An Introduction to Machine Learning and Neural Networks

Raymond Matson

University of California, Riverside

April 2, 2021

Overview

Notes

- This is only an introduction into the subject, geared towards surface-level programming and basic understanding as opposed to theory (which is easy to deep dive into).

Notes

- This is only an introduction into the subject, geared towards surface-level programming and basic understanding as opposed to theory (which is easy to deep dive into).
- This will be somewhere between what you normally find online, theory or application, but you'll get neither really.

Notes

- This is only an introduction into the subject, geared towards surface-level programming and basic understanding as opposed to theory (which is easy to deep dive into).
- This will be somewhere between what you normally find online, theory or application, but you'll get neither really.
- A lot of details are different "in practice."

Notes

- This is only an introduction into the subject, geared towards surface-level programming and basic understanding as opposed to theory (which is easy to deep dive into).
- This will be somewhere between what you normally find online, theory or application, but you'll get neither really.
- A lot of details are different "in practice."
- Everything discussed in this presentation was figured out between the 1960's and the 1980's.

Questions

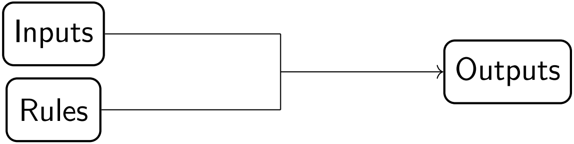
- What is machine learning? What does it mean for a machine to learn?

- How would you describe a neural network?

- What is a neuron in an artificial neural network?

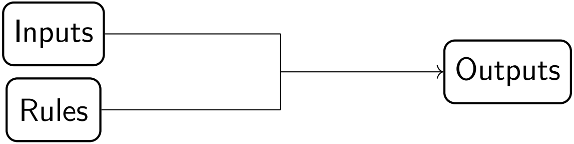
ML Coding vs Traditional Coding

- Traditional coding:

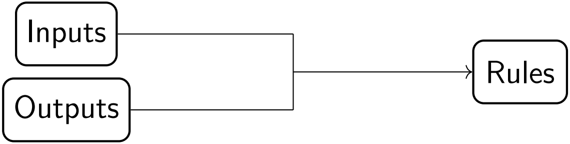


ML Coding vs Traditional Coding

- Traditional coding:

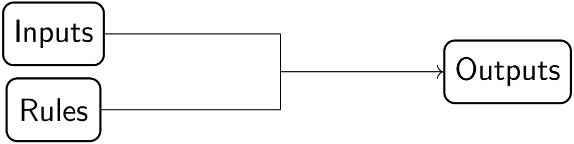


- ML coding:

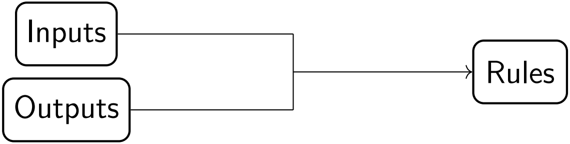


ML Coding vs Traditional Coding

- Traditional coding:



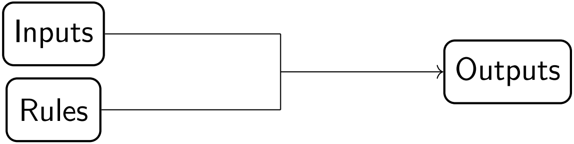
- ML coding:



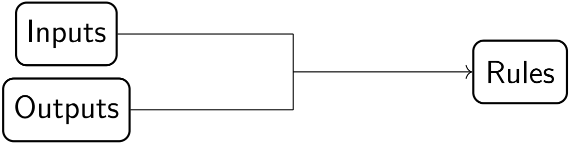
- Difficult to change this mindset $\longrightarrow$ job opportunities for mathematicians.

ML Coding vs Traditional Coding

- Traditional coding:



- ML coding:



- Difficult to change this mindset $\longrightarrow$ job opportunities for mathematicians.
- Write most of the program's backbone before testing.

Set Up

- Suppose you have a bunch of (labeled) data and you want to discover patterns or create predictions.

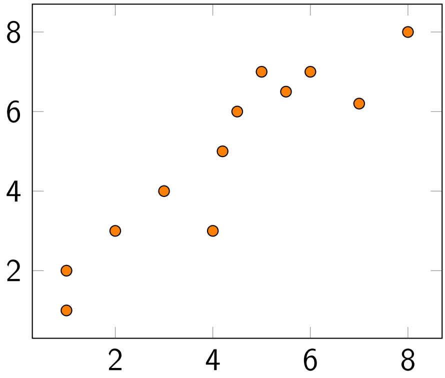
Set Up

- Suppose you have a bunch of (labeled) data and you want to discover patterns or create predictions.

- First separate your data into *training data* (data used to train the *model*) and *testing data* (data used to test accuracy).

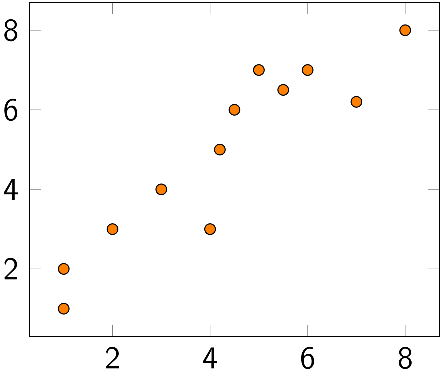
Simple Linear Regression Scenario

For now let's assume we only have 1 parameter for our data.
We want to make a "best fit line" within our data points.



Simple Linear Regression Scenario

For now let's assume we only have 1 parameter for our data.
We want to make a "best fit line" within our data points.



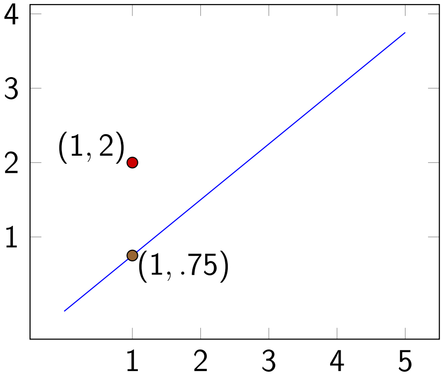
What would be a good slope for this?

Guessing

The program will first "randomly guess" the slope of the line.

Guessing

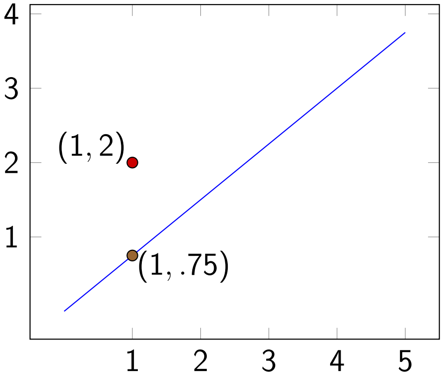
The program will first "randomly guess" the slope of the line.



Guesses $m = \frac{3}{4}$.
$\Rightarrow$ Error of 1.25.
Square the error.

Guessing

The program will first "randomly guess" the slope of the line.

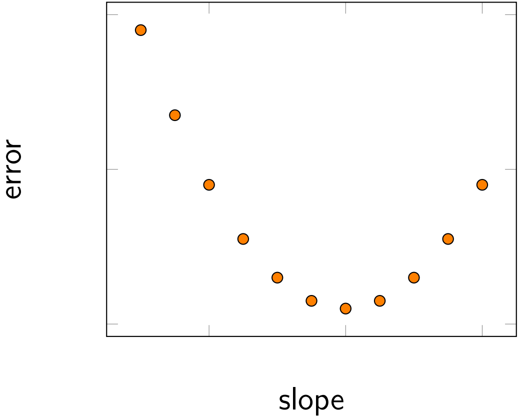


Guesses $m = \frac{3}{4}$.
$\Rightarrow$ Error of 1.25.
Square the error.

Can repeat for each data point and square the errors to keep it positive.

Loss Function

Plot the error. It's parabolic since we used a squared loss function. This would look different if we used a different error function, such as absolute value or Huber loss.



Optimize

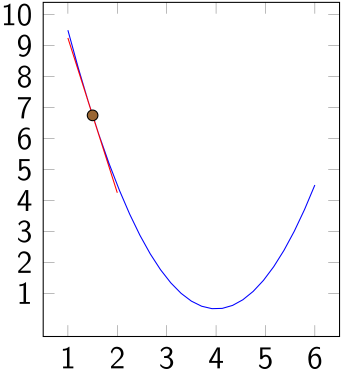
After the first guess, how does the model find a better guess?

Optimize

After the first guess, how does the model find a better guess? $m_{new} = m_{current} - kE'(m_{current})$ where $E(x)$ is this error function.

Optimize

After the first guess, how does the model find a better guess?
$m_{new} = m_{current} - kE'(m_{current})$ where $E(x)$ is this error function.



$$
\begin{aligned}
m_{new} &= 0.75 - (0.01)(-13) \\
&= 0.88
\end{aligned}
$$

which is a better slope.

Now repeat.

But What About...

- This process of finding the best slope via minimizing the error is called *gradient descent*.

But What About...

- This process of finding the best slope via minimizing the error is called *gradient descent*.

- But Raymond, aren't there already formulas out there that can just find the optimal error/best slope?

But What About...

- This process of finding the best slope via minimizing the error is called *gradient descent*.

- But Raymond, aren't there already formulas out there that can just find the optimal error/best slope?
  Yes, however, this is for simple cases (such as the previous example).

But What About...

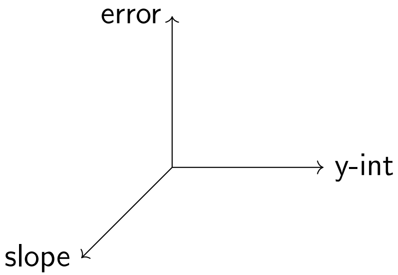
- This process of finding the best slope via minimizing the error is called *gradient descent*.

- But Raymond, aren't there already formulas out there that can just find the optimal error/best slope?
  Yes, however, this is for simple cases (such as the previous example).

- Notice in the example we just did we had a y-intercept at the origin. What do we do if the y-intercept is shifted?

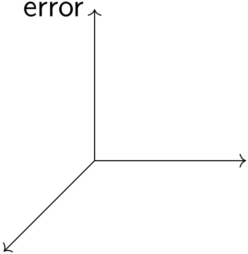
Guessing y-int as well

Add a dimension to guess the y-intercept as well



Guessing y-int as well

Add a dimension to guess the y-intercept as well



and use partials instead

$$m_{new} = m_{current} - k\frac{\partial}{\partial x}E(x, y)$$

$$b_{new} = b_{current} - k\frac{\partial}{\partial y}E(x, y).$$

Multiple Parameters

So if you have a bunch of parameters, you would instead use

Multiple Parameters

So if you have a bunch of parameters, you would instead use

$$
\begin{aligned}
y &= b + m_1 x_1 + m_2 x_2 + \cdots + m_{n-1} x_{n-1} \\
b_{new} &= b_{current} - k \frac{\partial}{\partial b} E(b, m_1, m_2, \cdots, m_{n-1}) \\
m_{1,new} &= m_{1,current} - k \frac{\partial}{\partial m_1} E(b, m_1, m_2, \cdots, m_{n-1}) \\
&\vdots \\
m_{n-1,new} &= m_{n-1,current} - k \frac{\partial}{\partial m_{n-1}} E(b, m_1, m_2, \cdots, m_{n-1}).
\end{aligned}
$$

Multiple Parameter Effects

- Clearly you would be plotting points in $\mathbb{R}^{n+1}$.

Multiple Parameter Effects

- Clearly you would be plotting points in $\mathbb{R}^{n+1}$.
- Something you may want to see is how the results are changing over time, which is hard to see conceptually.

Multiple Parameter Effects

- Clearly you would be plotting points in $\mathbb{R}^{n+1}$.
- Something you may want to see is how the results are changing over time, which is hard to see conceptually.
- Usually you can only graph individual parameters/slices at a time
  $\Rightarrow$ harder to understand results
  $\Rightarrow$ need more faith in the programming.

Multiple Parameter Effects

- Clearly you would be plotting points in $\mathbb{R}^{n+1}$.
- Something you may want to see is how the results are changing over time, which is hard to see conceptually.
- Usually you can only graph individual parameters/slices at a time
  $\Rightarrow$ harder to understand results
  $\Rightarrow$ need more faith in the programming.
- What about nonlinear best fit curves?

Multiple Parameter Effects

- Clearly you would be plotting points in $\mathbb{R}^{n+1}$.
- Something you may want to see is how the results are changing over time, which is hard to see conceptually.
- Usually you can only graph individual parameters/slices at a time
  $\Rightarrow$ harder to understand results
  $\Rightarrow$ need more faith in the programming.
- What about nonlinear best fit curves?
  Assuming you know the degree already, similar idea but grosser: $y = ax^3 + bx^2 + cx + d$
  - More annoying to graph.
  - How will the error function change?

Alternative Scenarios

- Logistic Regression: Two possibilities such as T/F, Pass/Fail, Survive/Die, etc.

Alternative Scenarios

- Logistic Regression: Two possibilities such as T/F, Pass/Fail, Survive/Die, etc.
    - Ex: 93% match on netflix

Alternative Scenarios

- Logistic Regression: Two possibilities such as T/F, Pass/Fail, Survive/Die, etc.
  - Ex: 93% match on netflix
- GA (Genetic Algorithm), Neat (NeuroEvolution of Augmenting Topologies), SA (Simulated Annealing), EM (Expectation-Maximization), PSO (Partial Swarm Optimization) are all different types of machine learning algorithms, all with their own benefits and limitations.

Alternative Scenarios

- Logistic Regression: Two possibilities such as T/F, Pass/Fail, Survive/Die, etc.
    - Ex: 93% match on netflix
- GA (Genetic Algorithm), Neat (NeuroEvolution of Augmenting Topologies), SA (Simulated Annealing), EM (Expectation-Maximization), PSO (Partial Swarm Optimization) are all different types of machine learning algorithms, all with their own benefits and limitations.
    - Different error minimizations for different situations
    - Supervised vs semi-supervised vs unsupervised/independent learning (tagged vs untagged)

Alternative Scenarios

- Logistic Regression: Two possibilities such as T/F, Pass/Fail, Survive/Die, etc.
  - Ex: 93% match on netflix
- GA (Genetic Algorithm), Neat (NeuroEvolution of Augmenting Topologies), SA (Simulated Annealing), EM (Expectation-Maximization), PSO (Partial Swarm Optimization) are all different types of machine learning algorithms, all with their own benefits and limitations.
  - Different error minimizations for different situations
  - Supervised vs semi-supervised vs unsupervised/independent learning (tagged vs untagged)
- Let's finally look at a neural network!

Alternative Scenarios

- Up until now, everything has either been “linear” or we know a lot of the information already, AKA that was the easy stuff.

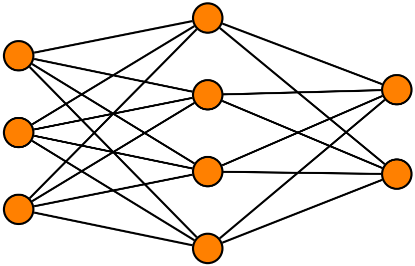
- Works for simple enough scenarios but will be fairly ineffective for more complicated situations (which aren’t hard to find).

What is a Neural Network

- In order to deal with more complicated scenarios, we'll use a neural network.

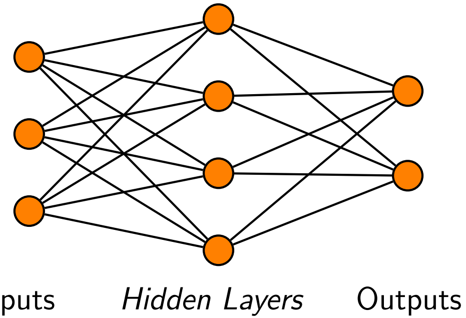
What is a Neural Network

- In order to deal with more complicated scenarios, we'll use a neural network.
- Conceptually, it's a chain of (complete) bipartite graphs.



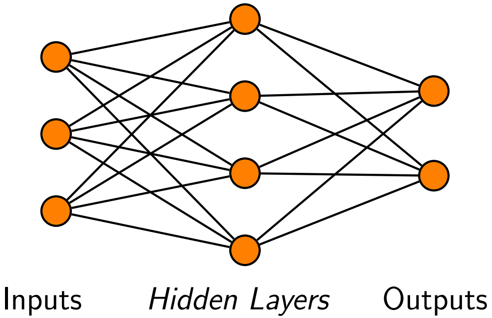
What is a Neural Network

- In order to deal with more complicated scenarios, we'll use a neural network.
- Conceptually, it's a chain of (complete) bipartite graphs.



What is a Neural Network

- In order to deal with more complicated scenarios, we'll use a neural network.
- Conceptually, it's a chain of (complete) bipartite graphs.



- Each edge has a weight that gets adjusted (like the slopes in linear regression model).

MNIST Example

- We'll use a categorical example to understand this, however, these neural networks can also be used in noncategorical machine learning as well.

MNIST Example

- We'll use a categorical example to understand this, however, these neural networks can also be used in noncategorical machine learning as well.
  - Makes more sense at first in a more categorical version in my opinion.

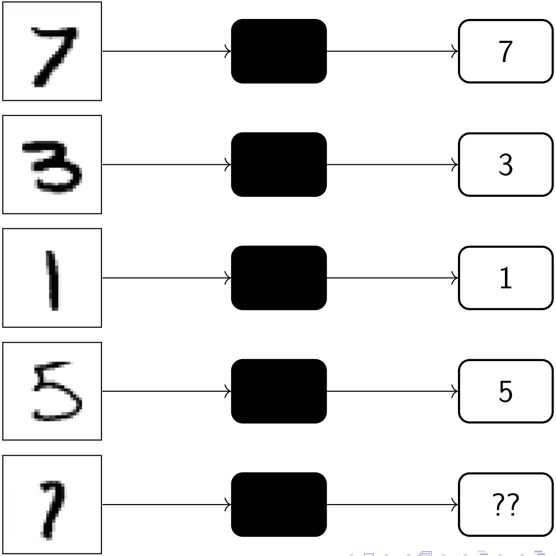
MNIST Example

- We'll use a categorical example to understand this, however, these neural networks can also be used in noncategorical machine learning as well.
    - Makes more sense at first in a more categorical version in my opinion.
- A model using the Modified National Institute of Standards & Technology's datasets is a good example to keep in the back of your mind.

MNIST Example

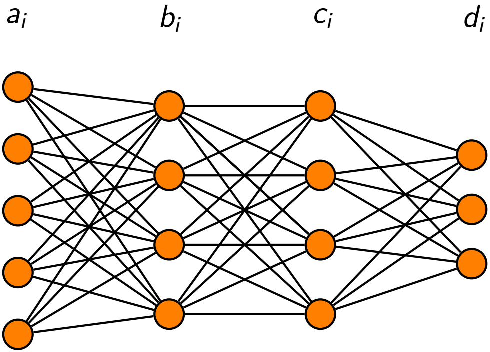
- We'll use a categorical example to understand this, however, these neural networks can also be used in noncategorical machine learning as well.
  - Makes more sense at first in a more categorical version in my opinion.
- A model using the Modified National Institute of Standards & Technology's datasets is a good example to keep in the back of your mind.
  - This particular example using MNIST datasets is typically considered to be the "hello, world" of neural nets.

MNIST Example



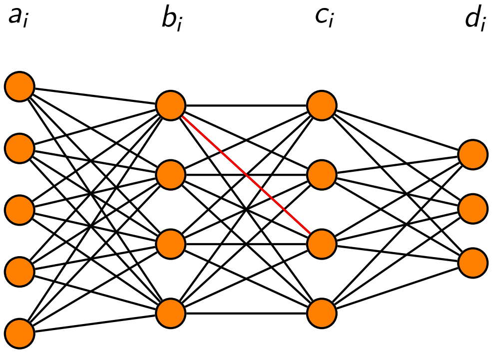
Edge Weights

Each edge has a weight that will get adjusted, just like the slopes in the linear regression example.



Edge Weights

Each edge has a weight that will get adjusted, just like the slopes in the linear regression example.



What do we want $b_1 c_3$'s weight to be? What parameters should it have?

Evaluating Nodes

- Given a bunch of edge weights, what do we do with it?

Evaluating Nodes

- Given a bunch of edge weights, what do we do with it?
- Suppose we have our inputs (the a_i's) and some randomly assigned weights (w_i's). To calculate, say b_1, sum over the products of weights and inputs connected to b_1 and normalize it somehow.

Evaluating Nodes

- Given a bunch of edge weights, what do we do with it?
- Suppose we have our inputs (the a_i's) and some randomly assigned weights (w_i's). To calculate, say b_1, sum over the products of weights and inputs connected to b_1 and normalize it somehow.

$$b_1 = \sigma(w_1 a_1 + w_2 a_2 + w_3 a_3 + w_4 a_4 + w_5 a_5)$$

where $\sigma(x)$ is either a sigmoid function, $\frac{1}{1+e^{-x}}$, or a rectifying activation function, $(\mathrm{ReLU})(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$.

Evaluating Nodes

- Given a bunch of edge weights, what do we do with it?
- Suppose we have our inputs (the a_i's) and some randomly assigned weights (w_i's). To calculate, say b_1, sum over the products of weights and inputs connected to b_1 and normalize it somehow.

$$b_1 = \sigma(w_1 a_1 + w_2 a_2 + w_3 a_3 + w_4 a_4 + w_5 a_5)$$

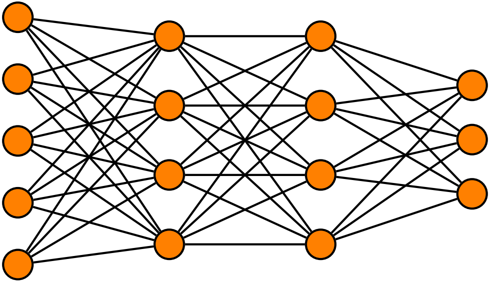
where $\sigma(x)$ is either a sigmoid function, $\frac{1}{1+e^{-x}}$, or a rectifying activation function, $(\mathrm{ReLU})(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$.

- We can also add a *bias* for a binary state by subtracting it inside σ (usually needed if using an activation function like ReLU).

$$b_1 = \sigma(w_1 a_1 + w_2 a_2 + w_3 a_3 + w_4 a_4 + w_5 a_5 - 3)$$

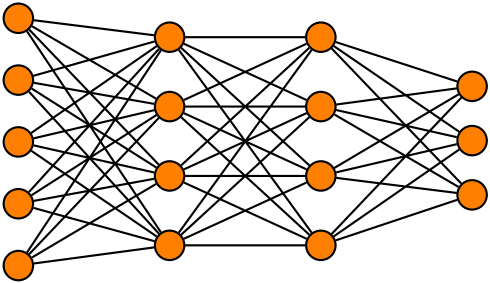
Counting The Variables

- Let's calculate how many weights and biases the neural network will need to figure out.



Counting The Variables

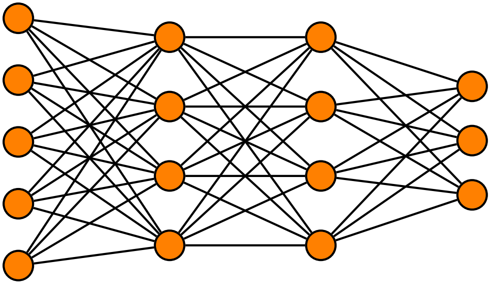
- Let's calculate how many weights and biases the neural network will need to figure out.



- 5 inputs, 2 hidden layers with 4 nodes each, and 3 possible outputs

Counting The Variables

- Let's calculate how many weights and biases the neural network will need to figure out.



- 5 inputs, 2 hidden layers with 4 nodes each, and 3 possible outputs $\Rightarrow (5 \times 4) + (4 \times 4) + (4 \times 3) = 48$ weights and $4 + 4 + 3 = 11$ biases, totaling 59 variables.

Realistically...

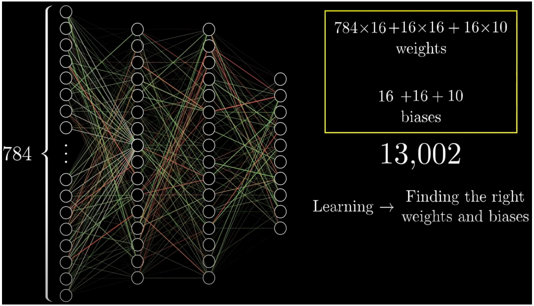
- Unfortunately, you will never find a NN this simple.

Realistically...

- Unfortunately, you will never find a NN this simple.
- In the MNIST example, each pixel is an input. Each jpg is $28 \times 28 \Rightarrow 784$ variables for the first layer alone!

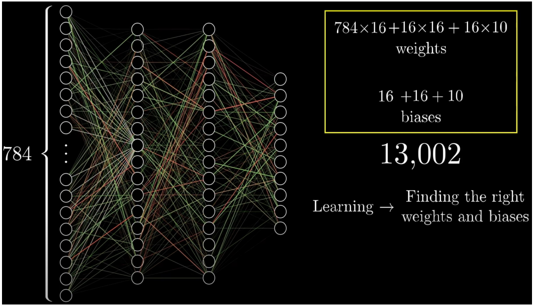
Realistically...

- Unfortunately, you will never find a NN this simple.
- In the MNIST example, each pixel is an input. Each jpg is $28 \times 28 \Rightarrow 784$ variables for the first layer alone!



Realistically...

- Unfortunately, you will never find a NN this simple.
- In the MNIST example, each pixel is an input. Each jpg is $28 \times 28 \Rightarrow 784$ variables for the first layer alone!



- You can imagine there are a lot of variables to look for and approximate $\Rightarrow$ it's better to leave it to a machine.

What is ML

- Back to the question:

What is machine learning? What does it mean for a machine to learn?

What is ML

- Back to the question:

What is machine learning? What does it mean for a machine to learn?

- Machine learning is just when a program minimizes the error of guessed weights and biases.

Linear Algebra Perspective

- But Raymond, everyone says you need to know linear algebra to understand machine learning? Where's all the linear algebra? I want my linear algebra!

Linear Algebra Perspective

- But Raymond, everyone says you need to know linear algebra to understand machine learning? Where's all the linear algebra? I want my linear algebra!
  Or my money back...

Linear Algebra Perspective

- But Raymond, everyone says you need to know linear algebra to understand machine learning? Where's all the linear algebra? I want my linear algebra!
  Or my money back...

- Like a lot of things, writing things with matrices and vectors can clean it up a bit.

Linear Algebra Perspective

- But Raymond, everyone says you need to know linear algebra to understand machine learning? Where's all the linear algebra? I want my linear algebra!
  Or my money back...

- Like a lot of things, writing things with matrices and vectors can clean it up a bit.

- Not to mention, this is probably how your program will calculate everything. Remember, *NumPy* is your friend!

Linear Algebra Perspective

- Let $w_{i,j}$ be the weights of edges between first and second layers, a_i be a the activations from the first layer, and b_i be the biases. Then

$$\sigma(Wa+b) = \sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_k \end{bmatrix}\right).$$

Linear Algebra Perspective

- Let $w_{i,j}$ be the weights of edges between first and second layers, a_i be a the activations from the first layer, and b_i be the biases. Then

$$\sigma(Wa+b) = \sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_k \end{bmatrix}\right).$$

- How does this do the job more efficiently or make things easier?

Neurons

- What is a “neuron” in an artificial neural network?

Neurons

- What is a “neuron” in an artificial neural network?
  A neuron is a function

$$n_i : \{\text{Outputs from Previous Layer}\} \to [0, 1]$$

defined by the weights and biases.

Neurons

- What is a “neuron” in an artificial neural network?
  A neuron is a function

  $$n_i : \{\text{Outputs from Previous Layer}\} \to [0, 1]$$

  defined by the weights and biases.

- Thus a neural network is really just a giant composition of functions.

Neurons

- What is a "neuron" in an artificial neural network?
  A neuron is a function

  $$n_i : \{\text{Outputs from Previous Layer}\} \to [0, 1]$$

  defined by the weights and biases.

- Thus a neural network is really just a giant composition of functions.

- Let's say we set up a NN as explained. What will happen? It's first run through it will spit back something not correct (probably).

Cost Function

- If we give it a 7, it will return random numbers. To change the weights, we need a *cost function*.

Cost Function

- If we give it a 7, it will return random numbers. To change the weights, we need a *cost function*.

$$
\begin{aligned}
\text{Node } 0: &\quad (0.43-0)^2 \longrightarrow 0.1863\\
\text{Node } 1: &\quad (0.28-0)^2 \longrightarrow 0.0784\\
\text{Node } 2: &\quad (0.19-0)^2 \longrightarrow 0.0361\\
\text{Node } 3: &\quad (0.88-0)^2 \longrightarrow 0.7744\\
\text{Node } 4: &\quad (0.72-0)^2 \longrightarrow 0.5184\\
\text{Node } 5: &\quad (0.01-0)^2 \longrightarrow 0.0001\\
\text{Node } 6: &\quad (0.64-0)^2 \longrightarrow 0.4096\\
\text{Node } 7: &\quad (0.86-1)^2 \longrightarrow 0.0196\\
\text{Node } 8: &\quad (0.99-0)^2 \longrightarrow 0.9801\\
\text{Node } 9: &\quad (0.63-0)^2 \longrightarrow 0.3969
\end{aligned}
$$

What do you notice about these numbers relative to how correct the model guessed?

Cost Function

- If we give it a 7, it will return random numbers. To change the weights, we need a *cost function*.

$$
\begin{aligned}
\text{Node } 0: &\quad (0.43-0)^2 \longrightarrow 0.1863\\
\text{Node } 1: &\quad (0.28-0)^2 \longrightarrow 0.0784\\
\text{Node } 2: &\quad (0.19-0)^2 \longrightarrow 0.0361\\
\text{Node } 3: &\quad (0.88-0)^2 \longrightarrow 0.7744\\
\text{Node } 4: &\quad (0.72-0)^2 \longrightarrow 0.5184\\
\text{Node } 5: &\quad (0.01-0)^2 \longrightarrow 0.0001\\
\text{Node } 6: &\quad (0.64-0)^2 \longrightarrow 0.4096\\
\text{Node } 7: &\quad (0.86-1)^2 \longrightarrow 0.0196\\
\text{Node } 8: &\quad (0.99-0)^2 \longrightarrow 0.9801\\
\text{Node } 9: &\quad (0.63-0)^2 \longrightarrow 0.3969
\end{aligned}
$$

What do you notice about these numbers relative to how correct the model guessed?

- The *cost* is the sum over these squares $= 3.3999$. We want to minimize this.

Cost Function

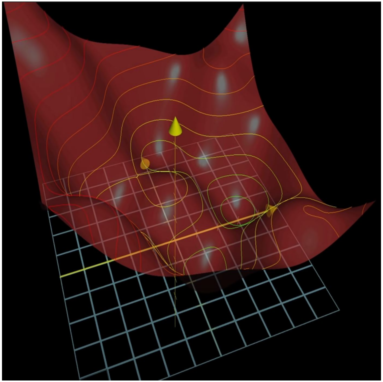
- Find the average cost of all of your training data.

Cost Function

- Find the average cost of all of your training data.
- Observation: Cost functions takes in weights and biases and outputs a single number. It's defined with respect to the training data.

Cost Function

- Find the average cost of all of your training data.
- Observation: Cost functions takes in weights and biases and outputs a single number. It's defined with respect to the training data.
- This is the corresponding "error function" we saw earlier.



This beast of a cost function lives in $\mathbb{R}^{|\{\text{weights}\}|+|\{\text{biases}\}|+1}$

Gradient Descent

- Recall that the gradient of a function gives the direction of the steepest ascent $\Rightarrow -\nabla C$ is the direction of steepest descent.

Gradient Descent

- Recall that the gradient of a function gives the direction of the steepest ascent $\Rightarrow -\nabla C$ is the direction of steepest descent.
- Put **all** of the weights and biases in a column vector, $\vec{v}$, and transform it the following way

$$\vec{v} \mapsto \vec{v} - \nabla C(\vec{v}).$$

Gradient Descent

- Recall that the gradient of a function gives the direction of the steepest ascent $\Rightarrow -\nabla C$ is the direction of steepest descent.
- Put **all** of the weights and biases in a column vector, $\vec{v}$, and transform it the following way

$$\vec{v} \mapsto \vec{v} - \nabla C(\vec{v}).$$

- Then repeat with the new weights and biases. ∇C can be found somewhat efficiently using *back propogation*.
  - A recursive alorithm nudging layers individually instead of the entire thing.

Stochastic Gradient Descent

- In practice, you would instead want to take training data, shuffle it up, and making mini-batches. Then compute a step of the back propogation according to the mini-batch. Repeat for each mini-batch.

Stochastic Gradient Descent

- In practice, you would instead want to take training data, shuffle it up, and making mini-batches. Then compute a step of the back propogation according to the mini-batch. Repeat for each mini-batch.
- This is less efficient as it's not over the entire set, however, this is a major computational speed up and fairly good at approximating.

Stochastic Gradient Descent

- In practice, you would instead want to take training data, shuffle it up, and making mini-batches. Then compute a step of the back propogation according to the mini-batch. Repeat for each mini-batch.
- This is less efficient as it's not over the entire set, however, this is a major computational speed up and fairly good at approximating.
- "It would be like a drunk man stumbling aimlessly down a hill but taking quick steps, rather than a carefully calculating man determining the exact downhill direction of each step, before taking a very slow and careful step in that direction." - Grant Sanderson.

Common Problems

Common Problems

- Bad or incomplete data

Common Problems

- Bad or incomplete data
  - How do you think this affects the model?
  - How/when can we deal with these holes?
  - More structured data $\Rightarrow$ the more even the local minima are with respect to each other.

Common Problems

- Bad or incomplete data
  - How do you think this affects the model?
  - How/when can we deal with these holes?
  - More structured data $\Rightarrow$ the more even the local minima are with respect to each other.
- Overtraining (memorizing vs generalizing)

Common Problems

- Bad or incomplete data
  - How do you think this affects the model?
  - How/when can we deal with these holes?
  - More structured data $\Rightarrow$ the more even the local minima are with respect to each other.
- Overtraining (memorizing vs generalizing)
  - Why is this bad?

Common Problems

- Bad or incomplete data
  - How do you think this affects the model?
  - How/when can we deal with these holes?
  - More structured data $\Rightarrow$ the more even the local minima are with respect to each other.
- Overtraining (memorizing vs generalizing)
  - Why is this bad?
- Obtaining a suffient amount of labeled data
  - *unsupervised learning*

Fun Thoughts

- The model is actually smarter than just guessing!
- Classifying data based on topology of cost function and hidden layers
- Manifold hypothesis
- Fundamental groups and higher homotopy

Thank you!