Introduction to Machine Learning

Owen Oertell

Chamblee Charter High School

2021 年 7 月 11 日

Machine Learning

The application of algorithms which learn automatically through experience.

- Although it sometimes seems like we don't know much about how ML algorithms pick out specific features, a lot is known experimentally and there still is a strong mathematical foundation behind it.

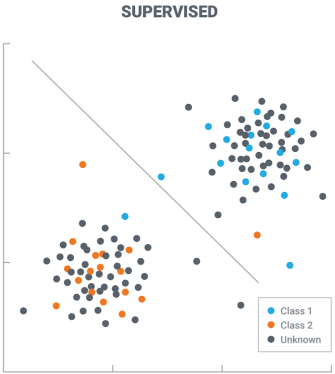
Machine Learning

The application of algorithms which learn automatically through experience.

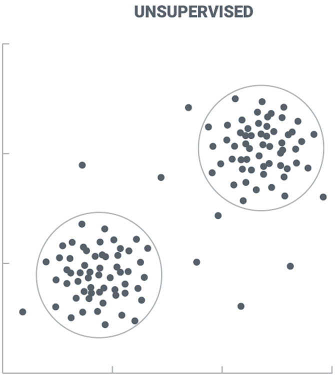
- Although it sometimes seems like we don't know much about how ML algorithms pick out specific features, a lot is known experimentally and there still is a strong mathematical foundation behind it.
- However, the field is still catching up with regard to understanding why models learn the way they do.

1. What is ML?
2. Types of ML Algorithms
3. Core Concepts of ML
4. Neural Nets
5. Backpropagation
6. Example
7. Conclusion

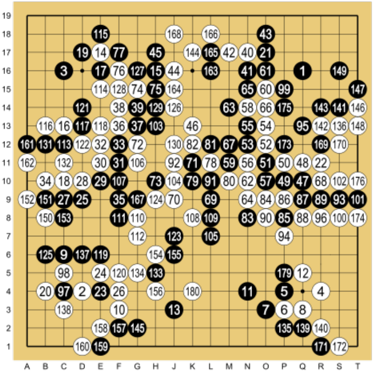
- Supervised Learning



- Supervised Learning
- Unsupervised Learning

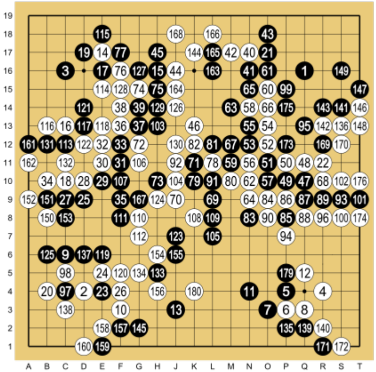


- Supervised Learning
- Unsupervised Learning
- Reinforcement Leaning



Lee Sedol (W) vs AlphaGo (B) - Game 4
177 at 51 178 at 57

- Supervised Learning
- Unsupervised Learning
- Reinforcement Leaning
- Move 37 and 78!



Lee Sedol (W) vs AlphaGo (B) - Game 4
177 at 51 178 at 57

- Supervised Learning
- Unsupervised Learning
- Reinforcement Leaning
- Move 37 and 78!
- We are going to focus on supervised learning for the remainder of this presentation

1. What is ML?
2. Types of ML Algorithms
3. **Core Concepts of ML**
4. Neural Nets
5. Backpropagation
6. Example
7. Conclusion

- For the first few slides I am going to use some terms like *regression*, *neural network* but don't worry, I will explain them in a second.

- For the first few slides I am going to use some terms like *regression*, *neural network* but don't worry, I will explain them in a second.

Features

The input values to neural network or regression equation. Denoted as $x_1, x_2 ... x_N$

- For the first few slides I am going to use some terms like *regression*, *neural network* but don't worry, I will explain them in a second.

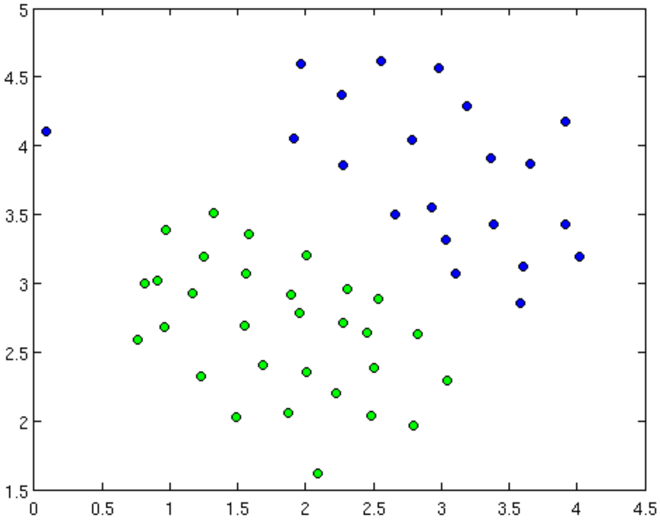
Features

The input values to neural network or regression equation. Denoted as $x_1, x_2...x_N$

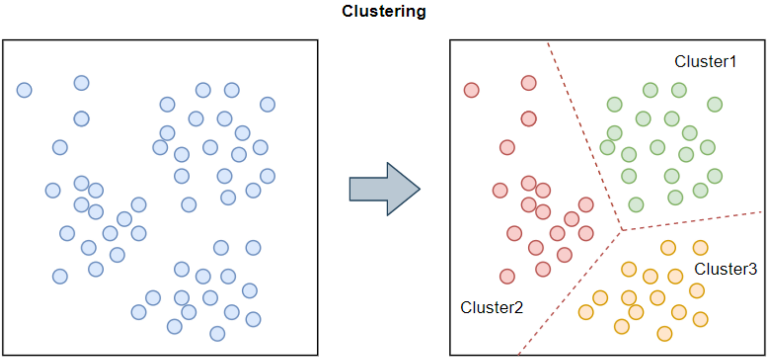
Label

The value that we are predicting. Denoted as y

Labeled Data Example



Unlabeled Data Example



Problems in Machine Learning

- Classification (Supervised)
  - Is this email spam or not spam?
  - What is the type of car in this picture?
  - Is this person wearing a mask or not wearing a mask? (Thanks COVID-19)

Problems in Machine Learning

- Classification (Supervised)
    - Is this email spam or not spam?
    - What is the type of car in this picture?
    - Is this person wearing a mask or not wearing a mask? (Thanks COVID-19)
- Clustering (Unsupervised)
    - How many clusters or categories are in this dataset?

Problems in Machine Learning

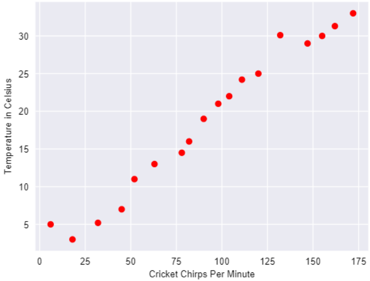
- Classification (Supervised)
  - Is this email spam or not spam?
  - What is the type of car in this picture?
  - Is this person wearing a mask or not wearing a mask? (Thanks COVID-19)
- Clustering (Unsupervised)
  - How many clusters or categories are in this dataset?
- Solve a Problem (Reinforcement)
  - What is the best move in Go for this given situation?
  - How can you drive this car?
  - Turning text into parseable computer actions.

Problems in Machine Learning

- Classification (Supervised)
  - Is this email spam or not spam?
  - What is the type of car in this picture?
  - Is this person wearing a mask or not wearing a mask? (Thanks COVID-19)
- Clustering (Unsupervised)
  - How many clusters or categories are in this dataset?
- Solve a Problem (Reinforcement)
  - What is the best move in Go for this given situation?
  - How can you drive this car?
  - Turning text into parseable computer actions.
- Regression (Supervised)
  - What is the estimated price of the house?
  - How much will this flight cost?
  - What score will they get on the test?

Linear Regression Explained

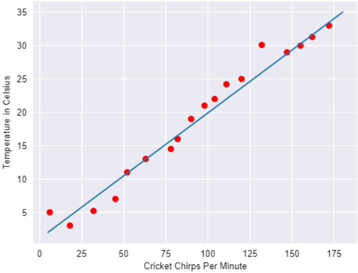
- Linear regression is the modeling of the relationship between two variables.



Graphs courtesy of Google.

Linear Regression Explained

- Linear regression is the modeling of the relationship between two variables.
- Here, the regression line, sometimes also called the *Least Squares Regression Line* (LSRL) seems to fit the data pretty well.



Graphs courtesy of Google.

Linear Regression Explained

- We know that lines take the form $y = a + bx$ (or more commonly written $y = mx + b$).

Linear Regression Explained

- We know that lines take the form $y = a + bx$ (or more commonly written $y = mx + b$).
- Or as we will see, in the language of machine learning, $y' = b + w_1 x_1$.

Linear Regression Explained

- We know that lines take the form $y = a + bx$ (or more commonly written $y = mx + b$).
- Or as we will see, in the language of machine learning, $y' = b + w_1 x_1$.
- The w_i, is called a **weight**, and essentially is a measure of the influence that this feature has on the point we are predicting.

Linear Regression Explained

- We know that lines take the form $y = a + bx$ (or more commonly written $y = mx + b$).
- Or as we will see, in the language of machine learning, $y' = b + w_1 x_1$.
- The w_i, is called a **weight**, and essentially is a measure of the influence that this feature has on the point we are predicting.
- b is the bias

Linear Regression Explained

- We know that lines take the form $y = a + bx$ (or more commonly written $y = mx + b$).
- Or as we will see, in the language of machine learning, $y' = b + w_1 x_1$.
- The w_i, is called a **weight**, and essentially is a measure of the influence that this feature has on the point we are predicting.
- b is the bias
- Since data is never perfect, we try and create the "line of best fit." Or in other words, closest to all of the points.

Linear Regression Explained

- We know that lines take the form $y = a + bx$ (or more commonly written $y = mx + b$).
- Or as we will see, in the language of machine learning, $y' = b + w_1 x_1$.
- The w_i, is called a **weight**, and essentially is a measure of the influence that this feature has on the point we are predicting.
- b is the bias
- Since data is never perfect, we try and create the "line of best fit." Or in other words, closest to all of the points.
- It allows us to make predictions about for a given x what the y would be.

Linear Regression Explained

- We know that lines take the form $y = a + bx$ (or more commonly written $y = mx + b$).
- Or as we will see, in the language of machine learning, $y' = b + w_1 x_1$.
- The w_i, is called a **weight**, and essentially is a measure of the influence that this feature has on the point we are predicting.
- b is the bias
- Since data is never perfect, we try and create the "line of best fit." Or in other words, closest to all of the points.
- It allows us to make predictions about for a given x what the y would be.
- There are some fairly easy to use statistical methods for finding these lines. So... that concludes my presentation then!

But Wait!

This becomes much more complicated in N-dimensions

The Crux of Machine Learning

- Machine learning is all about iterating through a **ton** of examples and learning from them.

The Crux of Machine Learning

- Machine learning is all about iterating through a **ton** of examples and learning from them.
- It starts with one prediction, updates that prediction when it sees new data, and makes adjustments on if this new prediction is more or less accurate.

The Crux of Machine Learning

- Machine learning is all about iterating through a **ton** of examples and learning from them.
- It starts with one prediction, updates that prediction when it sees new data, and makes adjustments on if this new prediction is more or less accurate.
- This is called *gradient descent*. But we will talk about this in a bit.

Training and Loss

- I've already mentioned that a model makes adjustments based on new predictions.

Training and Loss

- I've already mentioned that a model makes adjustments based on new predictions.
- But how do we tell if the model becomes more or less accurate?

Training and Loss

- I've already mentioned that a model makes adjustments based on new predictions.
- But how do we tell if the model becomes more or less accurate?

Loss

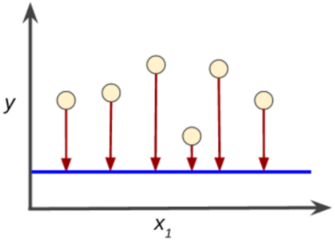
Loss is a measure of the difference between the value the model predicts (y') and the actual value (y).

Training and Loss

- I've already mentioned that a model makes adjustments based on new predictions.
- But how do we tell if the model becomes more or less accurate?

Loss

Loss is a measure of the difference between the value the model predicts (y') and the actual value (y).



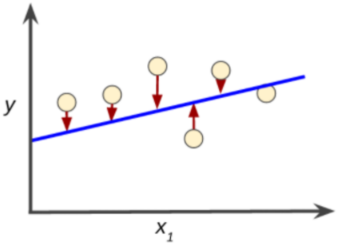
Graphs courtesy of Google.

Training and Loss

- I've already mentioned that a model makes adjustments based on new predictions.
- But how do we tell if the model becomes more or less accurate?

Loss

Loss is a measure of the difference between the value the model predicts (y') and the actual value (y).



Graphs courtesy of Google.

Loss Equation

- One problem that can occur is loss can be either negative or positive.

Loss Equation

- One problem that can occur is loss can be either negative or positive.
- This poses an issue if you are trying to minimize loss. Because $-100 < 0$ but no loss (perfect prediction) is better than -100 loss!

Loss Equation

- One problem that can occur is loss can be either negative or positive.
- This poses an issue if you are trying to minimize loss. Because $-100 < 0$ but no loss (perfect prediction) is better than -100 loss!
- Or put more formally:

$$\min_{\lambda\,:\,\mathbb{R}^n \to \mathbb{R}} \sum_{i=1}^{n} (y_i - \lambda(x)) = -\infty$$

Loss Equation

- One problem that can occur is loss can be either negative or positive.
- This poses an issue if you are trying to minimize loss. Because $-100 < 0$ but no loss (perfect prediction) is better than -100 loss!
- Or put more formally:

$$\min_{\lambda : \mathbb{R}^n \to \mathbb{R}} \sum_{i=1}^{n} (y_i - \lambda(x)) = -\infty$$

- Where λ represents the neural network with n inputs and 1 output.

Loss Equation

- One problem that can occur is loss can be either negative or positive.
- This poses an issue if you are trying to minimize loss. Because $-100 < 0$ but no loss (perfect prediction) is better than -100 loss!
- Or put more formally:

$$\min_{\lambda : \mathbb{R}^n \to \mathbb{R}} \sum_{i=1}^{n} (y_i - \lambda(x)) = -\infty$$

- Where λ represents the neural network with n inputs and 1 output.
- But we don't want loss to be $-\infty$! We want loss to be 0.

A New Loss Equation

- In order to combat this, one method (the one we will use) is called **mean squared error (MSE)**.

A New Loss Equation

- In order to combat this, one method (the one we will use) is called **mean squared error (MSE)**.
- There also are other techniques like taking the absolute value of loss but the most common is MSE.

A New Loss Equation

- In order to combat this, one method (the one we will use) is called **mean squared error (MSE)**.
- There also are other techniques like taking the absolute value of loss but the most common is MSE.

Mean Squared Error (MSE)

MSE is the average squared loss per example over the entire dataset. It is represented by the equation:

$$MSE = \frac{1}{N} \sum_{(x,y) \in D} (y - \lambda(x))^2$$

- This equation is only for one feature, x, but predictions often take into account many other features.

Training and Loss Minimization

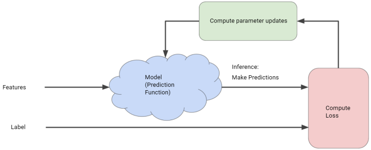
- As I have said many times, a model learns from trial and error.

Training and Loss Minimization

- As I have said many times, a model learns from trial and error.
- This is called an **iterative** process.

Training and Loss Minimization

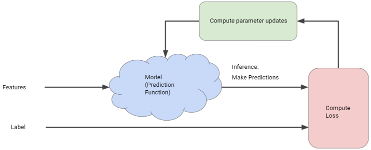
- As I have said many times, a model learns from trial and error.
- This is called an **iterative** process.



Graphs courtesy of Google.

Training and Loss Minimization

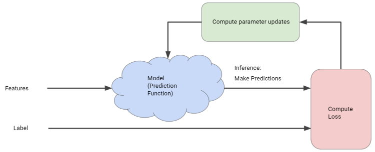
- As I have said many times, a model learns from trial and error.
- This is called an **iterative** process.
- Tweaking of the model to be more accurate.



Graphs courtesy of Google.

Training and Loss Minimization

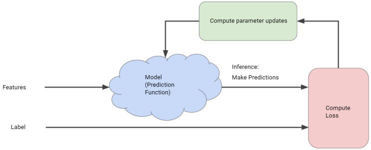
- As I have said many times, a model learns from trial and error.
- This is called an **iterative** process.
- Tweaking of the model to be more accurate.
- We keep doing this until loss doesn't change much anymore. At that point, the model has **converged**.



Graphs courtesy of Google.

Training and Loss Minimization

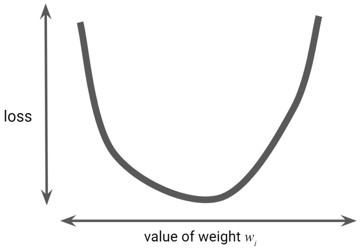
- As I have said many times, a model learns from trial and error.
- This is called an ***iterative*** process.
- Tweaking of the model to be more accurate.
- We keep doing this until loss doesn't change much anymore. At that point, the model has **converged**.
- But how do we know how to tweak the model?



Graphs courtesy of Google.

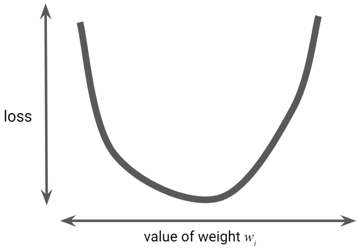
Loss Minimization and Gradient Descent

- The graph of a weight vs loss is convex up.



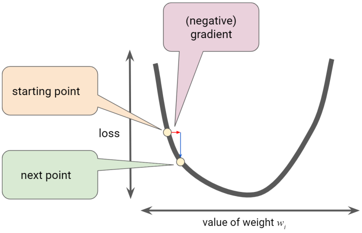
Loss Minimization and Gradient Descent

- The graph of a weight vs loss is convex up.
- We adjust this model by minimizing loss, or in other words descending the gradient.



Loss Minimization and Gradient Descent

- The graph of a weight vs loss is convex up.
- We adjust this model by minimizing loss, or in other words descending the gradient.
- All we need to do is estimate the derivative at that point and move to the lower side.



Graphs courtesy of Google.

Hyperparameters

- One factor we control is how big of a jump do we take between the starting point (or previous point) and the new point. This is called the **learning rate**.

Hyperparameters

- One factor we control is how big of a jump do we take between the starting point (or previous point) and the new point. This is called the **learning rate**.
- When choosing the learning rate, if it is too big, then you can overshoot and start to oscillate, jumping over the global minimum. Too small, and it takes forever to converge.

Hyperparameters

- One factor we control is how big of a jump do we take between the starting point (or previous point) and the new point. This is called the **learning rate**.
- When choosing the learning rate, if it is too big, then you can overshoot and start to oscillate, jumping over the global minimum. Too small, and it takes forever to converge.
- Another factor we control is the **batch**. This is the number of examples used to update the weights. Although in small datasets the batch size can be equal to N, often we just look at a few examples.

Hyperparameters

- One factor we control is how big of a jump do we take between the starting point (or previous point) and the new point. This is called the **learning rate**.
- When choosing the learning rate, if it is too big, then you can overshoot and start to oscillate, jumping over the global minimum. Too small, and it takes forever to converge.
- Another factor we control is the **batch**. This is the number of examples used to update the weights. Although in small datasets the batch size can be equal to N, often we just look at a few examples.
- Looking at, and updating weights based off of a randomly selected single example is called **stochastic gradient descent (SGD)**.

Hyperparameters

- One factor we control is how big of a jump do we take between the starting point (or previous point) and the new point. This is called the **learning rate**.
- When choosing the learning rate, if it is too big, then you can overshoot and start to oscillate, jumping over the global minimum. Too small, and it takes forever to converge.
- Another factor we control is the **batch**. This is the number of examples used to update the weights. Although in small datasets the batch size can be equal to N, often we just look at a few examples.
- Looking at, and updating weights based off of a randomly selected single example is called **stochastic gradient descent (SGD)**.
- However this often is too extreme and yields noisy results. A combination approach is instead taken.

Hyperparameters

- This is called **mini-batch stochastic gradient descent (mini-batch SGD)** and is a very common method. It uses 10-1000 examples to calculate the gradient and update the model.

Hyperparameters

- This is called **mini-batch stochastic gradient descent (mini-batch SGD)** and is a very common method. It uses 10-1000 examples to calculate the gradient and update the model.
- We also are able to control the number of **epochs**, or the number of full passes through the data, as well as the number of **steps per epoch** which is the number of batches before an epoch is considered complete.

Epoch? I've Seen That Before!

- Depending on how much you do with computers you may have heard of something called *UNIX time*.

Epoch? I've Seen That Before!

- Depending on how much you do with computers you may have heard of something called *UNIX time*.
- It is the time, in milliseconds since January 1st, 1970 (also known as the Unix epoch!)

Epoch? I've Seen That Before!

- Depending on how much you do with computers you may have heard of something called *UNIX time*.
- It is the time, in milliseconds since January 1st, 1970 (also known as the Unix epoch!)
- Does this relate to ML?

Epoch? I've Seen That Before!

- Depending on how much you do with computers you may have heard of something called *UNIX time*.
- It is the time, in milliseconds since January 1st, 1970 (also known as the Unix epoch!)
- Does this relate to ML?
- Nope! Just something interesting that I wanted to share.

Quiz Time!

- True or False: A model will converge at the same minima every time.

Quiz Time!

- True or False: A model will converge at the same minima every time.
- `https://www.youtube.com/watch?v=vWFjqgb-ylQ`

Quiz Time!

- True or False: A model will converge at the same minima every time.
- `https://www.youtube.com/watch?v=vWFjqgb-ylQ`
- False.

Quiz Time!

- True or False: A model will converge at the same minima every time.
- `https://www.youtube.com/watch?v=vWFjqgb-ylQ`
- False.
- True or False: Too high of a learning rate will cause oscillations.

Quiz Time!

- True or False: A model will converge at the same minima every time.
- `https://www.youtube.com/watch?v=vWFjqgb-ylQ`
- False.
- True or False: Too high of a learning rate will cause oscillations.
- True.

Metrics

- When training a model, it is often good to track metrics.

Metrics

- When training a model, it is often good to track metrics.
- For example, tracking the **mean squared error** is common.

Metrics

- When training a model, it is often good to track metrics.
- For example, tracking the **mean squared error** is common.
- However, we also often track **accuracy** if we are running classification algorithms as well as **false positive rate.**.

Metrics

- When training a model, it is often good to track metrics.
- For example, tracking the **mean squared error** is common.
- However, we also often track **accuracy** if we are running classification algorithms as well as **false positive rate.**.
- It is important to note however that one metric does not tell the whole story.

Metrics

- When training a model, it is often good to track metrics.
- For example, tracking the **mean squared error** is common.
- However, we also often track **accuracy** if we are running classification algorithms as well as **false positive rate.**.
- It is important to note however that one metric does not tell the whole story.
- For example, in classification problems, **accuracy** alone can be misleading and one should also keep track of the F_1 score for example.

Metrics

- When training a model, it is often good to track metrics.
- For example, tracking the **mean squared error** is common.
- However, we also often track **accuracy** if we are running classification algorithms as well as **false positive rate.**.
- It is important to note however that one metric does not tell the whole story.
- For example, in classification problems, **accuracy** alone can be misleading and one should also keep track of the F_1 score for example.
- We will get to see an example of loss going down soon!

Training, Validation, and Overfitting

- **Overfitting** is what happens when the model fits the training data too much and then starts to make incorrect predictions.

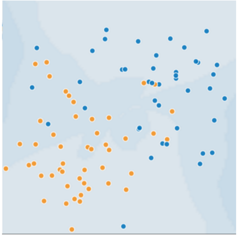


Figure 1. Sick (blue) and healthy (orange) trees.

Training, Validation, and Overfitting

- **Overfitting** is what happens when the model fits the training data too much and then starts to make incorrect predictions.

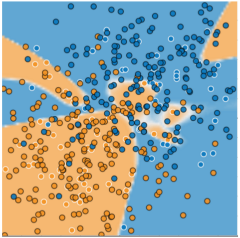
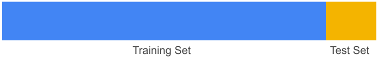


Figure 3. The model did a bad job predicting new data.

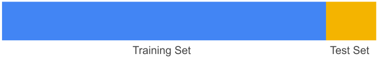
Training, Validation, and Overfitting

- **Overfitting** is what happens when the model fits the training data too much and then starts to make incorrect predictions.
- To determine how well the model is able to predict data that it wasn't trained on, we split the dataset into a **training** and a **validation** set.



Training, Validation, and Overfitting

- **Overfitting** is what happens when the model fits the training data too much and then starts to make incorrect predictions.
- To determine how well the model is able to predict data that it wasn't trained on, we split the dataset into a **training** and a **validation** set.
- 80% training data and 20% validation data.



Questions

- The next section is going to be about activation functions and neurons themselves—what I consider to be the coolest but also most complicated topic.

Questions

- The next section is going to be about activation functions and neurons themselves—what I consider to be the coolest but also most complicated topic.
- So before we begin, lets make sure that everyone understands everything in this section because if you don't have a good grasp, the next section won't make sense.

Questions

- The next section is going to be about activation functions and neurons themselves—what I consider to be the coolest but also most complicated topic.
- So before we begin, lets make sure that everyone understands everything in this section because if you don't have a good grasp, the next section won't make sense.
- Questions?

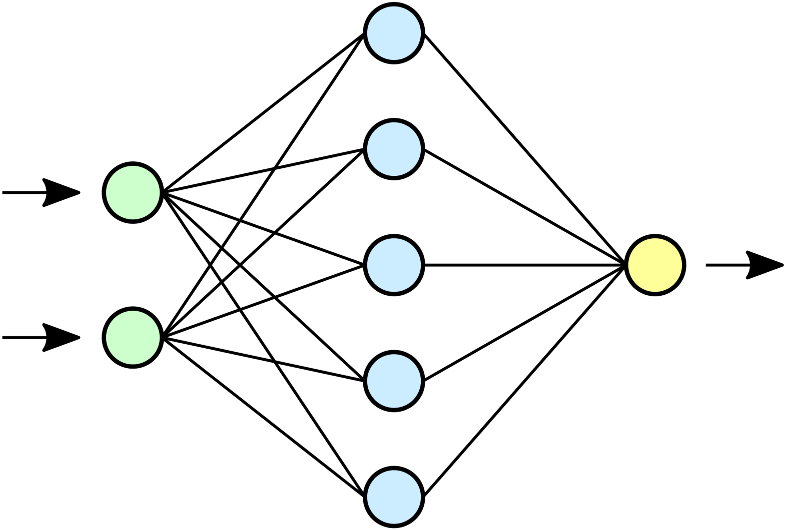
Neural Networks

- Now that we have a *basic* understanding of machine learning, we can start to harness its power.

Neural Networks

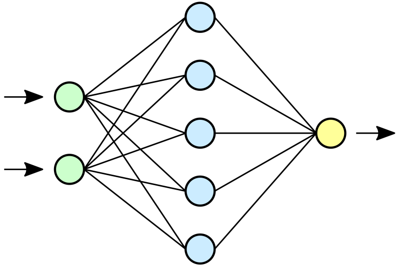
- Now that we have a *basic* understanding of machine learning, we can start to harness its power.
- However, I will have to limit the scope even more, so we are only going to focus on feed forward networks. We can also briefly talk about how convolutional neural networks work at the end if there is interest.

A Network



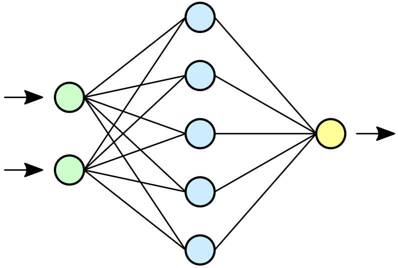
Parts of a Network

- **Input Layer**: Quantitative versions of features.



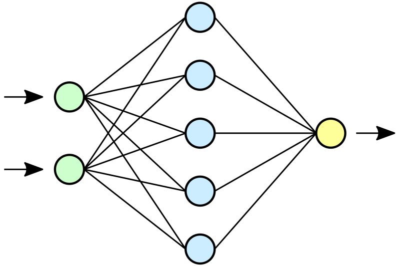
Parts of a Network

- **Input Layer**: Quantitative versions of features.
- **Hidden Layer(s)**: Layers where non-linear transformations occur.



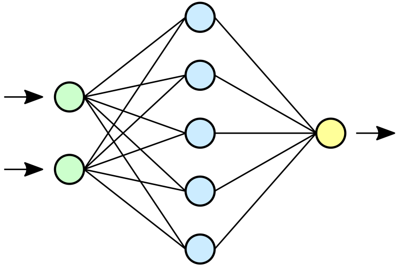
Parts of a Network

- **Input Layer**: Quantitative versions of features.
- **Hidden Layer(s)**: Layers where non-linear transformations occur.
- **Output Layer**: The output; used to calculate loss and predict models.

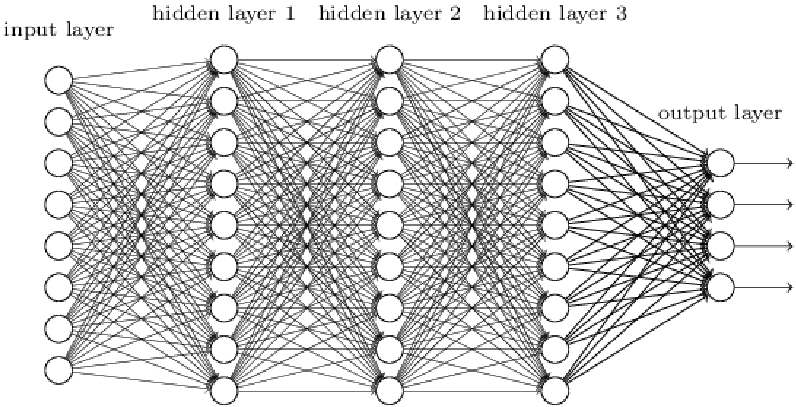


Parts of a Network

- **Input Layer**: Quantitative versions of features.
- **Hidden Layer(s)**: Layers where non-linear transformations occur.
- **Output Layer**: The output; used to calculate loss and predict models.
- Note: Non-linear transformations can happen between input and hidden as well as hidden and output.



A Deep Neural Network Example



What do those lines mean?

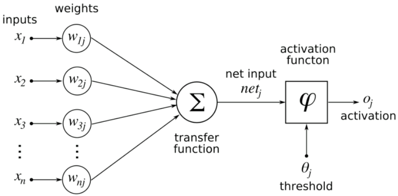
- Those lines represent the inputs.

What do those lines mean?

- Those lines represent the inputs.
- Every neuron in one layer becomes an input for the neurons in the next layer.

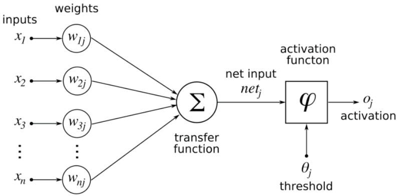
What do those lines mean?

- Those lines represent the inputs.
- Every neuron in one layer becomes an input for the neurons in the next layer.
- Well... what happens in a neuron?



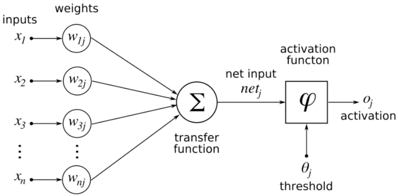
What do those lines mean?

- Those lines represent the inputs.
- Every neuron in one layer becomes an input for the neurons in the next layer.
- Well... what happens in a neuron?
- This looks complicated! But it's not.

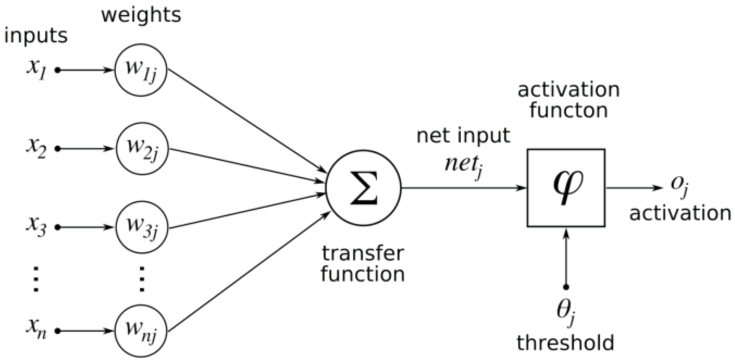


What do those lines mean?

- Those lines represent the inputs.
- Every neuron in one layer becomes an input for the neurons in the next layer.
- Well... what happens in a neuron?
- This looks complicated! But it's not.
- Lets break this down..



A Closer Look look at a Neuron



$$f(x_1, x_2, x_3, ..., x_n) = a(\sum_{0}^{n} x_i w_i + b)$$

where a is an activation function

Activation functions

- Activation functions can essentially be any nonlinear function.

Activation functions

- Activation functions can essentially be any nonlinear function.
- There are two main activation functions

Activation functions

- Activation functions can essentially be any nonlinear function.
- There are two main activation functions
  - Rectified Linear Unit (ReLU)

$$R(z) = max(0, z)$$

Activation functions

- Activation functions can essentially be any nonlinear function.
- There are two main activation functions
  - Rectified Linear Unit (ReLU)

$$R(z) = max(0, z)$$

  - Sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

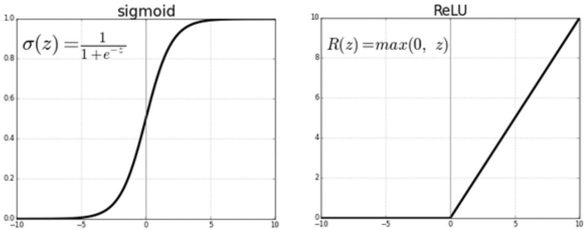
Activation functions

- Activation functions can essentially be any nonlinear function.
- There are two main activation functions
  - Rectified Linear Unit (ReLU)

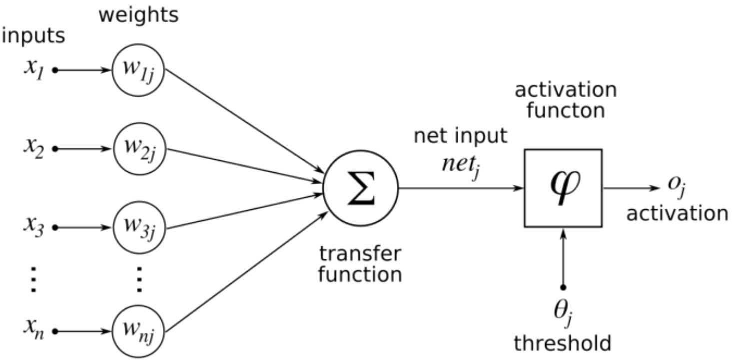
$$R(z) = max(0, z)$$

  - Sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Looking Back At the Diagram



$$f(x_1, x_2, x_3, ..., x_n) = a(\sum_{0}^{n} x_i w_i + b)$$

where a is an activation function

Backpropagation

- We saw that it seemed relatively trivial to update the weights on a single node.

Backpropagation

- We saw that it seemed relatively trivial to update the weights on a single node.
- But what about if there are multiple nodes? How about if there are multiple hidden layers?

Backpropagation

- We saw that it seemed relatively trivial to update the weights on a single node.
- But what about if there are multiple nodes? How about if there are multiple hidden layers?
- Well, you probably guessed it from the title of this frame but the answer is **backpropagation** (RMSprop also would suffice, however that is less common).

Backpropagation

- What exactly is backpropagation then?

Backpropagation

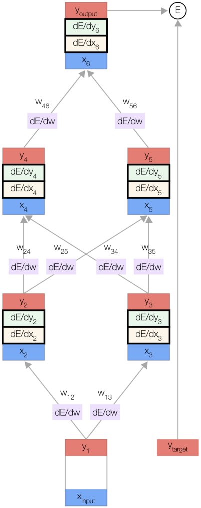
- What exactly is backpropagation then?
- Well, backpropagation is an algorithm which allows one to calculate the gradient (∇) of the loss function for each weight.

Backpropagation

- What exactly is backpropagation then?
- Well, backpropagation is an algorithm which allows one to calculate the gradient (∇) of the loss function for each weight.
- Now let's take a look at the math behind backpropagation (backprop).

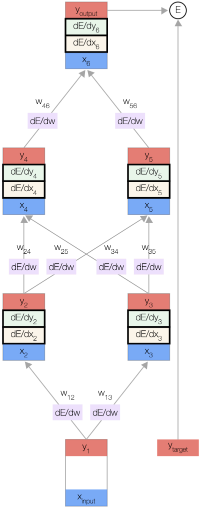
Backpropagation Explained

- The first step to backpropagation is in fact forward propagation.



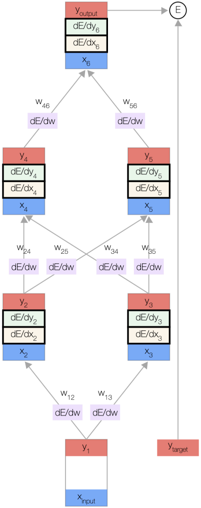
Backpropagation Explained

- The first step to backpropagation is in fact forward propagation.
- If you recall, this means applying the formula of each neuron: $N(\mathbf{x}) = a(\sum_0^n \mathbf{x}_i w_i + b)$



Backpropagation Explained

- The first step to backpropagation is in fact forward propagation.
- If you recall, this means applying the formula of each neuron: $N(\mathbf{x}) = a(\sum_0^n \mathbf{x}_i w_i + b)$
- Then we feed that value into all of the next neurons and repeat!



Backpropagation Explained

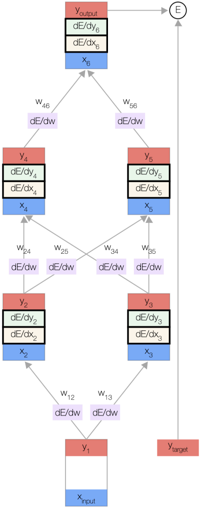
Now, we want to calculate the derivative of the error with respect to each specific weight. We will do this by first defining the function for error:

$$E = \frac{1}{2}(y_{output} - y_{target})^2$$

We can then take the derivative of the error yielding:

$$\frac{\partial E}{\partial y_{output}} = y_{output} - y_{target}$$

Remember the power rule! From now on we will refer to this as $\frac{\partial E}{\partial y}$.



Backpropagation Explained

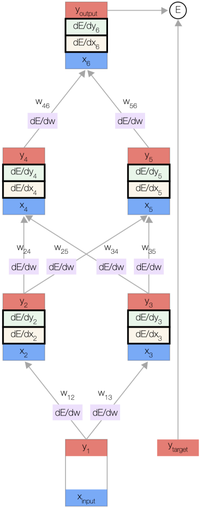
We can now recall the chain rule $\frac{d}{dx}\left[f(g(x))\right] = f'(g(x))g'(x)$ in order to get the derivative of E with respect to x. This would allow us to get closer to the weight because if we recall the equation for a neuron:

$$N(\mathbf{x}) = \sigma(\sum \mathbf{x}_i w_i + b)$$

Therefore, to get $\frac{\partial E}{\partial x}$ we can say:

$$\frac{\partial E}{\partial \mathbf{x}} = \frac{dy}{d\mathbf{x}} \cdot \frac{\partial E}{\partial y}$$

So what function do we pass all the output through for the neuron? (i.e. function for $\frac{dy}{d\mathbf{x}}$)

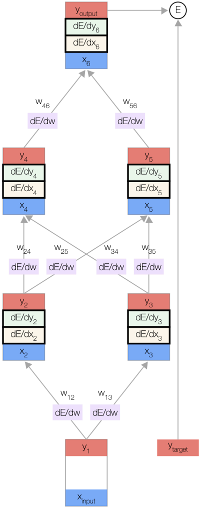


Backpropagation Explained

The activation function! You may have noticed that I replaced a with σ. That is because in this example we are going to assume that the activation function in this layer is a sigmoid function. Thus:

$$\frac{\partial E}{\partial \mathbf{x}} = \frac{dy}{d\mathbf{x}} \cdot \frac{\partial E}{\partial y} = \frac{d\sigma}{d\mathbf{x}} \cdot \frac{\partial E}{\partial y}$$

Bonus: Why did I use sigmoid instead of ReLU for this example?



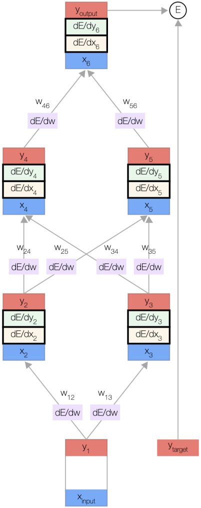
Backpropagation Explained

Now, we have $\frac{\partial E}{\partial \mathbf{x}}$, but we want $\frac{\partial E}{\partial w_{ij}}$. Luckily, we can call on our old friend chain rule to help us out again. Therefore:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial \mathbf{x}}{\partial w_{ij}} \cdot \frac{\partial E}{\partial \mathbf{x}}$$

But what is $\frac{\partial \mathbf{x}}{\partial w_{ij}}$? Well lets recall that the neuron with respect to the inside (ignoring the activation function since we are using the chain rule) is just $(\sum \mathbf{x}_i \cdot w_i) + b$. Therefore, removing the *constant* b:

$$\frac{\partial \mathbf{x}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum \mathbf{x}_i w_i\right) = \mathbf{x}$$



Backpropagation Explained

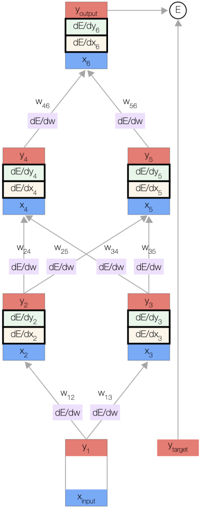
Thus,

$$\frac{\partial E}{\partial w_{ij}} = \mathbf{x}\frac{\partial E}{\partial \mathbf{x}}$$

We can now get the derivative of the error with respect to the previous output (and then we can repeat!)

$$\frac{\partial E}{\partial y^{(L-1)}} = \sum \frac{\partial \mathbf{x}_i}{\partial y_i} \cdot \frac{\partial E}{\partial \mathbf{x}_j} = \sum w_{ij} \frac{\partial E}{\partial \mathbf{x}_j}$$

The gradient for the biases are calculated similarly.

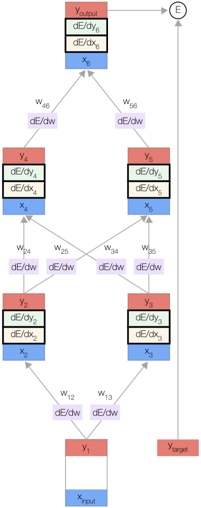


Backpropagation Summary

If we combine all the work from previous slides, we get:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial w_{ij}}$$

Don't worry if this seems confusing. Backpropagation is one of the hardest parts of machine learning.



Bringing It All Together

- Neural Networks consist of a number of layers of nodes where each has a special **weight** for each input and a **bias**.

Bringing It All Together

- Neural Networks consist of a number of layers of nodes where each has a special **weight** for each input and a **bias**.
- The value is then put through an **activation function** where a nonlinear transformation changes the value.

Bringing It All Together

- Neural Networks consist of a number of layers of nodes where each has a special **weight** for each input and a **bias**.
- The value is then put through an **activation function** where a nonlinear transformation changes the value.
- Activation functions are what gives a network the ability to learn nonlinear patterns.

Bringing It All Together

- Neural Networks consist of a number of layers of nodes where each has a special **weight** for each input and a **bias**.
- The value is then put through an **activation function** where a nonlinear transformation changes the value.
- Activation functions are what gives a network the ability to learn nonlinear patterns.
- The final value(s) in the output layer then constitutes the prediction

Bringing It All Together

- Neural Networks consist of a number of layers of nodes where each has a special **weight** for each input and a **bias**.
- The value is then put through an **activation function** where a nonlinear transformation changes the value.
- Activation functions are what gives a network the ability to learn nonlinear patterns.
- The final value(s) in the output layer then constitutes the prediction
- **Backpropagation** then updates all of the weights based on **loss** (often MSE) which is known as **Stochastic Gradient Descent**.

Lets Do It!

- This may seem like a lot to program—and it is!

Lets Do It!

- This may seem like a lot to program—and it is!
- Luckily, we can use a very popular open source library called `tensorflow` to make this process much easier!

Lets Do It!

- This may seem like a lot to program—and it is!
- Luckily, we can use a very popular open source library called `tensorflow` to make this process much easier!
- The example code we are going to work with is going to be in python.

Lets Do It!

- This may seem like a lot to program—and it is!
- Luckily, we can use a very popular open source library called `tensorflow` to make this process much easier!
- The example code we are going to work with is going to be in python.
- If you are not familiar with python, or any other programming language, this portion may be a little hard to follow but I will try to do my best to make this understandable.

The Dataset

- So... what dataset are we going to use?

The Dataset

- So... what dataset are we going to use?
- I chose to use the *concrete strength* dataset.

The Dataset

- So... what dataset are we going to use?
- I chose to use the *concrete strength* dataset.
- But wait? I don't know anything about concrete!

The Dataset

- So... what dataset are we going to use?
- I chose to use the *concrete strength* dataset.
- But wait? I don't know anything about concrete!
- Neither do I! Whats so good about this that it shows you the true power of machine learning—to predict labels from features where you have no idea what they do.

The Dataset

- So... what dataset are we going to use?
- I chose to use the *concrete strength* dataset.
- But wait? I don't know anything about concrete!
- Neither do I! Whats so good about this that it shows you the true power of machine learning—to predict labels from features where you have no idea what they do.
- Even better, we will be able to get a very accurate prediction using only knowledge that we have just learned!

The Dataset

- So... what dataset are we going to use?
- I chose to use the *concrete strength* dataset.
- But wait? I don't know anything about concrete!
- Neither do I! Whats so good about this that it shows you the true power of machine learning—to predict labels from features where you have no idea what they do.
- Even better, we will be able to get a very accurate prediction using only knowledge that we have just learned!
- Just one more thing before we begin: I am going to use what is called **Google Collab Notebook**. Think of it as a way to run slices of python code while keeping the variables from prior runs alive.

The Dataset

- So... what dataset are we going to use?
- I chose to use the *concrete strength* dataset.
- But wait? I don't know anything about concrete!
- Neither do I! Whats so good about this that it shows you the true power of machine learning—to predict labels from features where you have no idea what they do.
- Even better, we will be able to get a very accurate prediction using only knowledge that we have just learned!
- Just one more thing before we begin: I am going to use what is called **Google Collab Notebook**. Think of it as a way to run slices of python code while keeping the variables from prior runs alive.
- You can also find and modify this notebook here: `https://github.com/Owen-Oertell/introToMLConcretePrediction`.

Lets Begin!

- First, let's create a google colab notebook.

Lets Begin!

- First, let's create a google colab notebook.
- Head over to `https://colab.research.google.com/` and click New notebook.

Lets Begin!

- First, let's create a google colab notebook.
- Head over to `https://colab.research.google.com/` and click New notebook.
- If you miss a step, all the code is also here: `https://bit.ly/3jUQLBT`. However, I encourage that you type each line because it will greatly improve your understanding of what the code does.

Lets Begin!

- First, let's create a google colab notebook.
- Head over to `https://colab.research.google.com/` and click New notebook.
- If you miss a step, all the code is also here: `https://bit.ly/3jUQLBT`. However, I encourage that you type each line because it will greatly improve your understanding of what the code does.
- When you need to read the csv, you will want to put in the URL for the data here: `https://bit.ly/3AJE02X`.

1. What is ML?
2. Types of ML Algorithms
3. Core Concepts of ML
4. Neural Nets
5. Backpropagation
6. Example
7. Conclusion

Going Forward: What I Didn't Cover

- ML is a vast field and today we didn't even come close to scratching the surface. So, where next? (not in order)
  - Techniques for preventing overfitting.
  - Feature crossing for weirder non-linear data.
  - Encoding of Qualitative data (One-Hot Vectors)
  - Classification
  - Regularization
  - Convolutional Neural Networks (CNNs)
  - Optimization
  - Reinforcement Learning
  - Anomaly Detection

Where to start learning more? Check out the Machine Learning Crash Course by Google. It contains what was covered in this presentation and much more!

FIN.