MACHINE LEARNING IN PYTHON

FROM 1 TO 80 REAL QUICK

BY COADBROS, 2018

JOSIAH COAD

INTRODUCTION

- What you'll walk away from here with (from past students)
  - How the basic of neural nets work
  - Get your hands dirty with some of the software tools in data science and machine learning
  - A rough guide to how machine learning works
  - How to implement neural networks in Python
  - Forward-Backward propagation
  - "I learned enough to begin practicing to set up a basic neural network."
  - Getting an overall understanding of what machine learning is and how it works

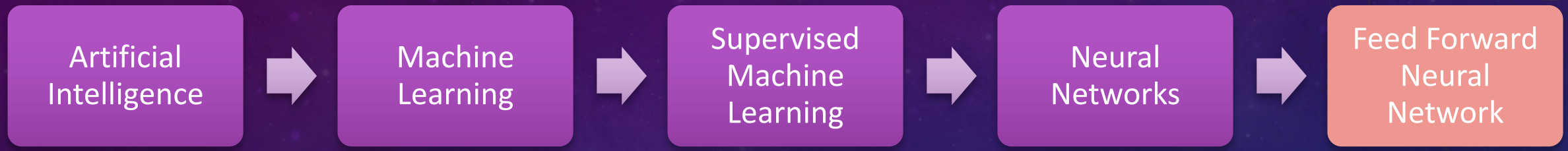
MACHINE LEARNING

- “Machine learning is based on algorithms that can learn from data without relying on rules-based programming.”- McKinsey & Co.

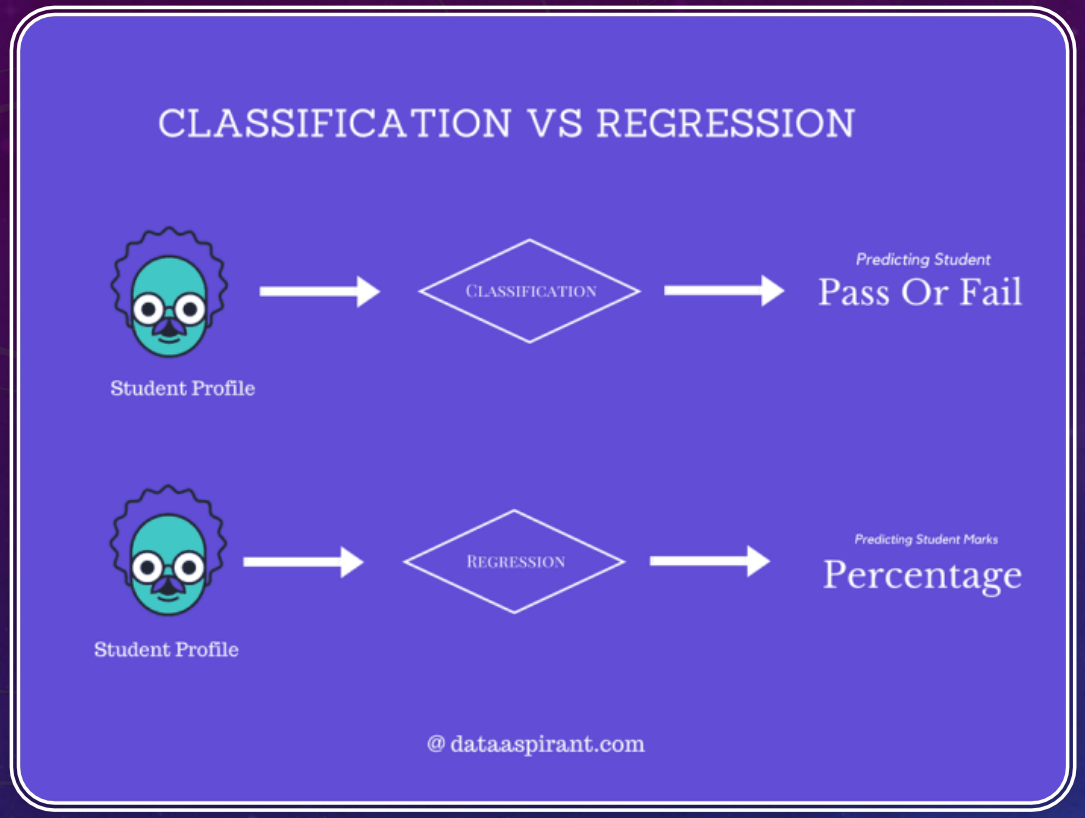
```
Algorithm 1 – Rule Based, Classical AI

def predict_grade(hrs_slept, hrs_studied):
    if hrs_slept > 7 and hrs_studied > 10:
        return “A”
    elif hrs_slept > 5 and hrs_studied > 10:
        return “B”
    elif hrs_slept > 5 and hrs_studied > 5:
        return “C”
    …
```

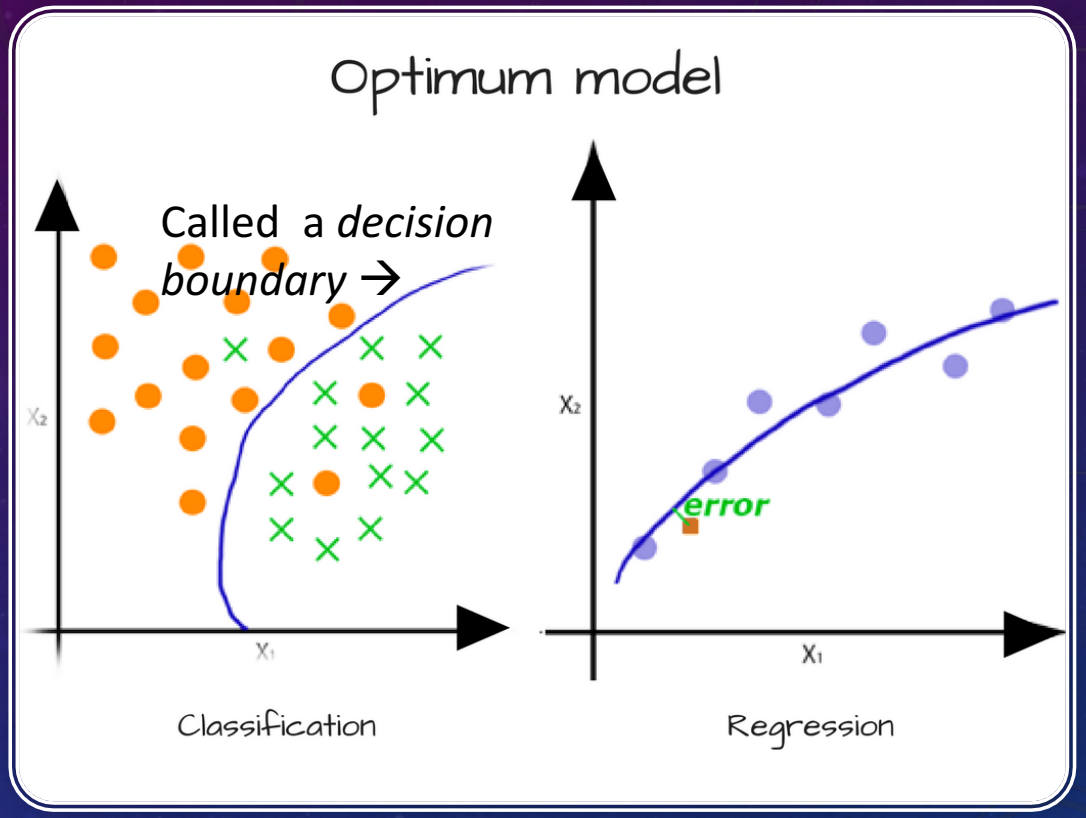
```
Algorithm 2 – Machine Learning Based
past_data =
[(8, 10, “A”), (5, 10, “B”), (6, 12, “B”) …]

def predict_grade(past_data, hrs_slept , hrs_studied):
    threshold_slept = randomint()
    threshold_studied = randomint()
    # iterate through your past_data, trying to
    # find the perfect thresholds that most
    # accurately predicts your outcome grade.
    # finally, predict the grade given the threshold
    # AKA parameters that you learned.
```

WHERE DOES THIS LECUTRE FIT IN TO THE BIG PICTURE?



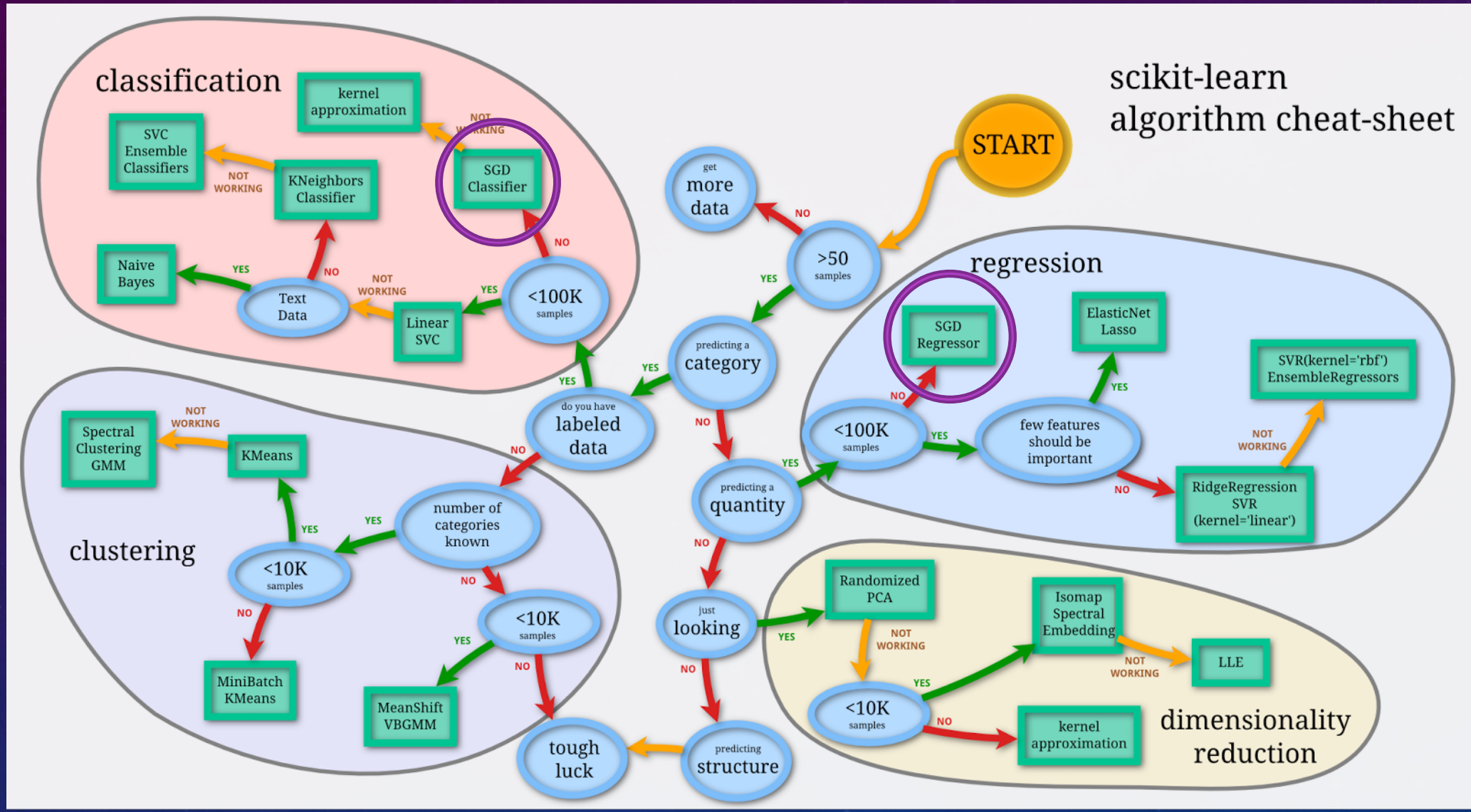
- **Artificial Intelligence** is a branch of computer science dealing with the simulation of intelligent behavior in computers
- **Machine learning** is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use to learn for themselves.
- **Supervised learning** is the task of learning from labeled training data. The labeled training data is used to fine tune the model to make more accurate predictions by minimalizing error.
- **A Densely Connected Feed Forward Neural Network** is one type of supervised machine learning that we will learn about in this class used to predict values and classify data. More on this in the slides to come!





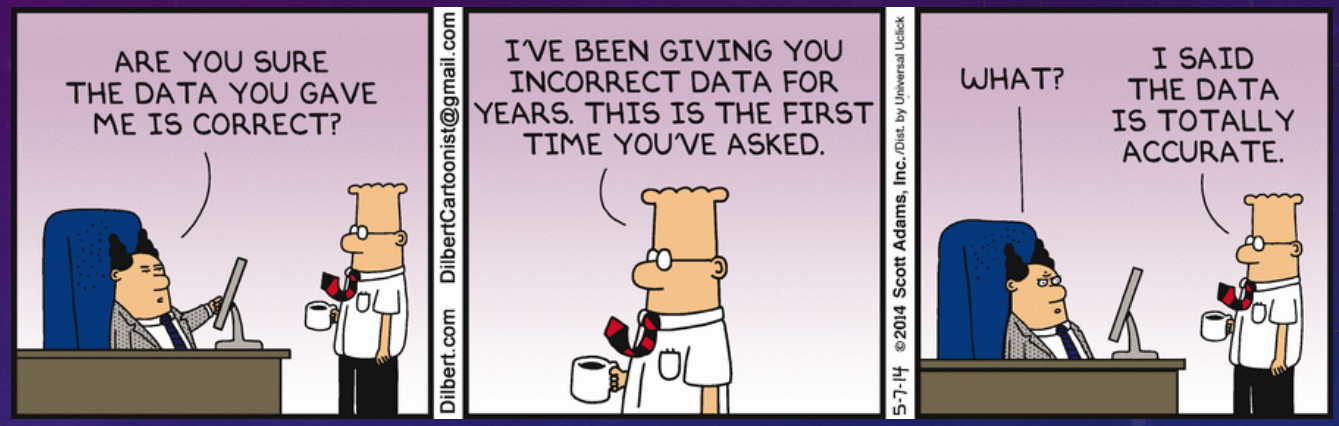
CLASSIFICATION VERSUS REGRESSION

When to Use a Neural Network? (Here called a SGD Classifier/Regressor)

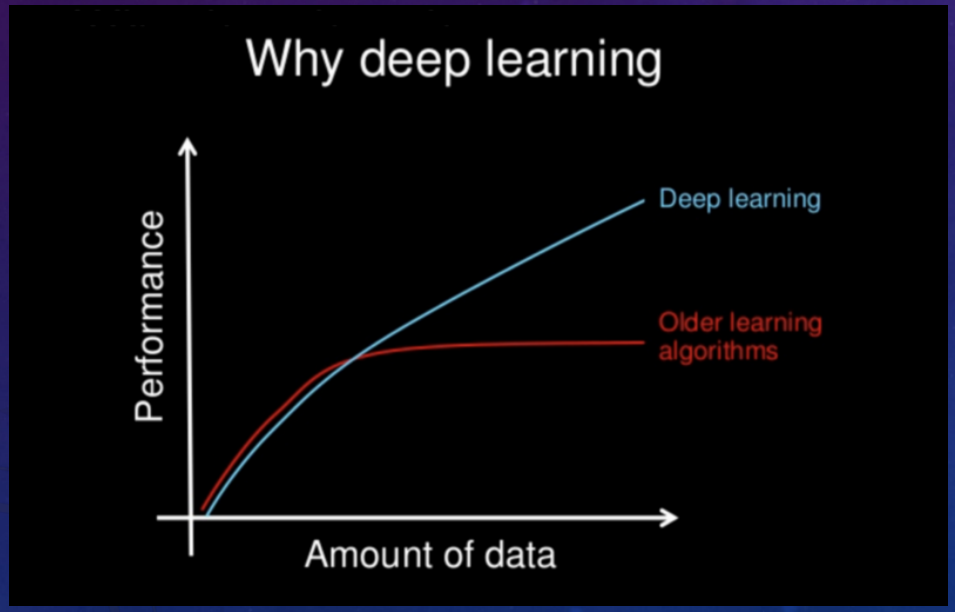


WHEN TO USE A NEURAL NETWORK (CONT)

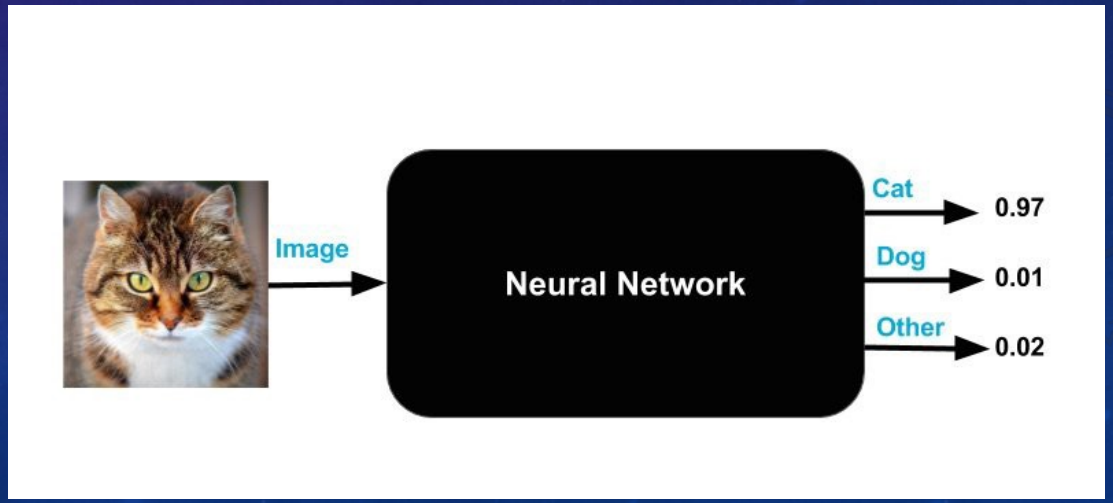
You've *still* have got to understand your data!



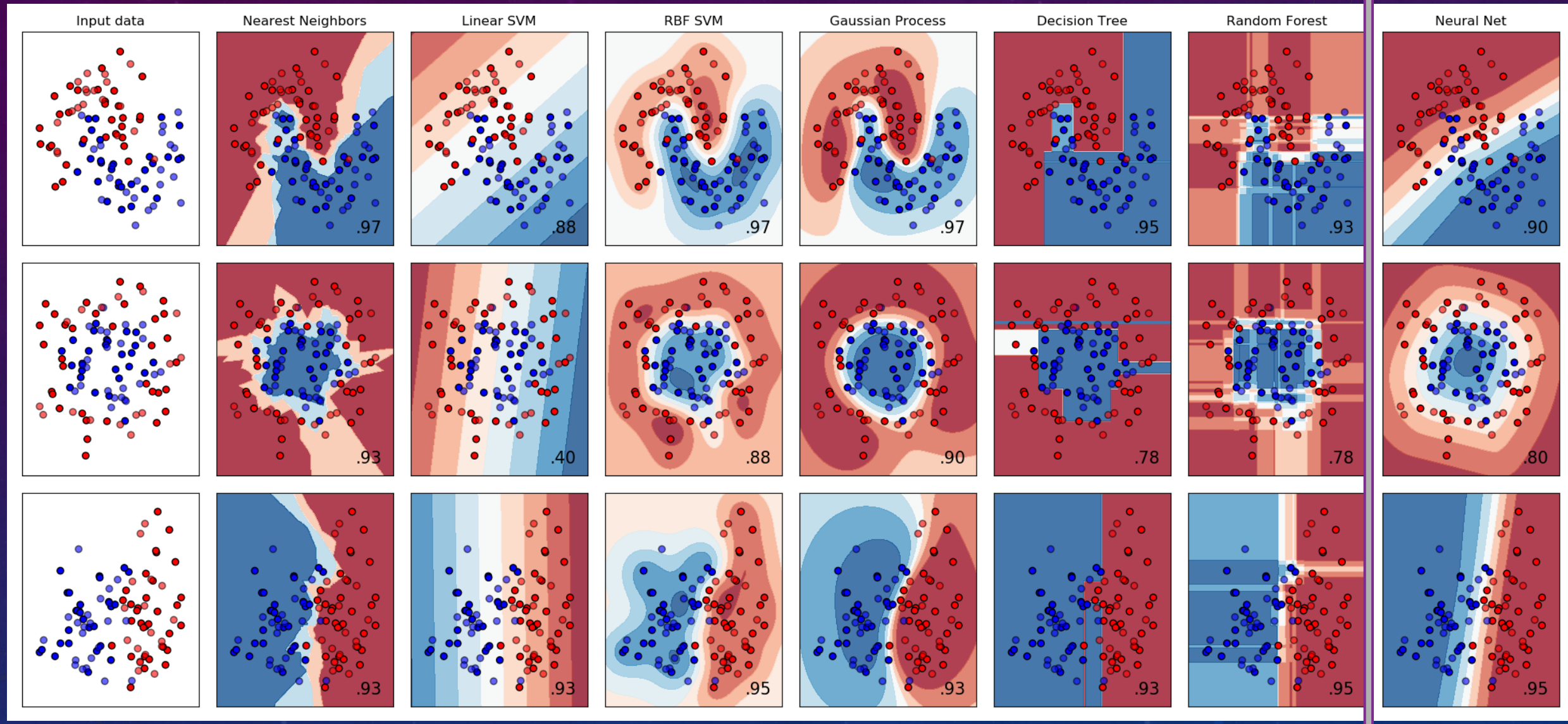
Lots of data (>100K* for good results)

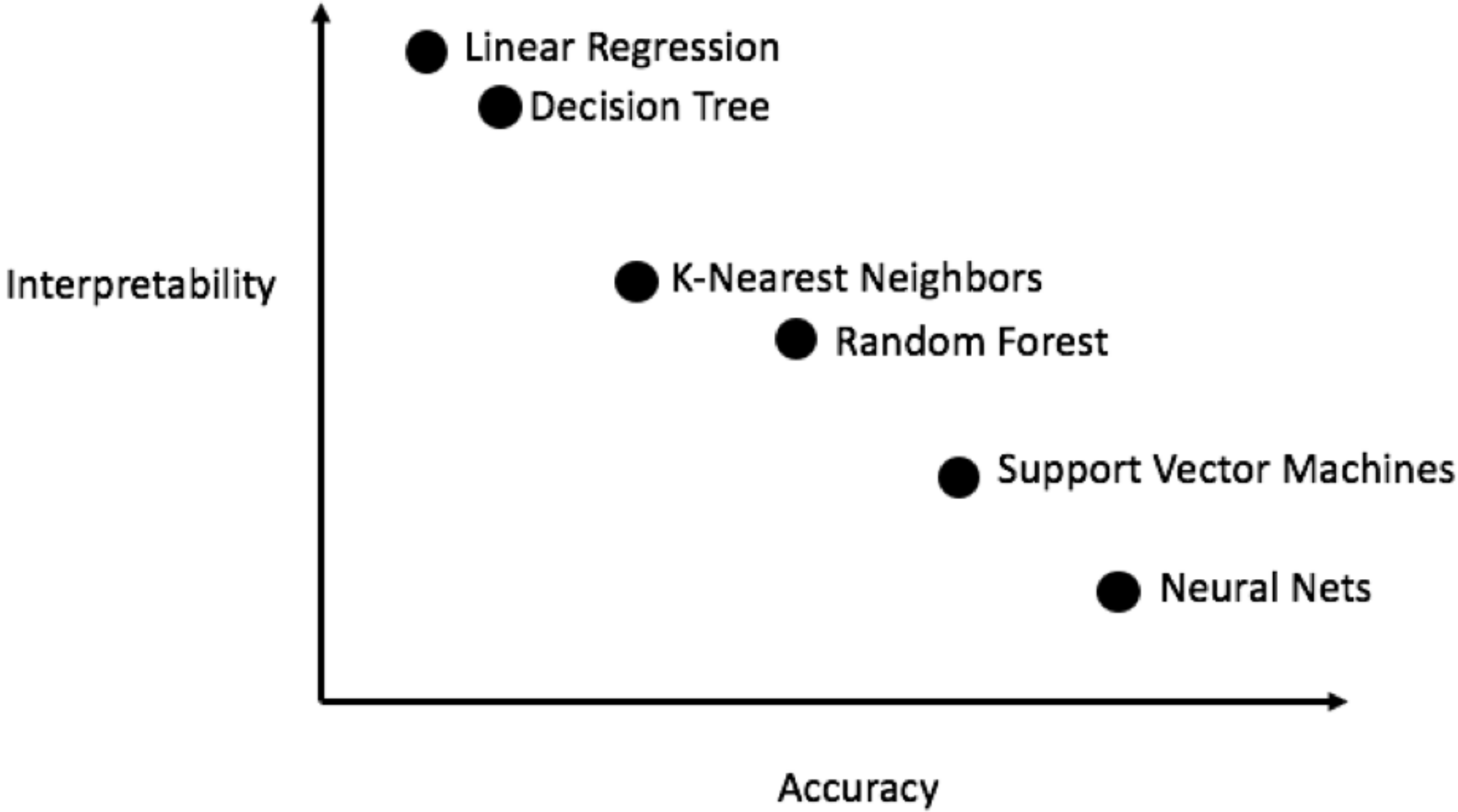


Black Box: Why did it do that?



How different models make *decision boundaries* for classification.




Linear Regression
Decision Tree
K-Nearest Neighbors
Random Forest
Support Vector Machines
Neural Nets
Interpretability
Accuracy

INTRODUCTION RECAP

- **Ockham's Razor:** Simplest Is best. Always use the simplest method you can find – this stuff can get complicated!
- Feed Forward Fully Connected Neural Networks are a very specific type of machine learning model which can be used for classification (predicting classes) and regression (predicting values)
- Trade-off between (complex) predictive ability and (simple) interpretability. Neural Networks are highly accurate but have very low interpretability.
- Machine learning discovers complex patterns in your data by a trial-error iterative process. (For a neural networks, this trial-error process is called forward and backward propagation)

CHAPTER 1:
HOW A NEURAL NETWORK MAKES A (NON-LINEAR) PREDICTION

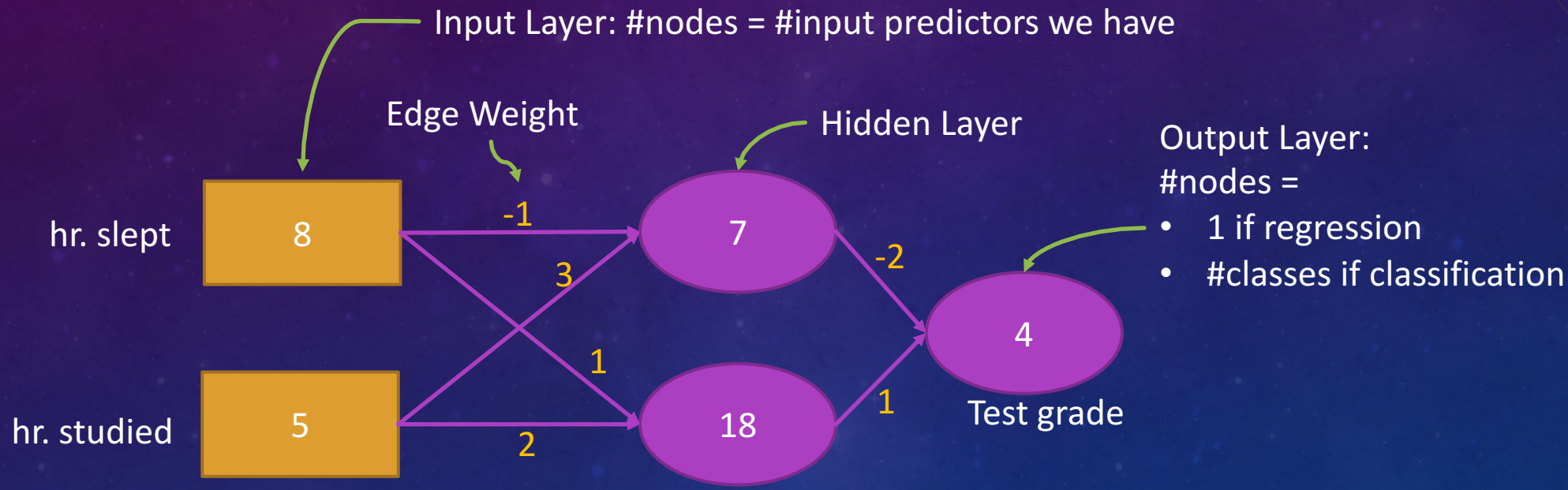
Key Terms

- Edge Weight
- Node / Multilayer-perceptron
- Hidden Layers
- Input Layer, Output Layer
- Forward Propagation
- Activation Functions
- Fully Connected, Feed Forward

```
sklearn.neural_network.MLPRegressor(
    hidden_layer_sizes=(100, ),
    activation='relu',
    solver='sgd',
    batch_size=10,
    learning_rate_init=0.001,
    early_stopping=False,
    validation_fraction=0.1)
```

A FORWARD PROPAGATION NETWORK

This is called a densely connected network since every node is connected to the node in the next layer.

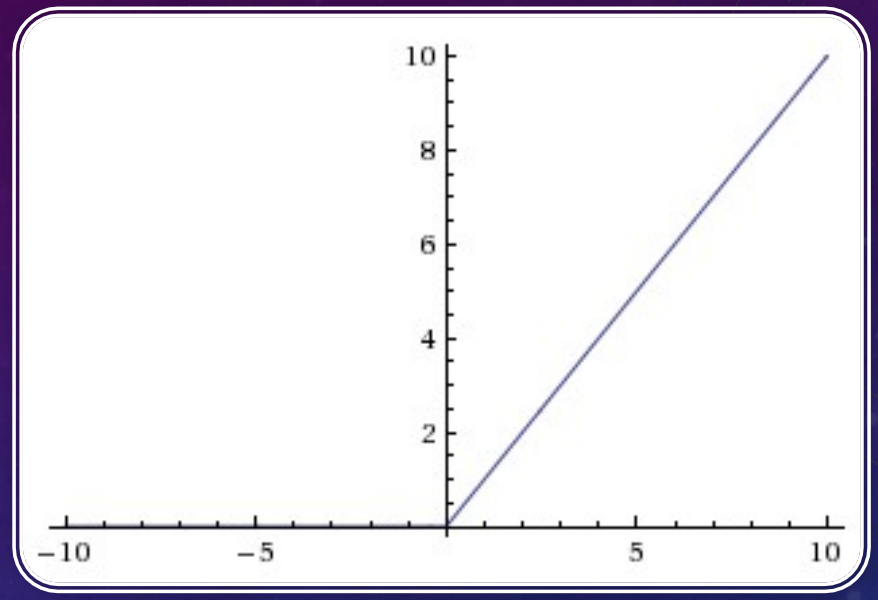


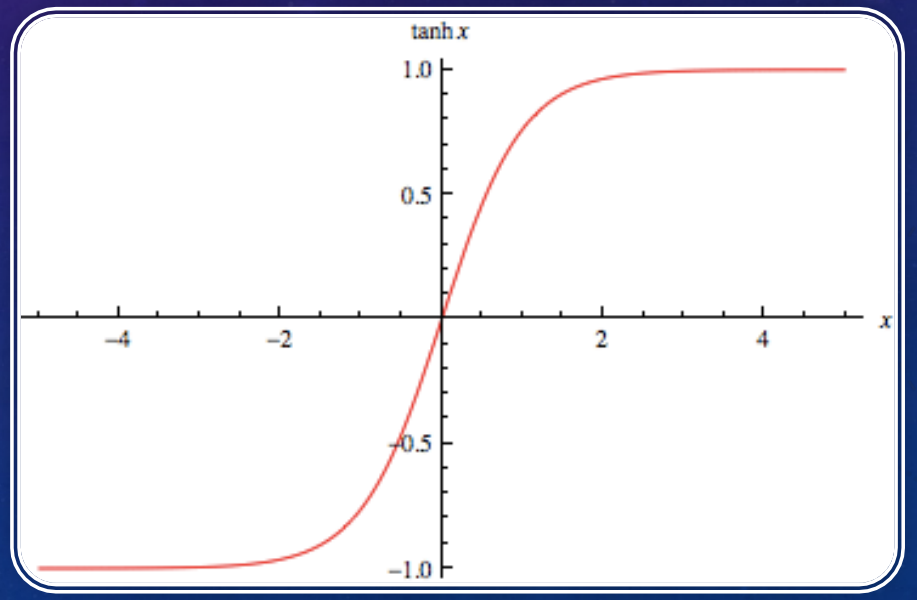
Node value = ∑ (weights * incoming values)

- Weights initialized randomly.
- Weights change to get better predictions using backward propagation (coming next)

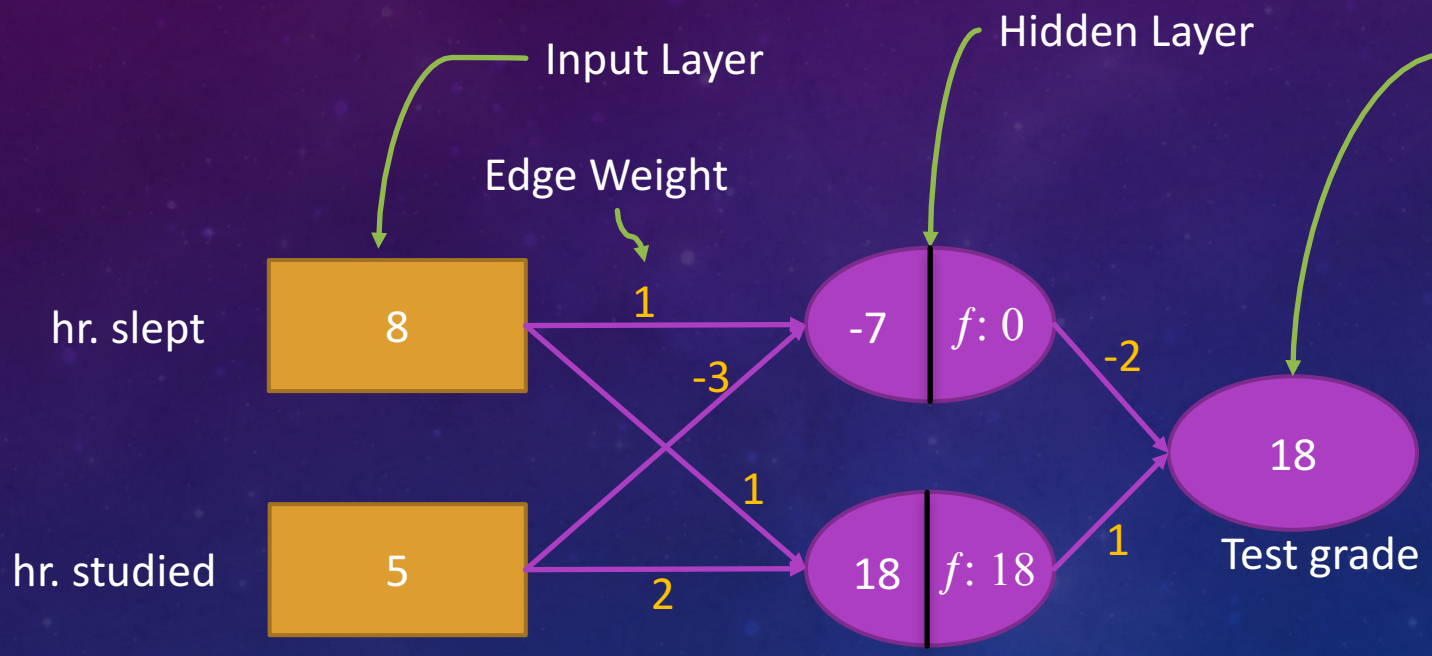
ACTIVATION FUNCTIONS

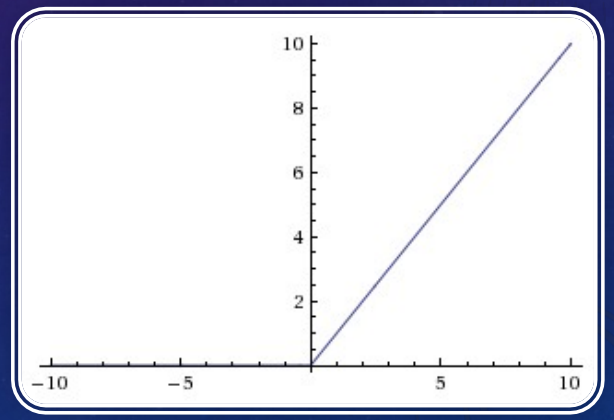
- Top: ReLu (Used in this course).
- Bottom: tanh (Also popular).
- Applied to the value of each node after “node value” has been calculated
- Used to capture nonlinearities in data (ex: will 0 to 8 hours of sleep have the same effect as 8 to 16?). An activation function gives the model the ability to use nonlinearities like these.
- Each layer gets an activation function. You can play around with different functions for different layers.





EXAMPLE WITH A **RELU** ACTIVATION FUNCTION

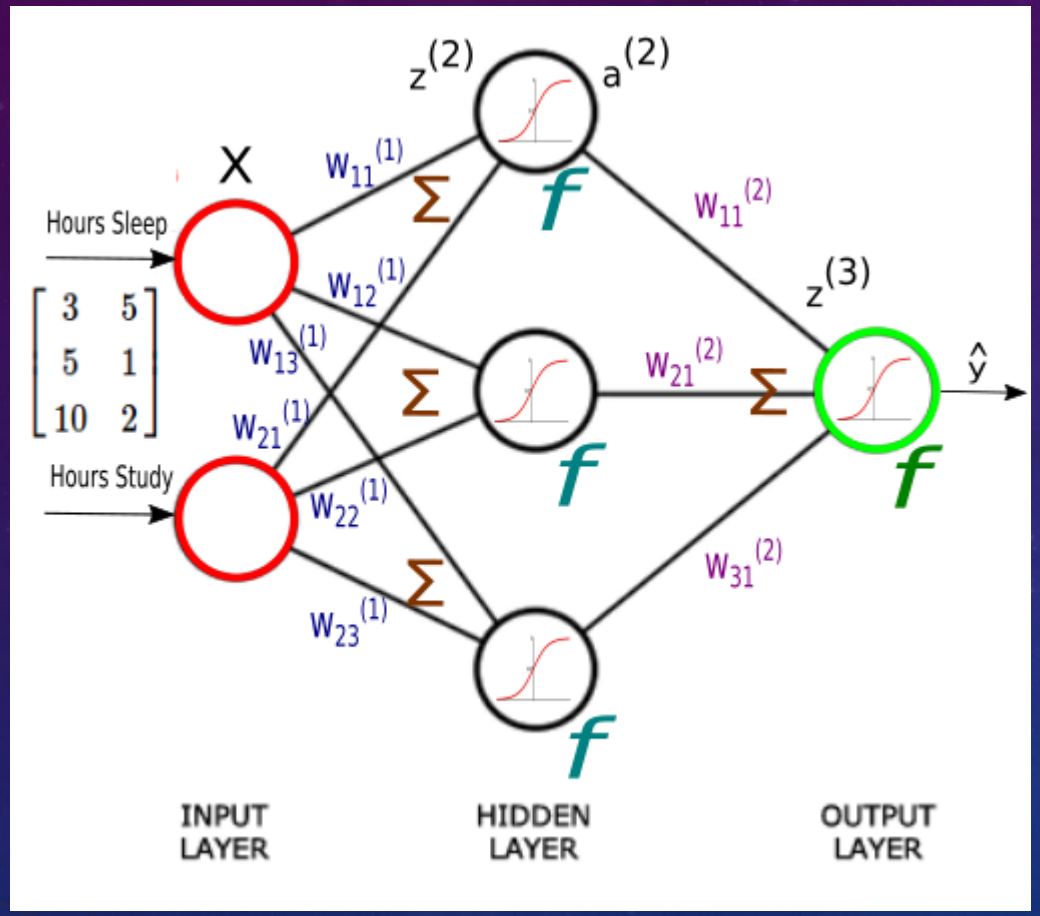


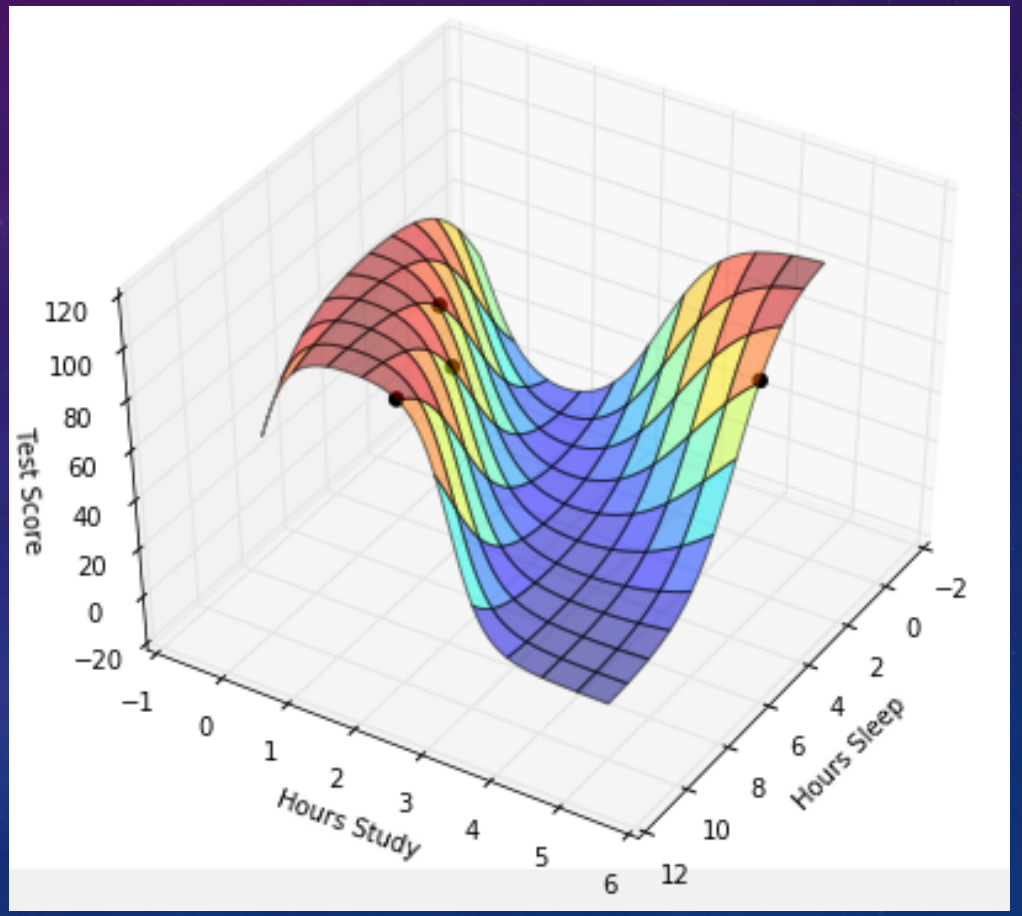


Node value = ∑ (weights * incoming values)

With the RELU activation function applied the hidden nodes, a negative node value outputs 0

TEST GRADE EXAMPLE

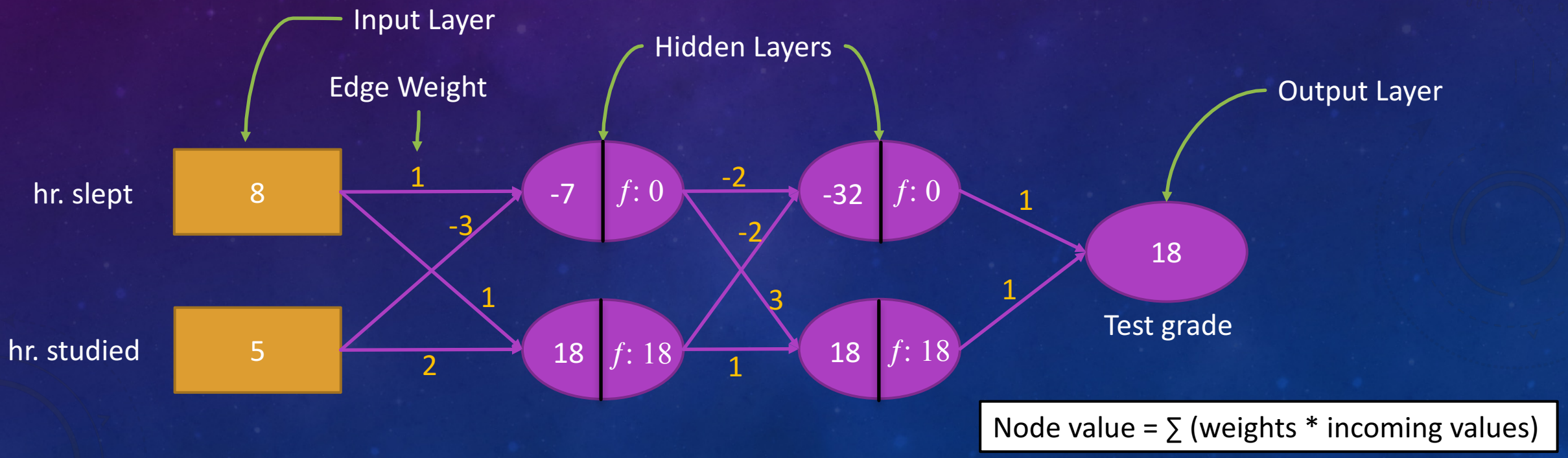




From BogoToBogo

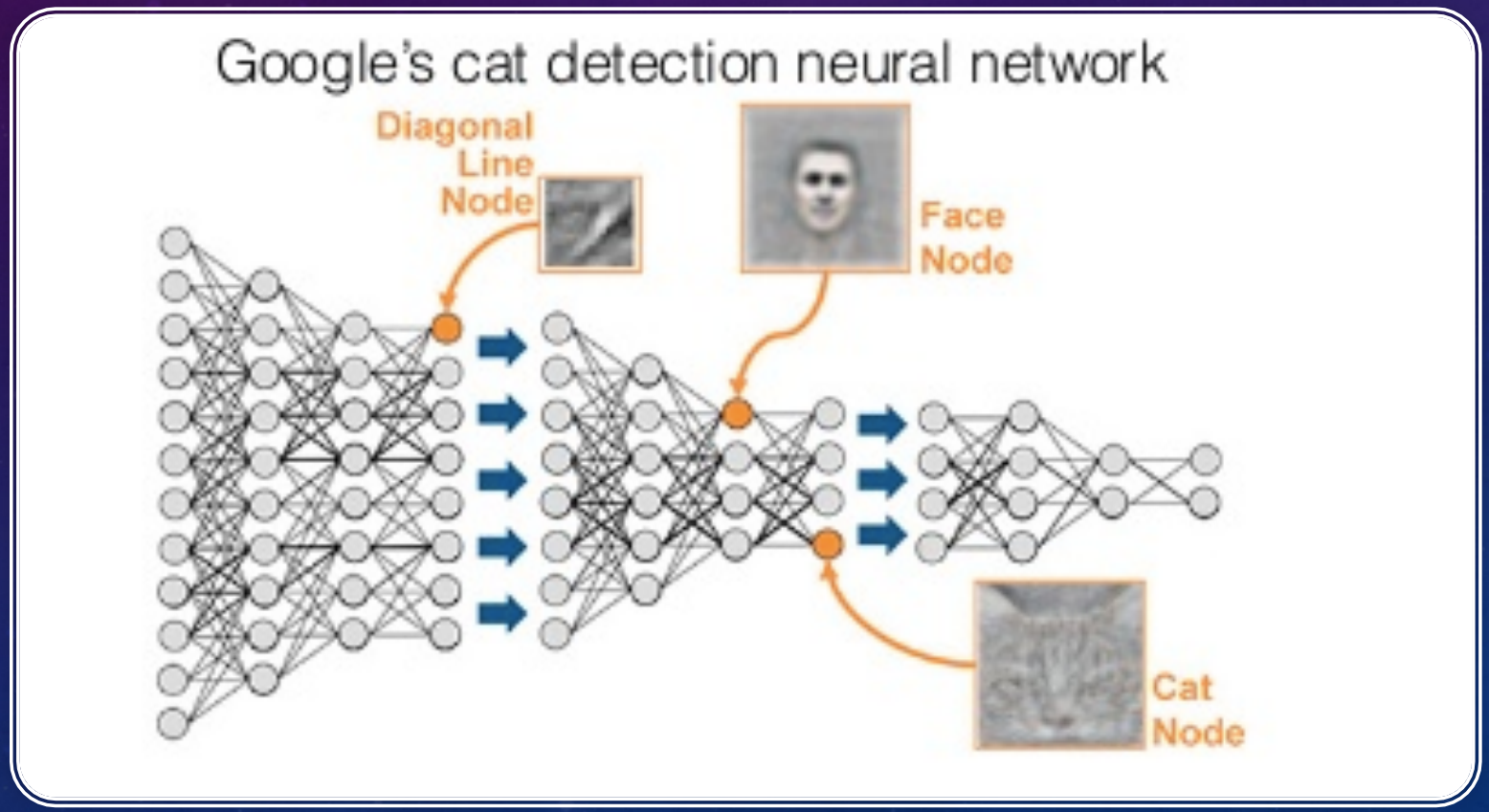
ADDING LAYERS

- Adding more layers gives the neural network the chance to learn more complex relationships in the data faster* (research still going on in this)
- Adding more layers doesn't change the math of what we learned. The forward propagation just gets applied sequentially from the first layer to the last.



ON A SIDE NOTE: DEEP LEARNING

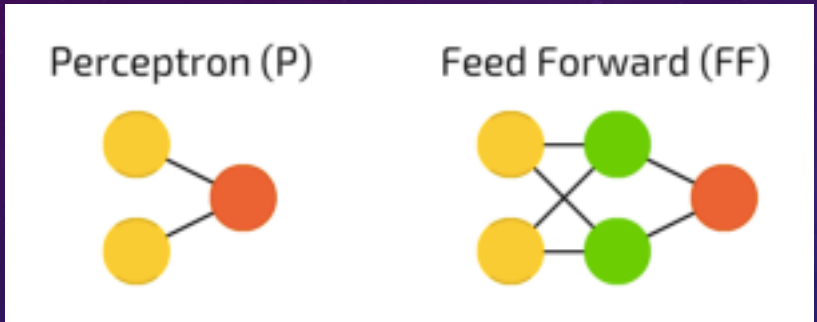
A deep neural network has the ability to capture complex relationships with many layers like how a combination of pixels (low level representation of data) can be understood as an object (high level understanding of data) with many layers.



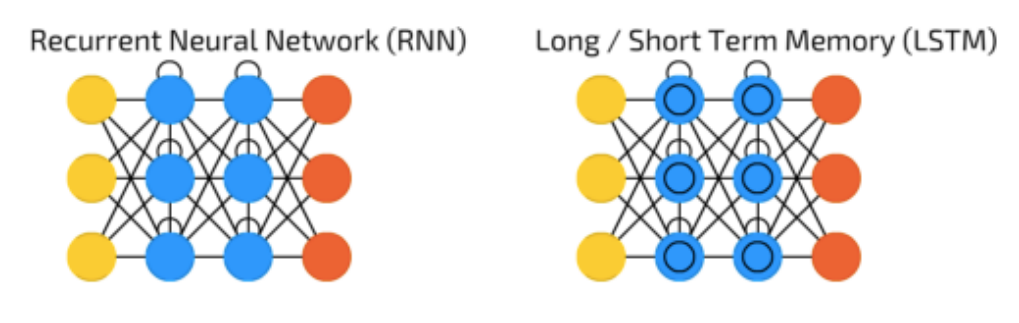
WHAT ARE NEURAL NETWORKS USED FOR?

Taken from: The mostly complete chart of Neural Networks, explained

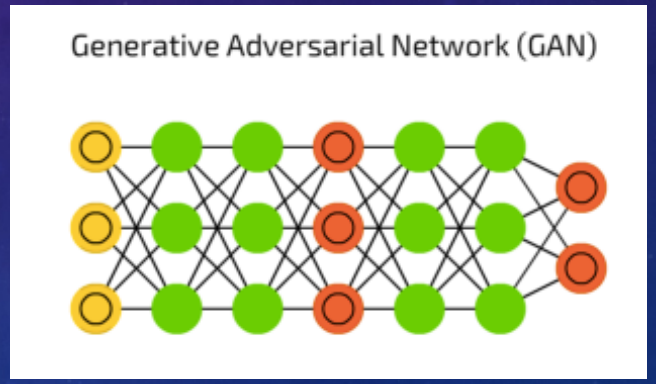
P and FF: Building blocks of all deep learning. Still used for predicting values and classification from complex data sets



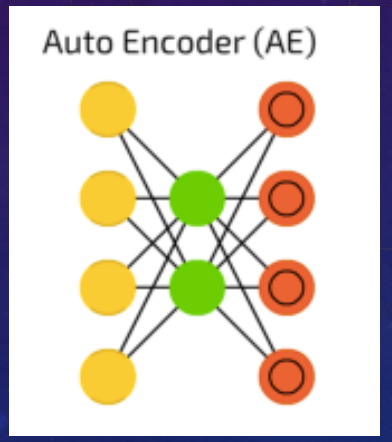
RNN (e.g. LSTM): Used for data that has correlation from one point to the next. i.e. natural language processing and video understanding.



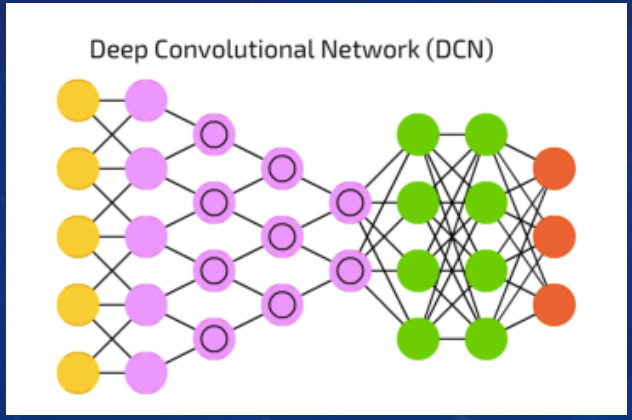
GANs: Built on game theory, used to generate content and "play against itself" to make itself better with no more data.



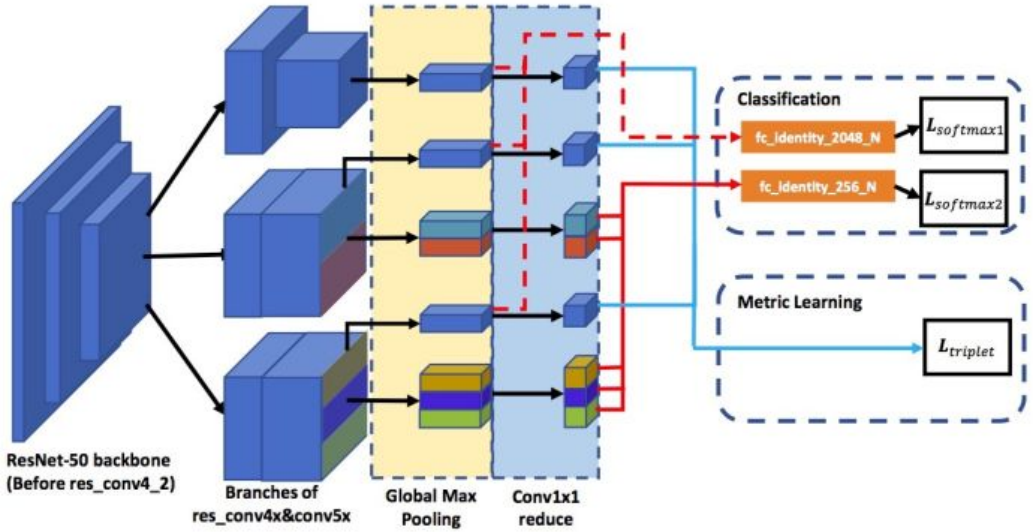
CNN/DCN: Used for dimension reduction and content generation.



CNN/DCN: Used for image processing and recognition.



Person Reidentification



CHAPTER 1 RECAP

HOW A NEURAL NETWORK MAKES A (NON-LINEAR) PREDICTION

Key Terms

- Edge Weight
- Node
- Layer: Input Layer, Hidden Layers, Output Layer
- Forward Propagation
- Activation Functions
- Fully Connected, Feed Forward

CHAPTER 1 RECAP CONT.
HOW A NEURAL NETWORK MAKES A (NON-LINEAR) PREDICTION

Common Questions

- How many nodes per layer should we have?
- How many layers should we have?
  - Recall Ockham's razor: simplest is best – you don’t want to over fit!!
  - Start with a single hidden layer with a few nodes, then asses model performance
  - Keep adding more neurons as long as your model accuracy increases
  - When accuracy stops increasing very much, add another layer and see if that improves
  - Point: Simpler is better so start simple and increase complexity as needed
- What activation functions should I use?
  - ReLU is good for most regressions (start here, change if you have reason to)
  - Tanh is good for classifications (note how it maps all outputs between 0 and 1)

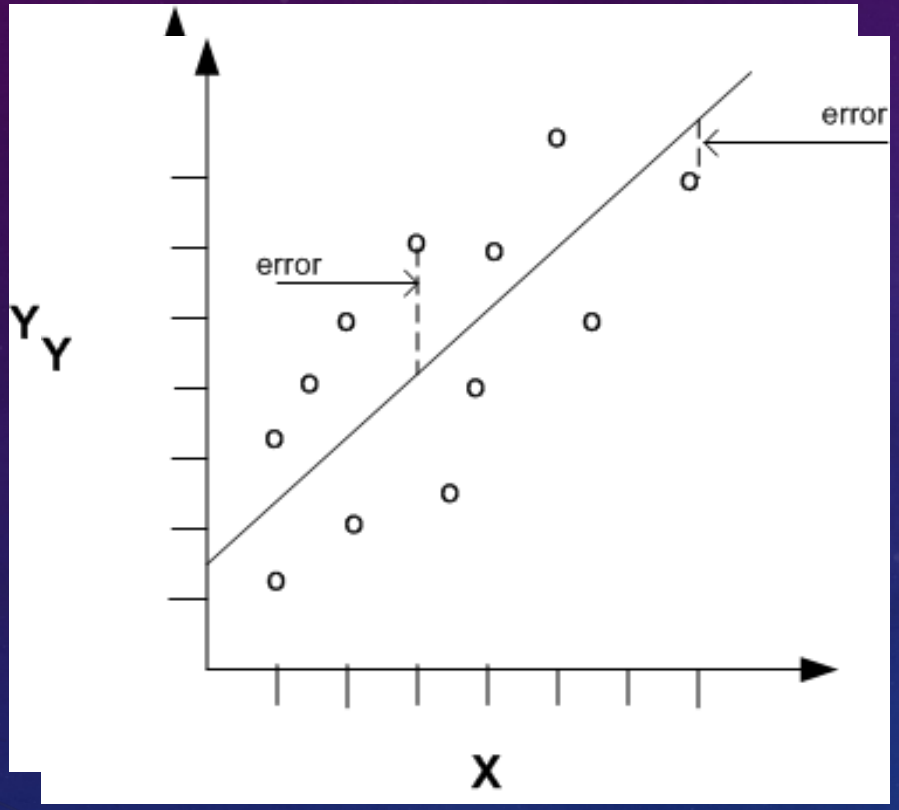
CHAPTER 2:
OPTIMIZING/ “LEARNING” A MACHINE LEARNING MODEL

Key Terms

- Loss Function
- Gradient Descent
- Learning Rate
- Batch Size
- Back-propagation

```
sklearn.neural_network.MLPRegressor(
    hidden_layer_sizes=(100, ),
    activation='relu',
    solver='sgd',
    batch_size=10,
    learning_rate_init=0.001,
    early_stopping=False,
    validation_fraction=0.1)
```

LOSS FUNCTIONS: MINIMIZING ERROR IN PREDICTION



$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

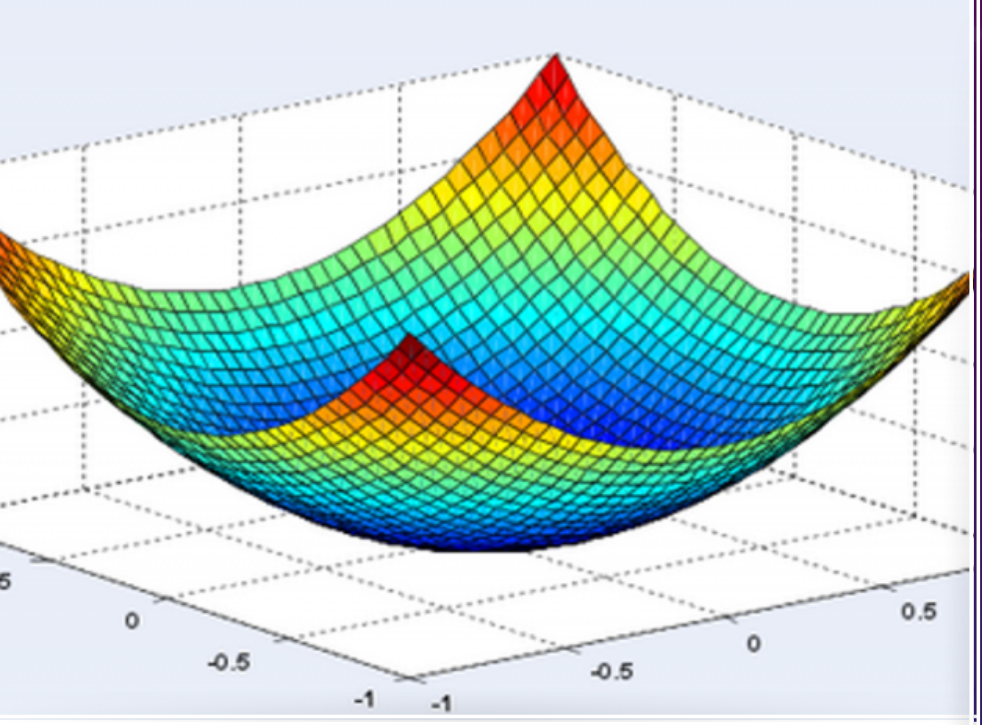
In linear regression, your goal is to draw a line through your data so that your line best "approximates" your data. AKA your line minimizes the total **loss** between your prediction and the data. Simple Linear Regression does this via **mean squared error**.

Mean Squared Error tells you how far your prediction (in this case a straight line) is from predicting you data. **MEAN** is so that you find the best possible fit for all your data. **SQUARED** is so that error is always positive and further away has more weight (formally this is a mean of the L2 distances)

MSE is an example of a **LOSS FUNCTION**: a way for your model to access its accuracy, AKA how far off it was. In the end, the model's goal is simply an optimization problem: Minimize the loss!

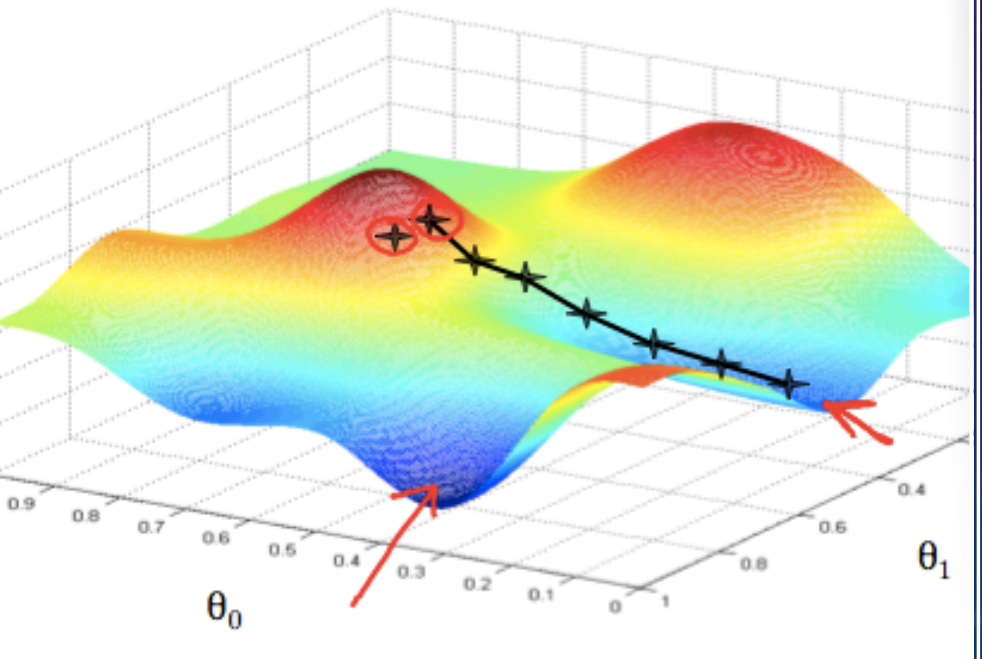
Note: Also called an *objective function*.

MORE ON OPTIMIZATION



← Convex Optimization: The loss function's global minimal is the local minimal:

- Much easier. Immediate minimization by finding the coefficients which sets first derivate of loss function = 0.
- Derivation of Linear Regression

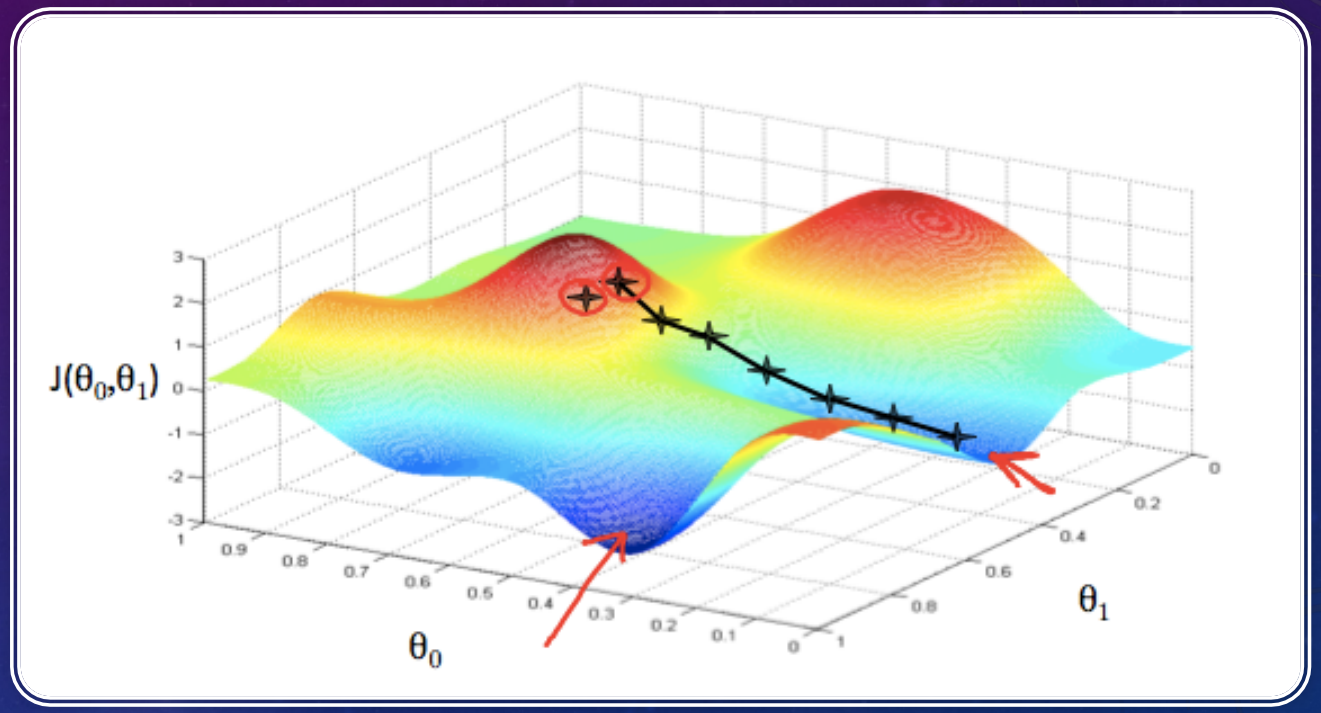


← Non-convex Optimization

- Much harder… but more common with more complex models.
- Many local optimals. How do we converge to a good one?
- When we have many parameters such that decreasing the loss along one parameter increases it along another axis.
- What we will focus on.

GRADIENT DESCENT

- The process/algorithm by which we update the parameters to minimize the error.
- **Gradient:** A collection of slopes
- **Decent:** Moving down a slope
- **Gradient Decent:** Moving down a collection of slopes (to minimize error)
- Gradient Decent moves a “little bit” in the direction that minimizes loss.
- Why not just move all the way?
- Gradient Descent Wikipedia



$$x^{(k+1)} = x^{(k)} - \alpha \nabla f\left(x^{(k)}\right)$$

θ_0, θ_1 are parameters which effect our loss (i.e. weights in our neural network.)

$J(\theta_0, \theta_1)\ is\ the\ loss\ function$

We want to move down the slope of the weights to minimize the loss

LEARNING RATE

- That “little bit” that we move in the direction of to minimize error.
- An industry common learning rate is **0.01**
- Later we’ll learn about *optimizers* which find the optimal learning rate for you.
- **Too big** of a learning rate will overshoot your loss-function minimum and your loss may “bounce around” and not converge to a min.
- **Too small** may make your machine learning take too long to learn.

BATCH SIZE

- A batch is a subset of your overall data that you feed through your model before calculating your loss and adjusting your weights.
- Why not just feed one data point through at a time? Why not feed it all through?
- The rule of thumb is the more noise you think there is in your data, the bigger your batch size should be.
- Batch size – is a concept of “stochastic gradient decent” where our walk towards a minimal is like a drunk person stumbling towards a goal.

CHAPTER 2.B:
OPTIMIZING/ “LEARNING” A NEURAL NETWORK MODEL

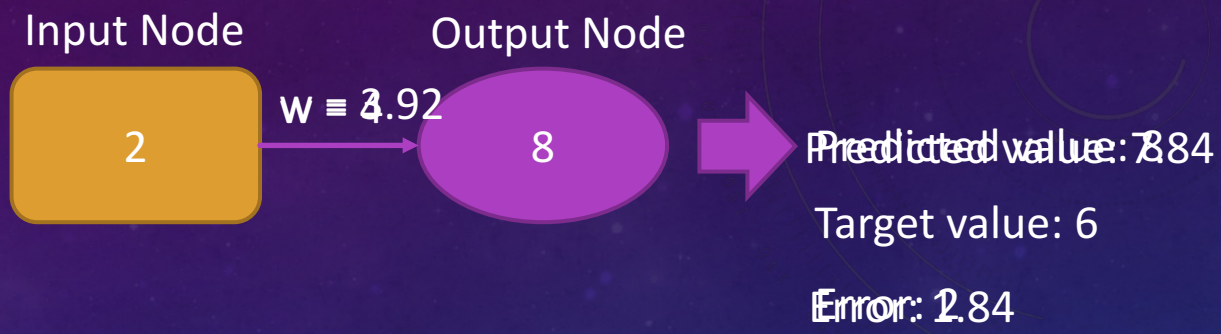
- Gradient Descent In a Neural Network
- Backpropagation (more than one layer)

CALCULATING GRADIENT DESCENT IN A NEURAL NET

- How will we adjust the weights to do better next time?
- An adjustment value for each weight will be calculated using three things with respect to that weight
  1. The node value feeding into that weight
  2. The *slope (*derivative) of the activation function of the node feeding into the weight
  3. The *slope (*derivative) of the loss function of the node the weight is feeding into
- Finally, the gradient gets multiplied by the *learning rate* to get the final adjustment weight
- AND NOW WE UPDATE THE WEIGHT
  - New_weight = old_weight - adjustment

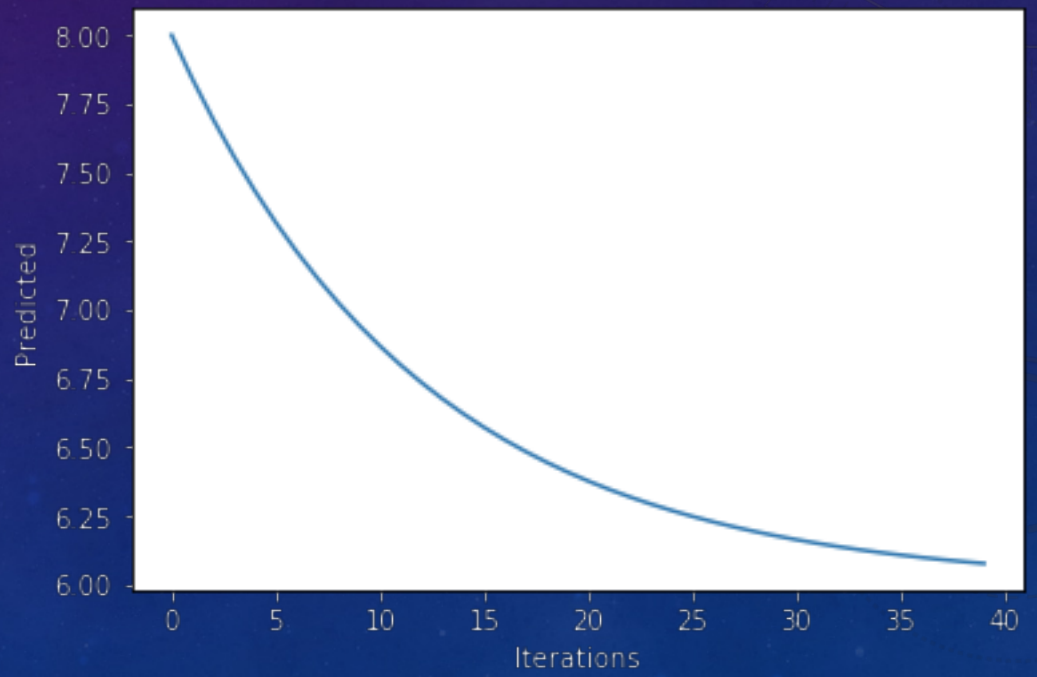
$$x^{(k+1)} = x^{(k)} - \alpha \nabla f\left(x^{(k)}\right)$$

SIMPLE EXAMPLE OF GRADIENT DESCENT



- Input node value: 2
- Recall loss function MSE is (predicted-target)^2 = (error)^2
- Derivative of loss function MSE: 2 * (error) = 4
- Activation function derivative: Is 1 because the node with value 2 is an input
- LEARNING RATE = .01
- Adjustment = (2 * 4 * 1) * .01 = .08
- $new\_weight = 4 - .08 = 3.92$
- (note this is a batchsize = 1)

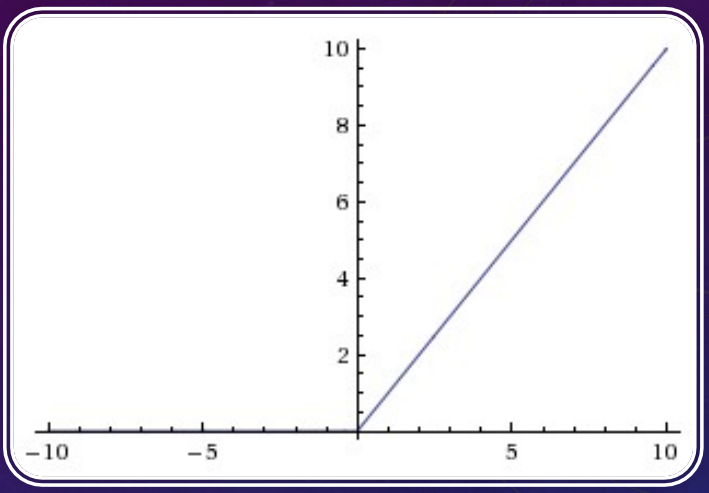
[A very good video explanation]
[Interactive code and math Together]

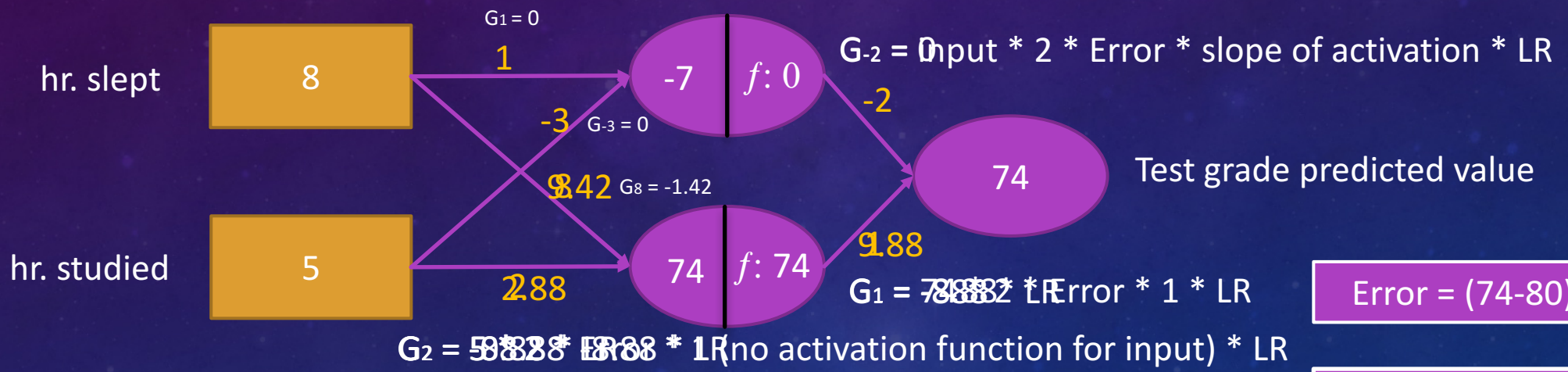


A graph of multiple iterations of gradient decent to adjust weights… see how the predictions get closer to the expected over iterations

EXPANDING TO MULTIPLE LAYERS (BACKPROPAGATION)

Assume each hidden layer has the ReLu activation function and batchsize = 1.





Error = (74-80) = -6

Test Grade Target Value: 80

$$gradient_w = (input\ value \times slope\ of\ loss\ function \times slope\ of\ activation\ function\ of\ the\ node\ your\ feeding\ from)$$

Learning Rate = .01

New Weight = old weight - gradient

CHAPTER 2:
OPTIMIZING/ “LEARNING” A MACHINE LEARNING MODEL

Key Terms

- Loss Function
- Gradient Descent
- Learning Rate
- Batch Size
- Back-propagation

```
sklearn.neural_network.MLPRegressor(
    hidden_layer_sizes=(100, ),
    activation='relu',
    solver='sgd',
    batch_size=10,
    learning_rate_init=0.001,
    early_stopping=False,
    validation_fraction=0.1)
```

CHAPTER 3:
OVERFITTING AND WHEN TO STOP TRAINING

Key Terms

- Validation Set
- Overfitting
- Early Stopping

```
sklearn.neural_network.MLPRegressor(
    hidden_layer_sizes=(100, ),
    activation='relu',
    solver='sgd',
    batch_size=10,
    learning_rate_init=0.001,
    early_stopping=False,
    validation_fraction=0.1)
```

VALIDATION SET

- If I want to really test how well you know a subject, I will give you a test with problems that look similar to the practice problems but are slightly different.
- If you learned the underlying concepts, you will *generalize* well to the new problems.
- However, if you studied the practice problems so much that you become only good at *those* problems, you may not do so well on new problems. This is *overfitting*.
- The goal then is to keep some “problems” (e.g. data) from the machine learning model and see how well it can do on *them*. That’s a validation set!

PROBLEMS WITH OVERFITTING

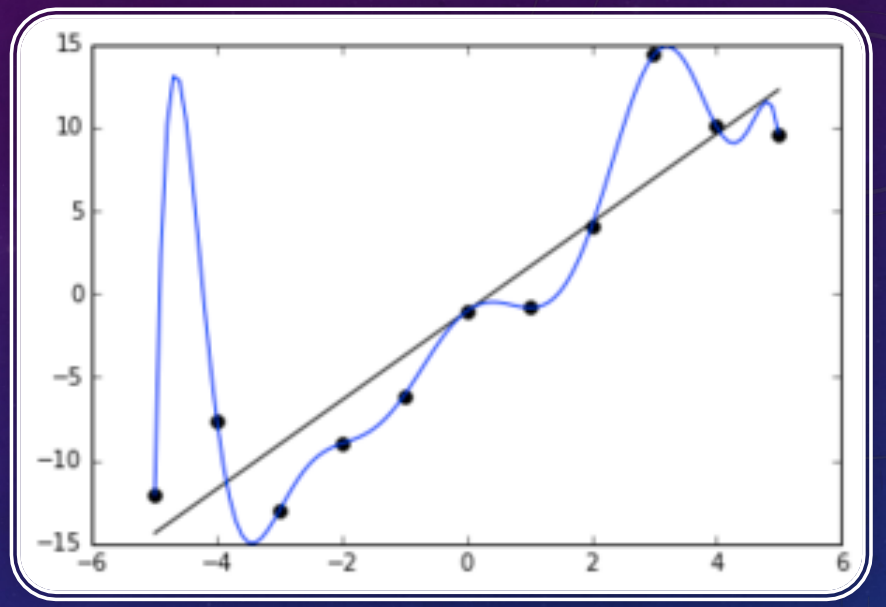
A model can over-fit some training data by becoming “too good” at understanding the training data so that its not learning the underlying trend.

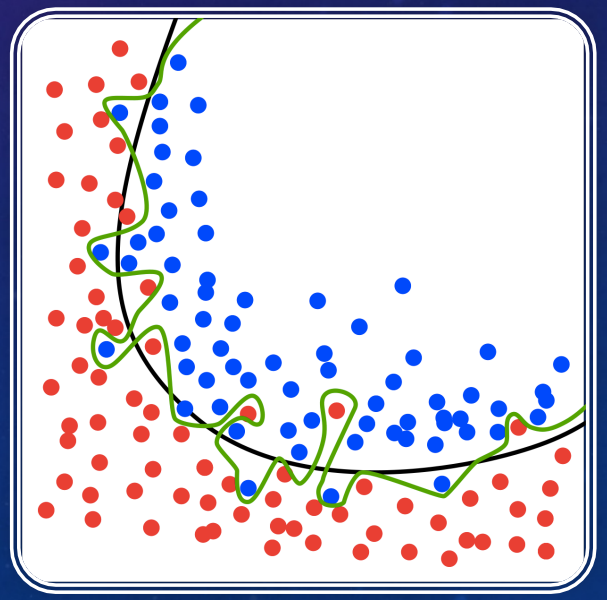
How can a model over-fit?

1. Increasing nodes or layers to the network increases model complexity.
2. Running more *epochs* also can over fit your data.

* *epochs* is the number of times your model runs through the entire training set (e.g. iterations)

Overfitting a regression model by capturing the noise more-so than the underlying trend.

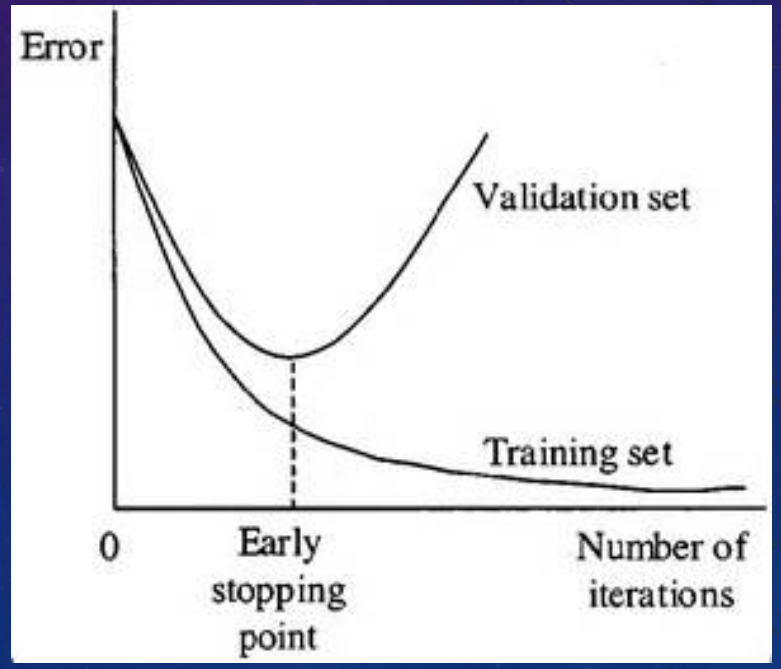




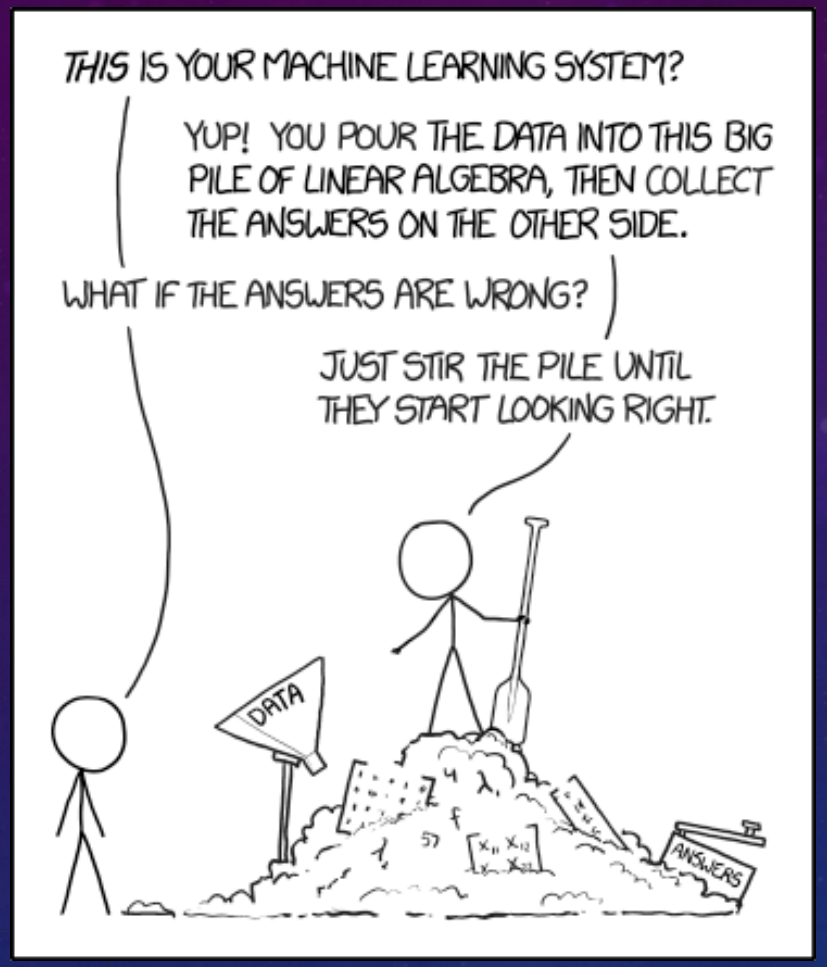
Overfitting a classification model by again capturing the noise more so then general trend.

EARLY STOPPING

- You can know overfitting is happening when your training loss keeps decreasing but your validation loss starts increasing!
- Early stopping will terminate training when validation score is not improving (or starts getting worse)!
- “Patience” or “n_iter_no_change” in sklearn is the number of iterations we are allowed to keep going without seeing any progress in the validation loss.



WHAT TO DO TO IMPROVE PERFORMANCE



Data

- Regularization – data on same scale?
- Randomization – look at your batches – are they sufficiently random and evenly represent your overall population?
- Training/test split. Does it represent the same distribution for all your predictor variables? Visualize your data!
- Data has too much noise? Visualize your data!
- Class imbalance? Do you have enough examples of both classes?
- Enough training examples? *Curse of dimensionality.*

Network

- Add methods to avoid overfitting: regularization and dropout

More ways to improve your model

IMPLEMENTING NEURAL NETWORKS
AT FOUR LEVELS OF DEPTH

- Novice: SkLearn (you do that here!)
- Intermediate: Keras
- Master: TensorFlow
- Superhuman: Implimenting From Scratch

TENSOR FLOW PLAYGROUND

- Using tensorflow playground
- Try setting noise to max and batch size small. See how it jumps around a lot more!
- Try setting the learning rate above 1 and see how much it jumps around, and how it over fits, and how thick (which means large valued) the weights get! Now try adding regularization. See how that counteracts those problems.
- Try adding more nodes/layers using the method we proposed earlier.
- Hover over each node and edge and really try to understand how the whole thing is working. Nearly everything there was explained in these slides so it's a great way to test your knowledge.

QUIZ TIME!

- Please go to this link