, but I would argue that things are not as bad as they seem.
- Once you initialize your weights (which can be done with a random number generator), you do not need to provide any further input during backpropagation.
- One potential pitfall with neural networks in this context pertains to the number of hidden layers/units.

Neural networks may be difficult to program, but they appear to require very little user input thereafter.

SPEED



Is the method fast, or is it possible via sophisticated algorithms to make it fast without altering its accuracy or other properties?

- Neural networks require O(*N*log*N*) time to build and O(log*N*) time to classify.
- Part of code that can significantly decrease speed: gradient descent and/or other method for optimizing weights
  - Optimal number of hidden layers/units?
- Possible topic to explore: performance based on number of features

NEURAL NETWORKS IN ASTRONOMY

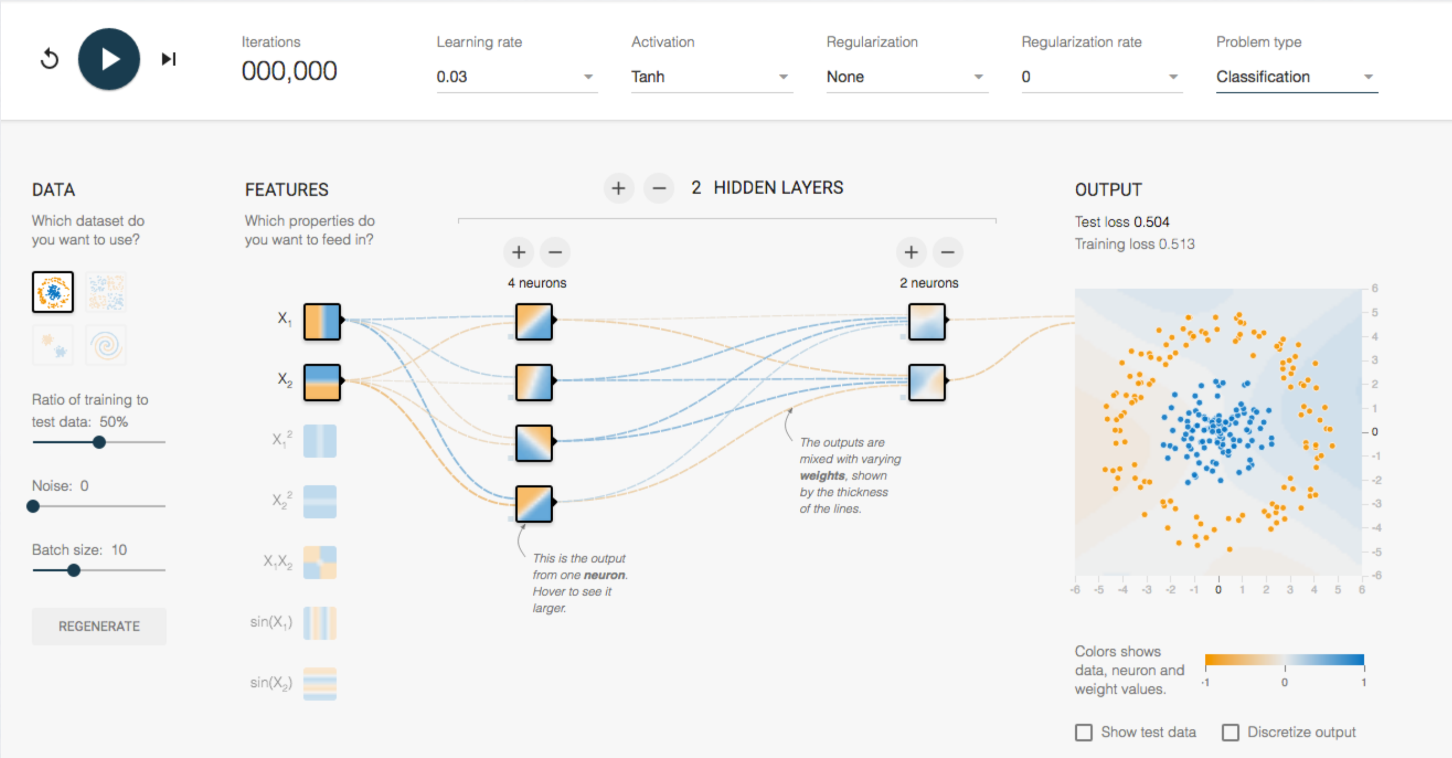
- Object identification (picking out sources from noise)
- Astronomical object classification (stars, galaxies, etc.)
  - Includes both spectral/morphological classification.
- Photometric redshifts (using spectroscopic data and/or predicted redshifts from spectral synthesis models)
- Time series analysis

In general, good for problems with high dimensionality - any ideas?

RTNN, CAA

GET YOUR HANDS DIRTY!

playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.54769&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false



REFERENCES

- Ciaramella, A. et al. *Applications of Neural Networks in Astronomy and Astroparticle Physics*. (CAA)
- Ivezic, Z. et al. *Statistics, Data Mining, and Machine Learning in Astronomy*. (SDML)
- LeCun, Y. et al. *Efficient BackProp*. (EBP)
- Nielsen, M. *Neural Networks and Deep Learning.* (MN)
- Raschka, Sebastian. *Python Machine Learning.* (PML)
- Shiffman, Daniel. *The Nature of Code*. (TNC)
- Tagliaferri, R. et al. *Neural Networks in Astronomy*. (RTNN)
- Welch Labs. *Neural Networks Demystified.* (YouTube sequence of videos) (WLNN)
- http://www.ieee.cz/knihovna/Zhang/Zhang100-ch03.pdf (Looks like a chapter of a book, but I could not find the full text online.) (ZNN)
- http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html

Pictures/Figures*

[2] http://www.explainthatstuff.com/introduction-to-neural-networks.html
[9] Code taken from PML (p. 25-26)
[14] Ivezic, Z. et al. (p. 400)

*Slide on which figure appears corresponds to number at left.