The following slides are modified versions from a presentation given by the Barnes Group in 2022

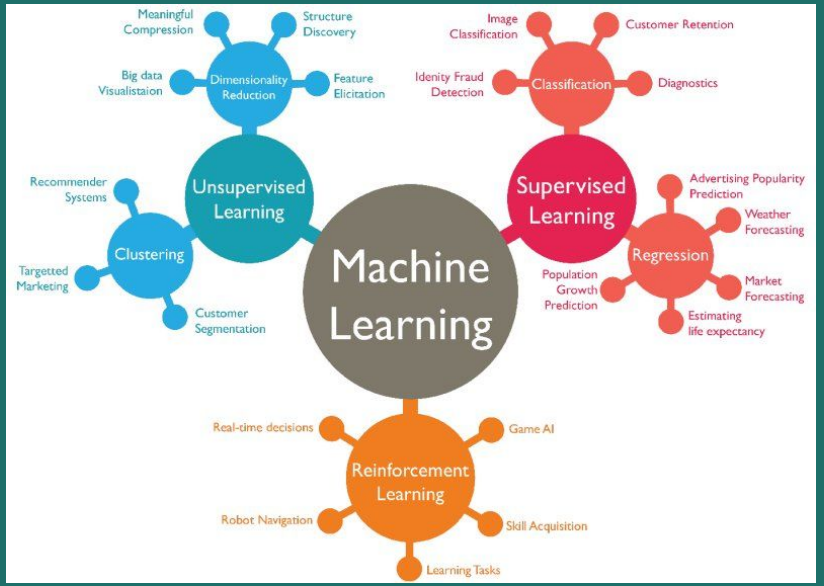
Elizabeth Barnes. (2022). eabarnes1010/ml_tutorial_csu: version 1.0 of tutorial material (v1.0). Zenodo. https://doi.org/10.5281/zenodo.6686879

An Introduction to Artificial Neural Networks

Lecturer: Kirsten Mayer (she/her)

12 May 2026

Birds Eye View



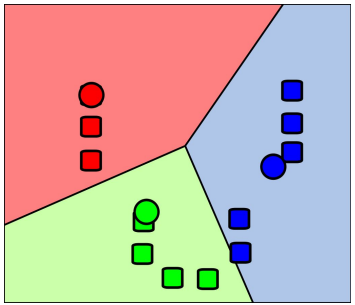
What is Machine Learning?

Field of study that gives computers the ability to learn **without being explicitly programmed**.
Defined by Arthur Samuel (1959)

Practically speaking:

- Techniques for fitting data.
- It is not restricted by a single or a set of simple mathematical expressions.

Two Types of machine learning

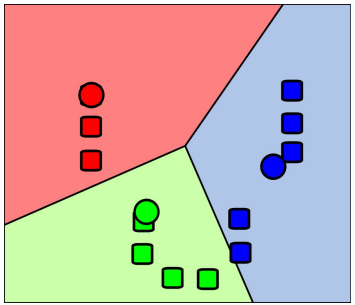


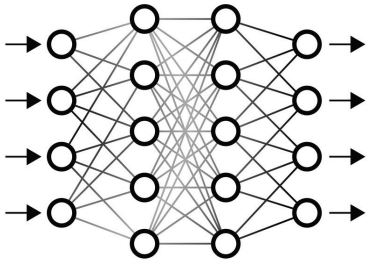
Unsupervised learning

looks for previously undetected patterns in a data set with no pre-existing labels and with a minimum of human supervision, e.g.

- principal component analysis
- clustering methods (k-means, self-organizing maps)
- autoencoders

Two Types of machine learning





Unsupervised learning

looks for previously undetected patterns in a data set with no pre-existing labels and with a minimum of human supervision, e.g.

- principal component analysis
- clustering methods (k-means, self-organizing maps)
- autoencoders

Supervised learning

maps an input to an output based on example **input-output pairs**, e.g.

- regression
- decision trees and random forests
- support vector machines
- artificial neural networks

Reasons to use ML for Science

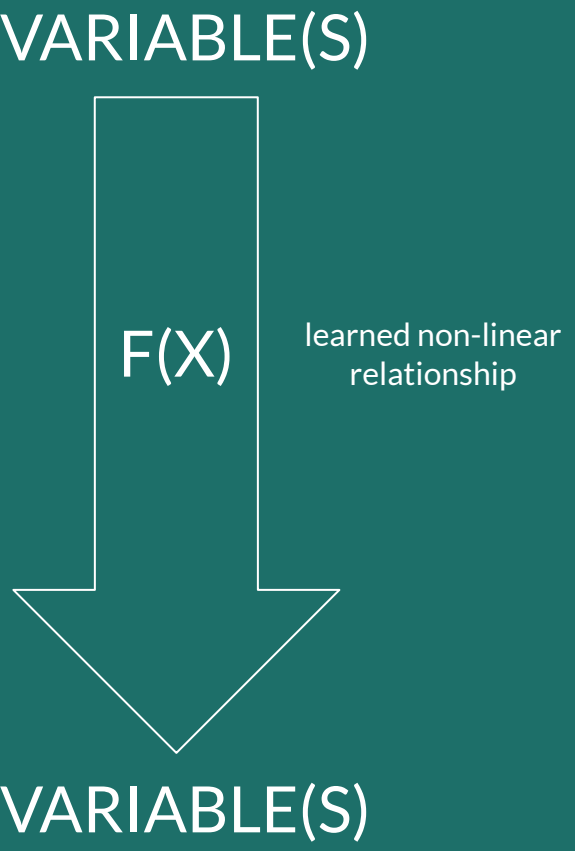
- Do it **better**
  - e.g. Convective parameterizations in models are not perfect, use ML to make them more accurate
- Do it **faster**
  - e.g. Radiation code in models is very slow (but we know the right answer) - use ML methods to speed things up
- Do **something new**
  - e.g. Go looking for non-linear relationships you didn't know were there

Example uses in Climate Science

- Making climate models better
  - Parameterizations
  - Speed-up computations
- Better predictions
  - Post-processing dynamical (physics-based) forecasts
  - Purely data-driven forecasts
- Sources of predictability
  - process-based understanding
  - identifying dynamical model errors/biases
- Feature identification
  - Counting clouds, labeling cloud types in images
  - Counting/identifying extreme events
- Summarizing a lot of data
  - Dimension reduction
  - Cluster/group behaviour

The Basic Idea

Find the **best function** F(X) for fitting the input to the output



Linear Regression

Imagine you have a paired data set **x** [input] and **y** [output].

If **x** and **y** are linearly related, their *modeled* linear relationship would look something like this...

$$\hat{y}_i = a_0 + a_1 x_i$$

prediction

y-intercept

slope

Linear Regression

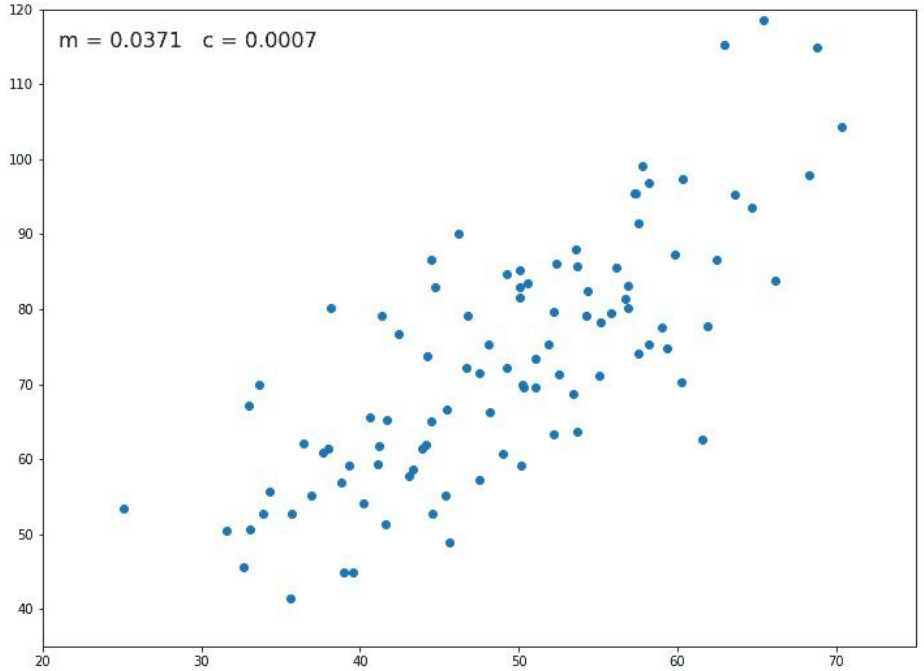
prediction

$$\hat{y}_i = a_0 + a_1 x_i$$

y-intercept slope

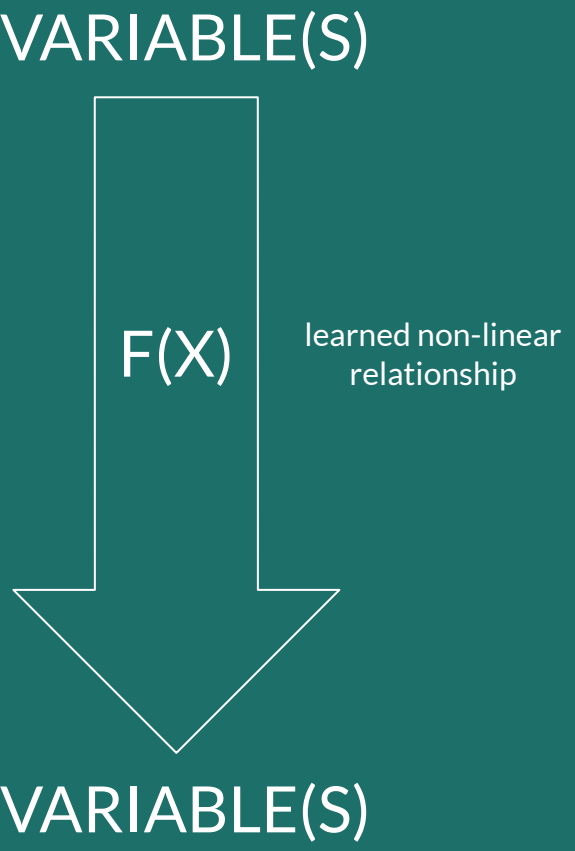
Find a_0 & a_1 to minimize MSE

$$MSE = \frac{1}{n} \sum (\hat{y}_i - y_i)^2$$



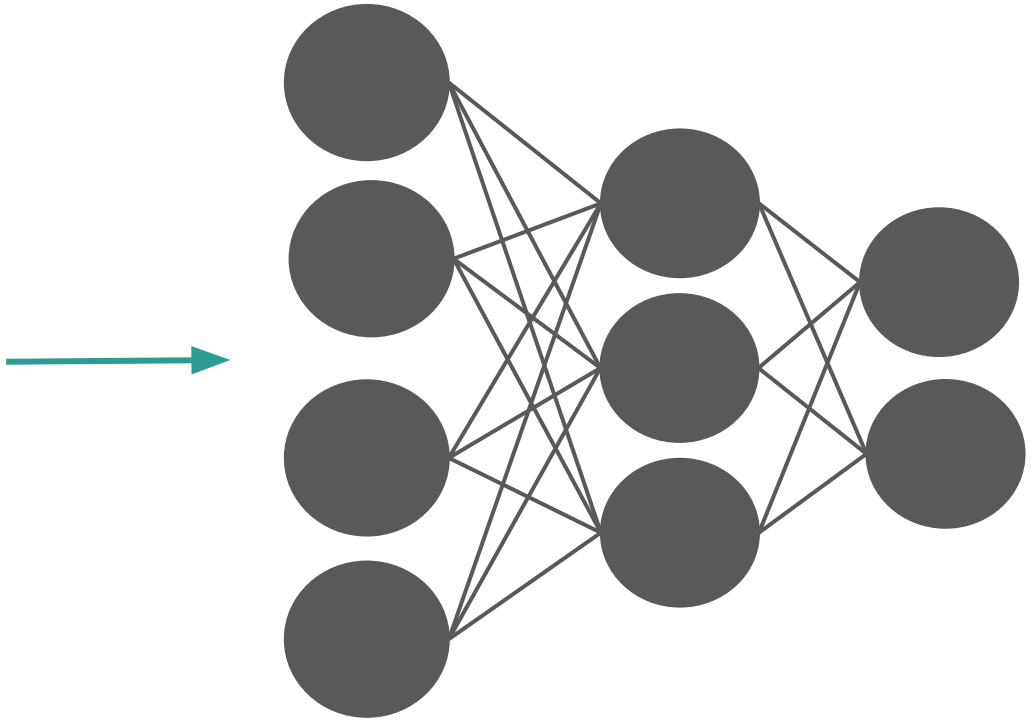
We have an equation for a_0 and a_1 ... but what about when we don't?

Artificial Neural Network (ANN)



Linear regression as an ANN

$$\hat{y}_i = a_0 + a_1 x_i$$



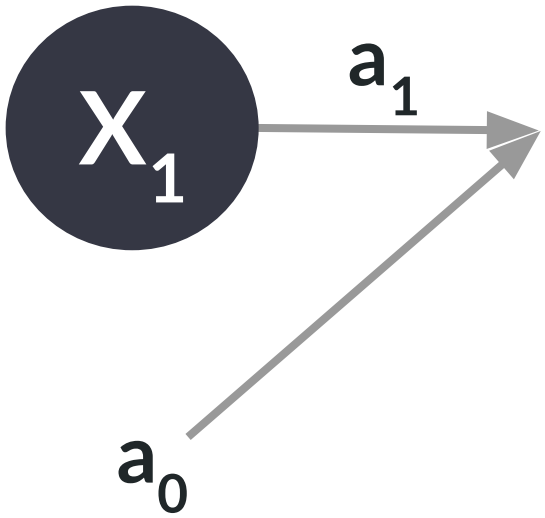
Linear regression as an ANN

$$\hat{y}_i = a_0 + a_1 x_i$$

$= x_1$

Linear regression as an ANN

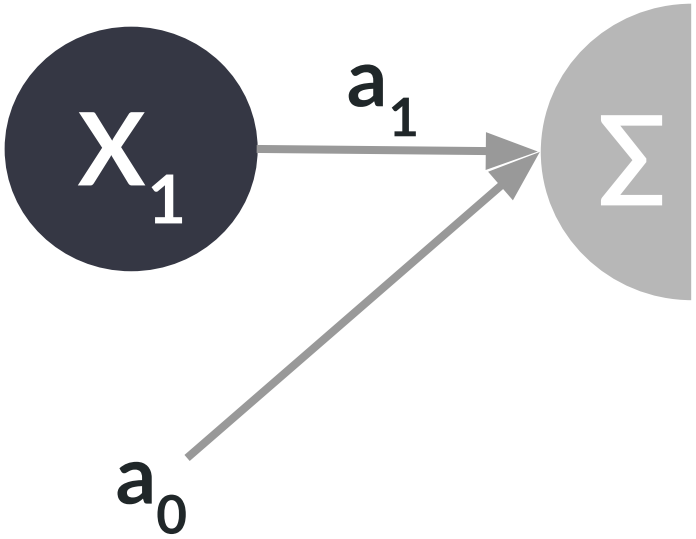
$$\hat{y}_i = a_0 + a_1 x_i$$



$= X_1 a_1 \quad a_0$

Linear regression as an ANN

$$\hat{y}_{x} = a_0 + a_1 x_{x}$$

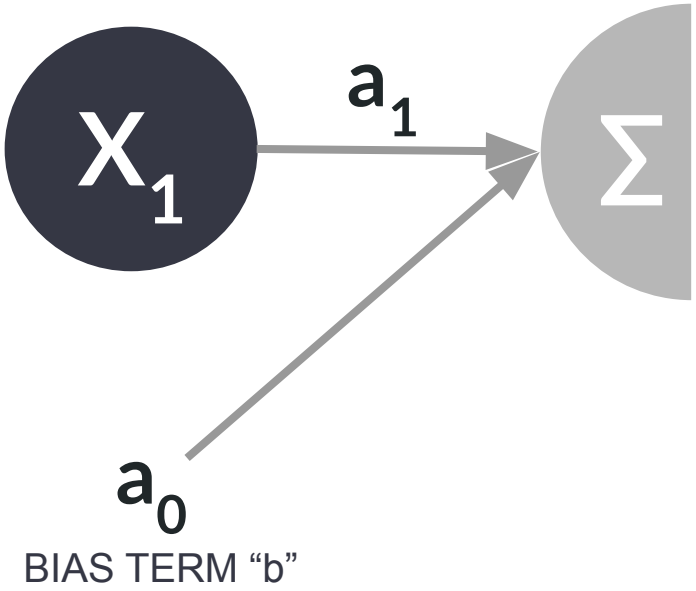


$$= x_1 a_1 + a_0$$

Linear regression as an ANN

$\hat{y}$ = a_0 + a_1 x

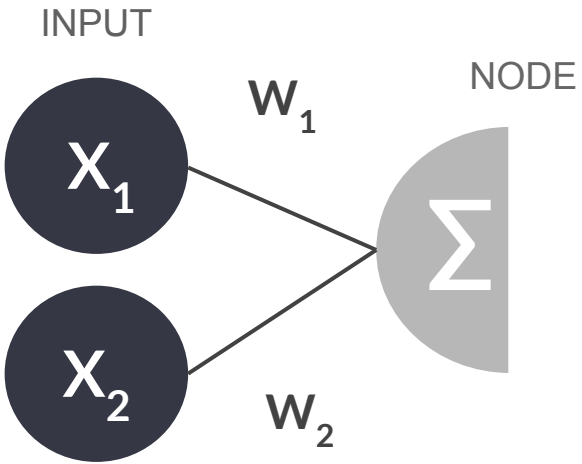
ARCHITECTURE



OUTPUT/PREDICTION

$= x_1 a_1 + a_0$

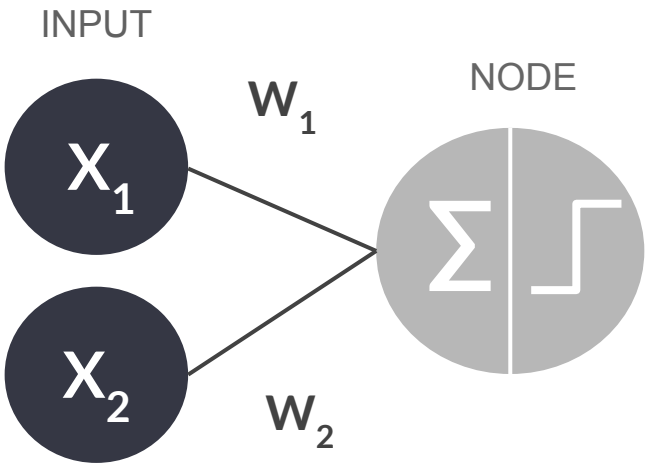
Artificial neural networks



$$= X_1W_1 + X_2W_2 + b$$

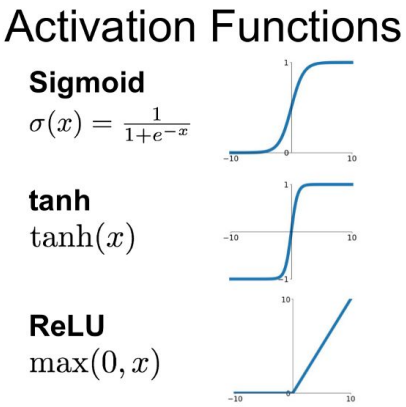
MULTIVARIATE
LINEAR REGRESSION

Artificial neural networks

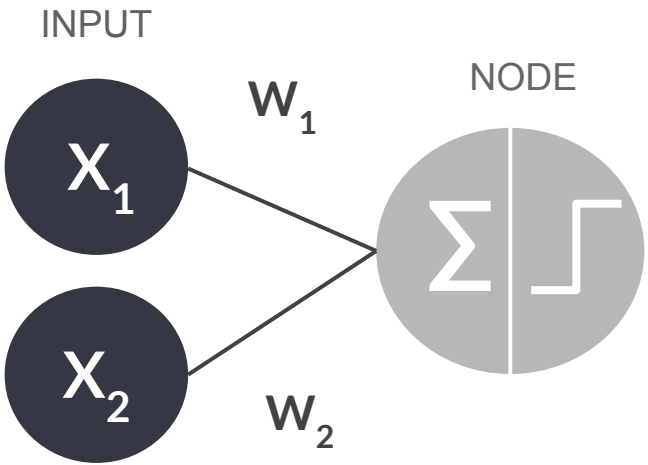


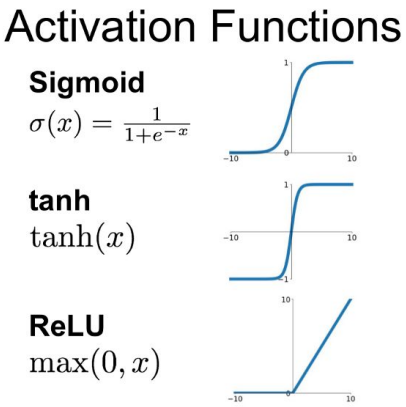
$$= f_{activation}(X_1W_1 + X_2W_2 + b)$$

linear regression with non-linear mapping by an “activation function”



Artificial neural networks

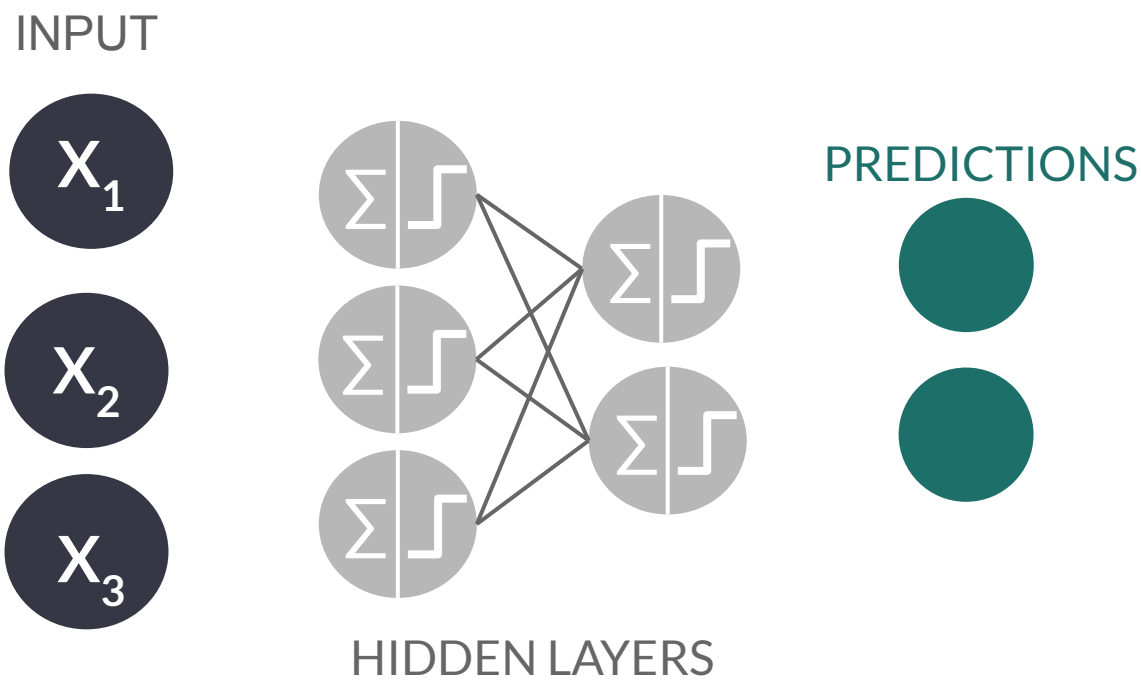




$$= f_{activation}(X_1W_1 + X_2W_2 + b)$$

Activation function determines, if information is moving forward from that specific node.

This is the step that allows for nonlinearity in these algorithms, without activation all we would be doing is linear algebra!



Training an ANN is determining the weights “w” and bias “b"

Once trained, you have an array of weights and biases which can be used for prediction on *new* data

Terminology

units/nodes/neurons: computational unit connected by weights to all other nodes

Weights/Bias: strength of the connection between nodes/shifting term

Activation Function: function that maps the output of a node; determines whether a node is “activated”

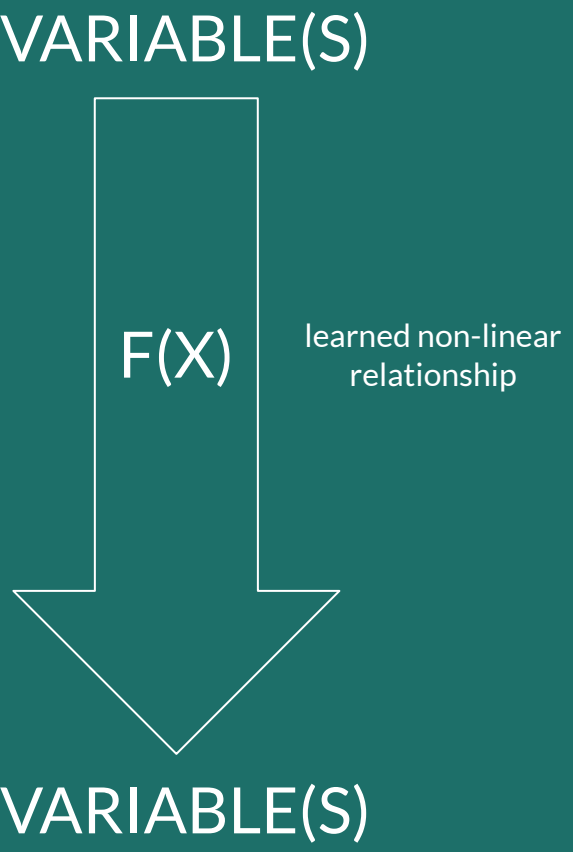
(Hidden) Layer: layer of nodes (not in input or output layer)

Deep: training a NN with **3+** layers

The Basic Idea

Find the **best function** F(X) for fitting the input to the output

... but what do we mean by **best**?



Splitting Your Data

Training Data

- to build the ML model (i.e. determine the parameters)

Validation Data

- to evaluate the ML model on unseen data
- to optimize parameters that are choices by the user

Testing Data

- to evaluate the final ML model on data not used for training or evaluation

How to Split Data

TRAINING data
VALIDATION data
TEST data

Randomly Split

Chronologically Split

How to Split Data

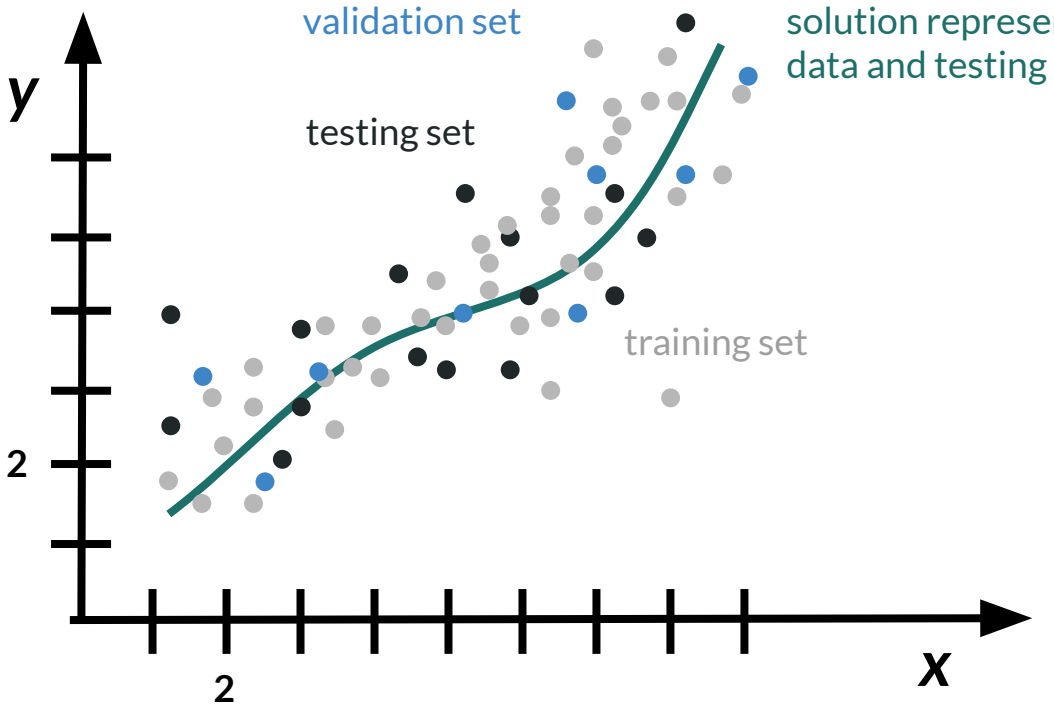
Climate data is often autocorrelated, so we split data **chronologically**

Example: Full dataset from 1900-2000

- Training: 1900-1980
- Validation: 1981-1990
- Testing: 1991- 2000

Chronologically Split

Machine Learning Approach



solution represents training data and testing data well

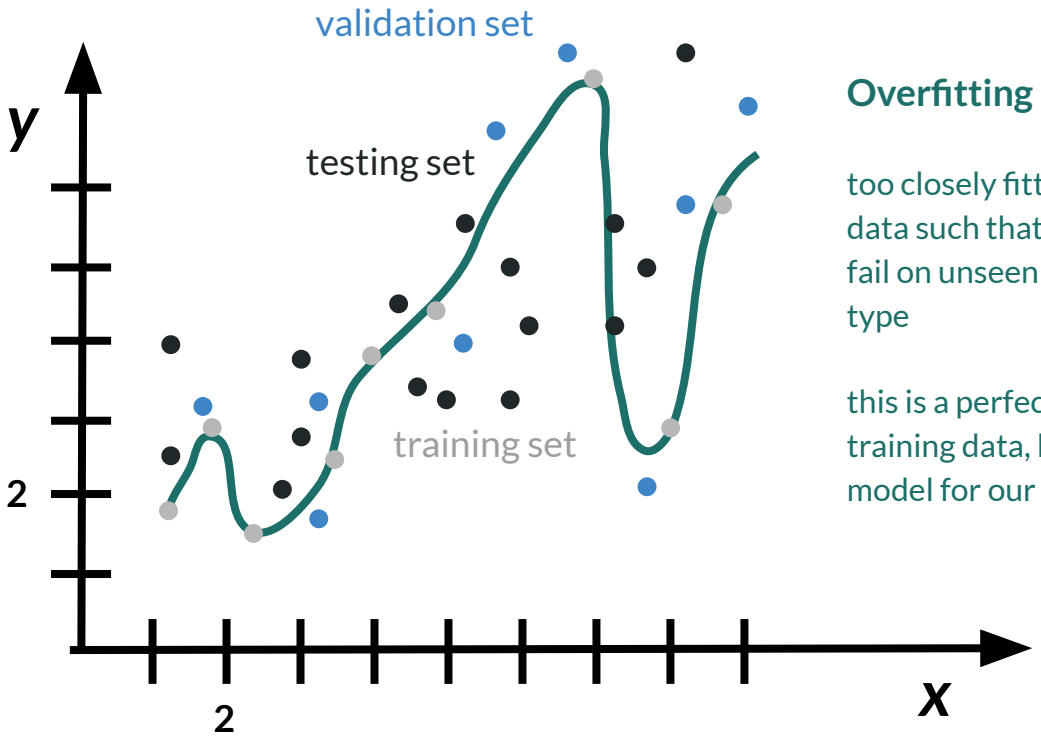
Training Data

Validation Data

Testing Data

ALL THE DATA

Machine Learning Approach



Overfitting

too closely fitting the training data such that the model will fail on unseen data of the same type

this is a perfect model for the training data, but a very poor model for our testing data

Training Data

Validation Data

Testing Data

ALL THE DATA

How do we quantify a good prediction?

cost/loss functions

$$MSE = \frac{1}{n} \sum(\hat{y}_i - y_i)^2$$

$$\text{loss/cost} = \frac{1}{n} \sum(\text{prediction} - \text{actual})^2$$

Find a quantity you want to minimize (the “loss”) to help determine the weights

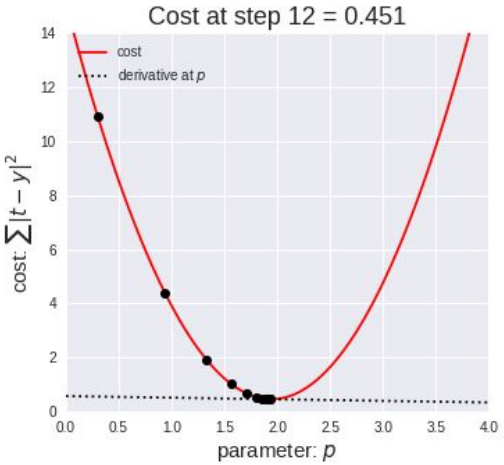
How do we find the minima in real life?

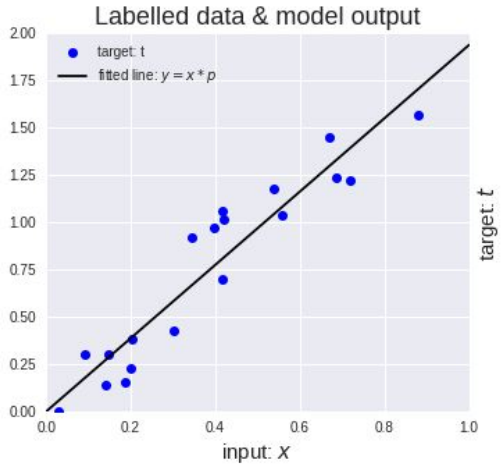
Let's start with an easy linear example

$$\tilde{y}_t = a_0 + a_1 x_t$$

An example loss function (MSE)

$$J = \frac{1}{N}\sum_{i=1}^{N}(\tilde{y}_i - y_i)^2$$





Gradient Descent

Gradient: gives you the direction of greatest/fastest increase

Descent: let's follow the greatest *decrease*…

Gradient Descent: a method used to minimize the loss/cost function; algorithm to update the weights to follow greatest loss decrease

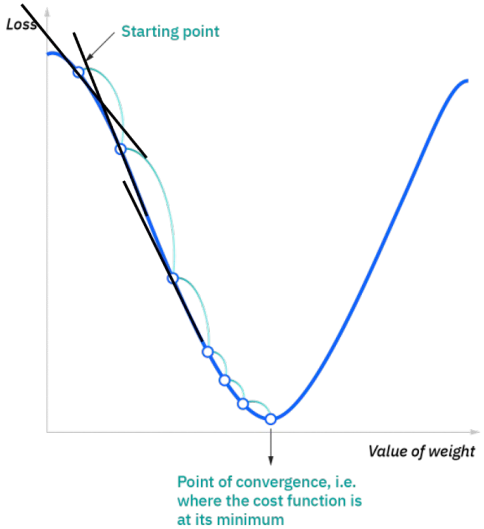
Learning Rate: how "quickly" the neural network tries to minimize the loss function

https://towardsdatascience.com/gradient-descent-3a7db7520711#:~:text=tl%3Bdr%20Gradient%20Descent%20is,used%20in%20machine%20learning%20models.

Gradient descent mathematically

$$J = \frac{1}{N}\sum_{i=1}^{N}(\tilde{y}_i - y_i)^2$$

$$\tilde{y}_t = a_0 + a_1 x_t$$

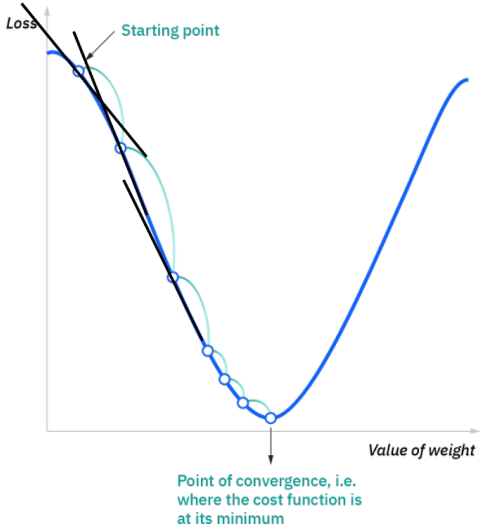


- MSE (our loss function) depends on the linear function and its weights: a0 and a1
- The **gradient** is calculated as partial derivatives of the loss function with relation to the weights

$$J = J(a_0, a_1) \qquad \nabla J_t = \begin{bmatrix} \frac{\partial J}{\partial a_0} \\ \frac{\partial J}{\partial a_1} \end{bmatrix}$$

https://www.ibm.com/cloud/learn/gradient-descent

Partial derivatives for MSE (linear regression)



$$J = \frac{1}{N}\sum_{i=1}^{N}(\tilde{y}_i - y_i)^2 = \frac{1}{N}\sum_{i=1}^{N}((a_0 + a_1 x_i) - y_i)^2$$

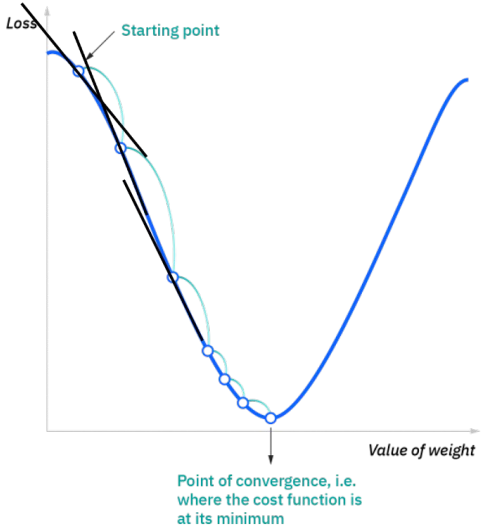
From this, we can calculate

$$\frac{\partial J}{\partial a_0} = \frac{\partial}{\partial a_0}\left(\frac{1}{N}\sum_{i=1}^{N}((a_0 + a_1 x_i) - y_i)^2\right) = \frac{2}{N}\sum_{i=1}^{N}((a_0 + a_1 x_i) - y_i)$$
$$= \frac{2}{N}\sum_{i=1}^{N}(\tilde{y}_i - y_i)$$

$$\frac{\partial J}{\partial a_1} = \frac{\partial}{\partial a_1}\left(\frac{1}{N}\sum_{i=1}^{N}((a_0 + a_1 x_i) - y_i)^2\right) = \frac{2}{N}\sum_{i=1}^{N}((a_0 + a_1 x_i) - y_i)x_i$$
$$= \frac{2}{N}\sum_{i=1}^{N}(\tilde{y}_i - y_i)x_i$$

https://www.ibm.com/cloud/learn/gradient-descent

Gradient descent mathematically



After taking the derivatives, the rest is easy!

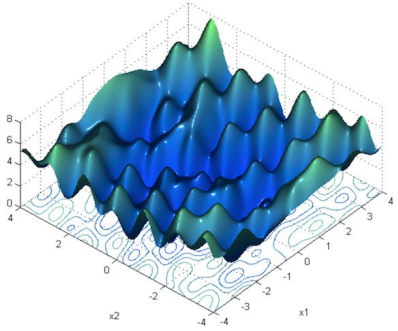
$$
\begin{aligned}
a_{0,new} &= a_0 - \lambda \frac{\partial \text{Loss}}{\partial a_0} \\
a_{1,new} &= a_1 - \lambda \frac{\partial \text{Loss}}{\partial a_1}
\end{aligned}
$$

The only other thing one must pay attention to is the learning rate λ (how big of a step to take). Too small and finding the right weights takes forever, too big and you might miss the minimum.

https://www.ibm.com/cloud/learn/gradient-descent

So far so good - this all looks super easy! What's the catch?

We calculated the gradients by hand because we knew the functional form we wanted to fit...



But the whole point of an ANN is that you do not need to assume the optimal functional form of the equation ahead of time - it's probably really complex!

So what do we do now...?

Backpropagation:
how gradients are calculated for gradient descent

- Short for “back propagation of errors”
- A method for determining the gradient of the loss function when you don’t know the functional form
- **The method calculates the gradient of the loss function with respect to the neural network weights.**
- Why only certain activation functions are allowed - must be differentiable!

$$J = \frac{1}{N}\sum_{i=1}^{N}(\tilde{y}_i - y_i)^2$$

$$w_{k,new} = w_k - \lambda \frac{\partial \text{Loss}}{\partial w_k}$$

The partial derivative of the PREDICTED y with respect to the weights w.

$$\frac{\partial \text{Loss}}{\partial w_k} = \frac{\partial}{\partial w_k}\left(\frac{1}{n}\sum_{i=1}^{n}(\tilde{y}_i - y_i)^2\right) = \frac{2}{n}\sum_{i=1}^{n}(\tilde{y}_i - y_i)\frac{\partial \tilde{y}_i}{\partial w_k}$$

What we need to know!

Backpropagation:

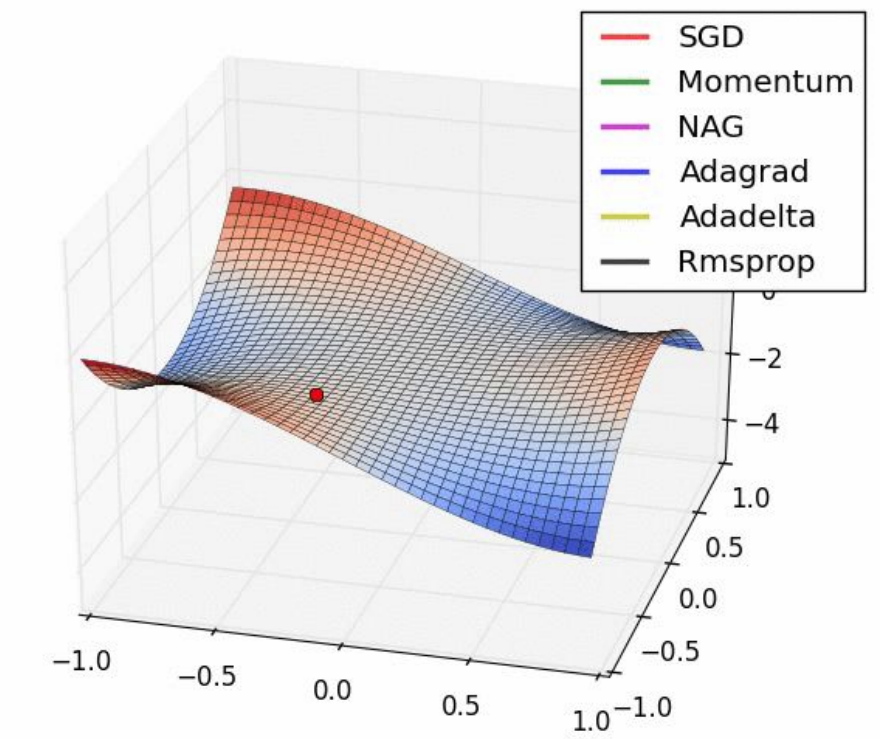
how gradients are calculated for gradient descent

3Blue1Brown Neural Network Series (Chapter 4):
https://www.youtube.com/watch?v=tleHLnjs5U8&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&index=4

Neural Networks & Deep Learning *Free* Book (Chapter 2):
http://neuralnetworksanddeeplearning.com/chap2.html

The Chain Rule!!

$$\frac{\partial \text{Loss}}{\partial w_k} = \frac{\partial}{\partial w_k}\left(\frac{1}{n}\sum_{i=1}^{n}(\tilde{y}_i - y_i)^2\right) = \frac{2}{n}\sum_{i=1}^{n}(\tilde{y}_i - y_i)\frac{\partial \tilde{y}_i}{\partial w_k}$$



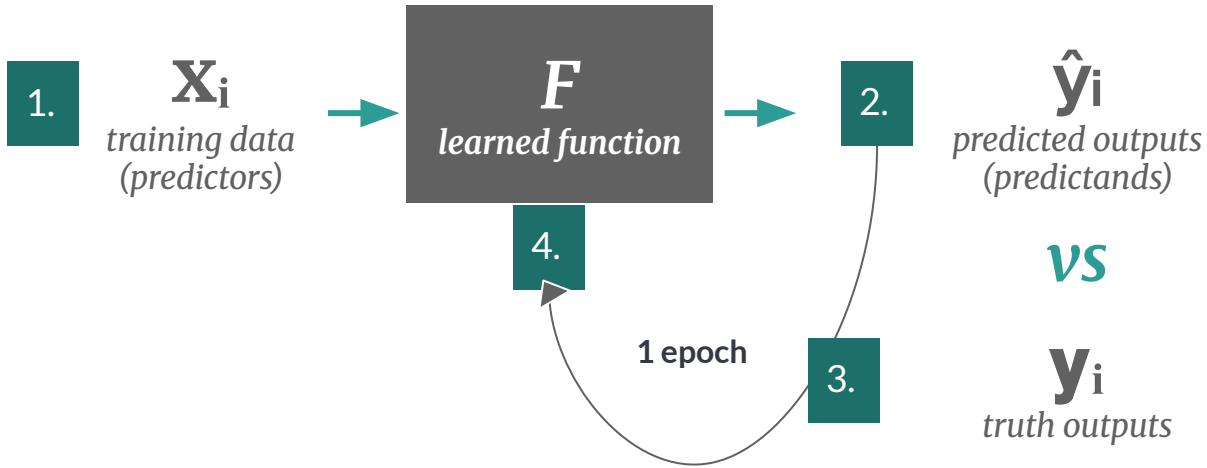
Gradient Descent Algorithms

Figure by Alec Radford

Machine Learning Approach

1) Training data is put into the machine learning model's learned function.
2) The model outputs its predictions.
3) These predictions are compared with their truth values.
4) The weights are updated to decrease the error between predicted and truth outputs.

This cycle continues.



Terminology

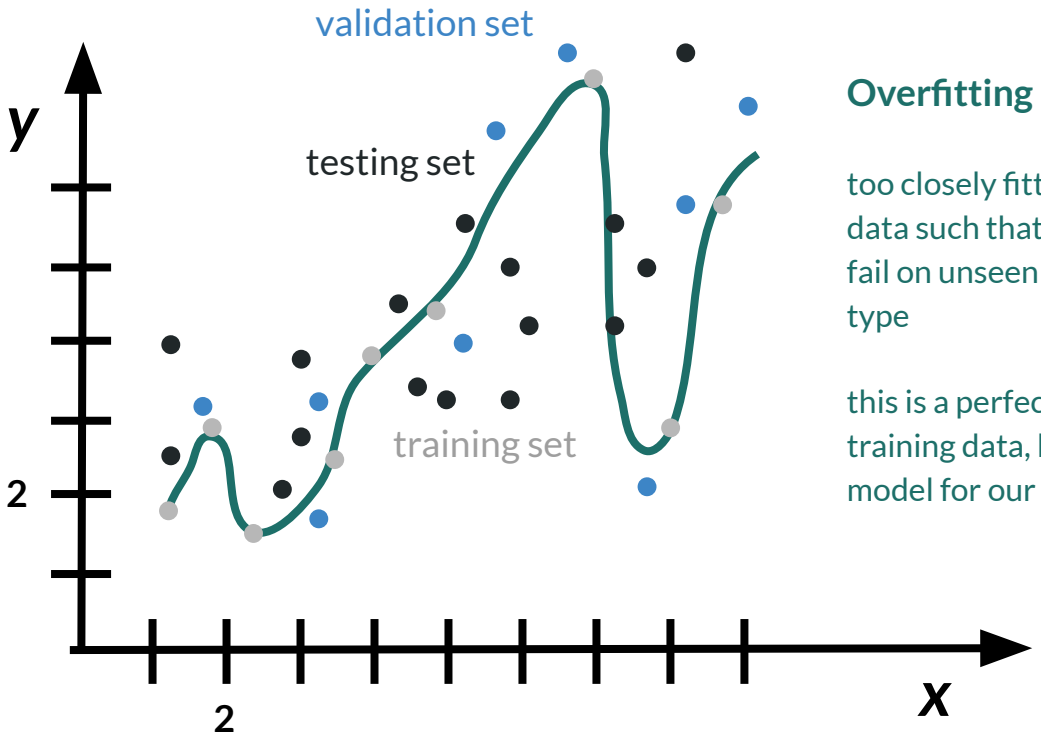
Batch size: number of samples in a “chunk” of training data that is iterated before updating the weights and biases

Epoch: One complete pass through the entire training dataset.

Example:

Samples: N = 100
Batch Size: N = 20
1 Epoch = 5 batches of 20 samples

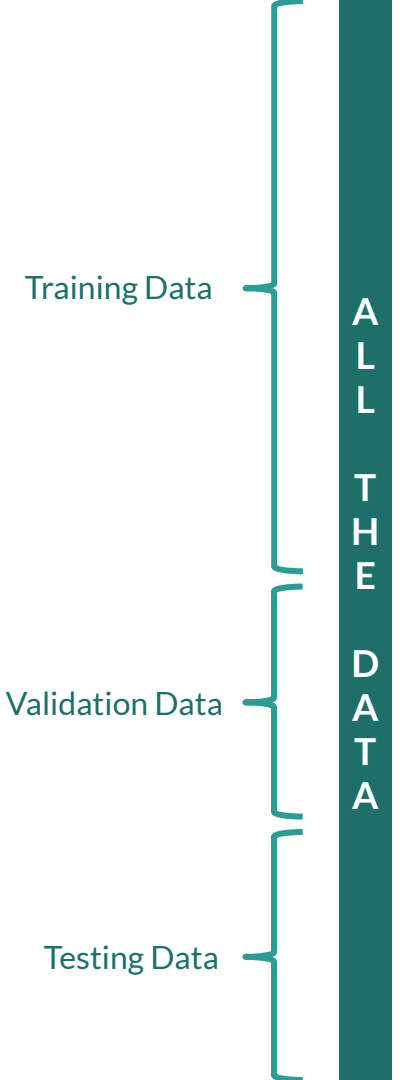
How do we know if we're overfitting?



Overfitting

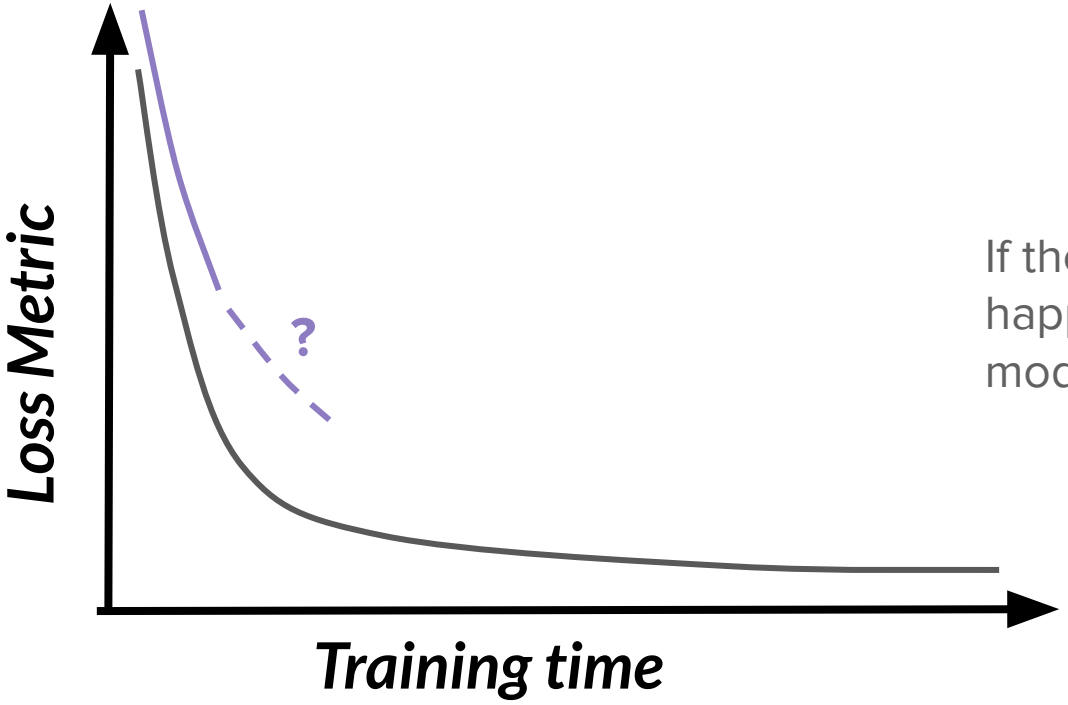
too closely fitting the training data such that the model will fail on unseen data of the same type

this is a perfect model for the training data, but a very poor model for our testing data



What does **overfitting** look like?

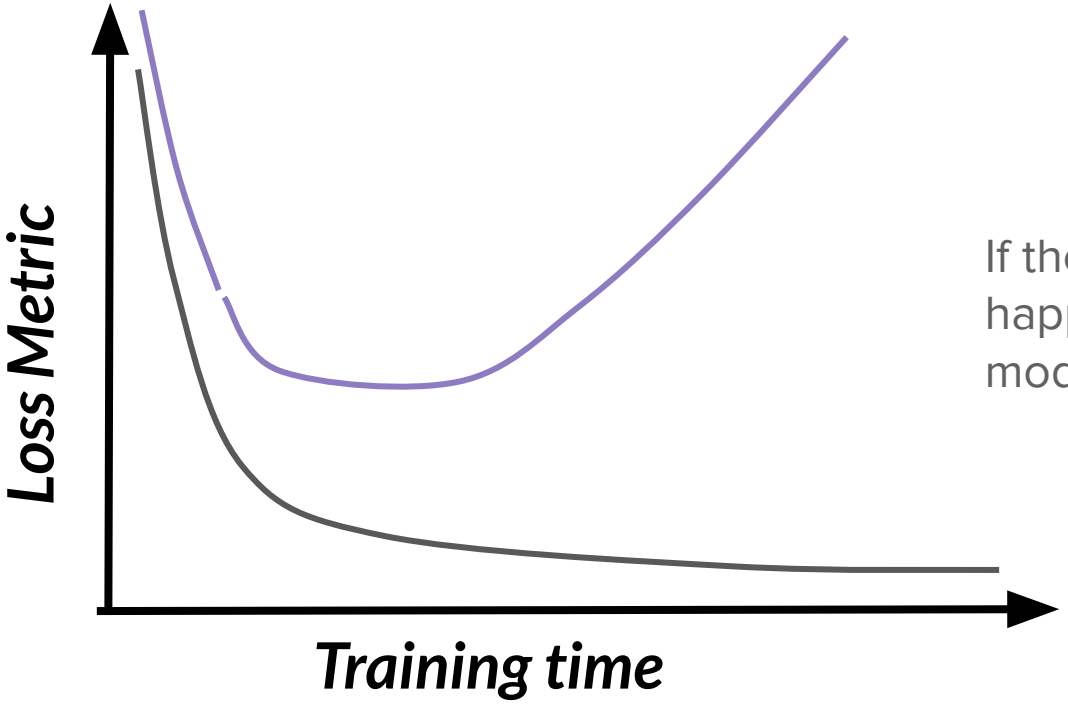
TRAINING data
VALIDATION data



If the model is overfitting, what will happen to the performance of the model on **validation**?

What does **overfitting** look like?

TRAINING data
VALIDATION data



If the model is overfitting, what will happen to the performance of the model on **validation**?

Common ways to address overfitting:

Ridge Regression/L2

- Adds a term to the loss/error function that forces coefficients to share the weight

$$\text{Loss} = \text{RMSE} + \lambda \sum_{i=1}^{p} w_i^2$$

Ridge Parameter
How much to penalize large weights

Shrinkage Term
Forces the individual coefficients to be small

>> sharing of weight across coefficients

Common ways to address overfitting:

LASSO Regression/L1

- Adds a term to the loss/error function that forces some weights to zero

$$\text{Loss} = \text{RMSE} + \lambda \sum_{i=1}^{p} |w_i|$$

LASSO Parameter
How much to penalize large weights

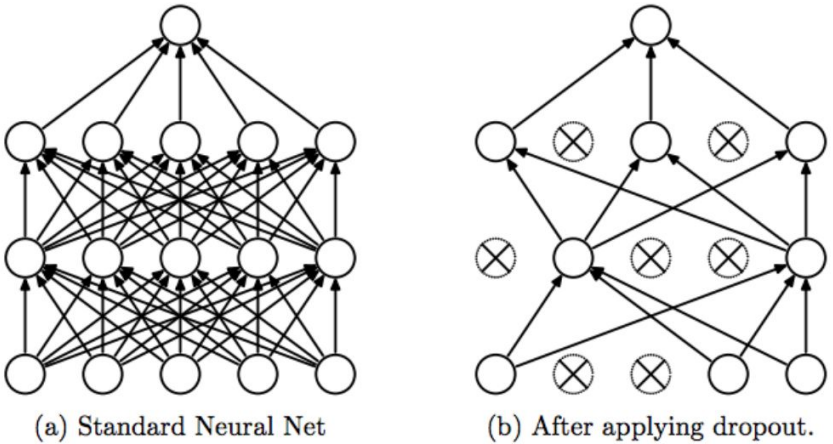
Shrinkage Term
Punishes high values but actually sets them to exactly zero if not important

>> helpful for determining which features are **most** important

Common ways to address overfitting:

Dropout *(only during training)*

- Randomly remove or “drop out” nodes, requiring that network to still make accurate predictions in spite of losing these nodes.
- Forces nodes within a layer to take on more or less responsibility for the inputs



(a) Standard Neural Net

(b) After applying dropout.

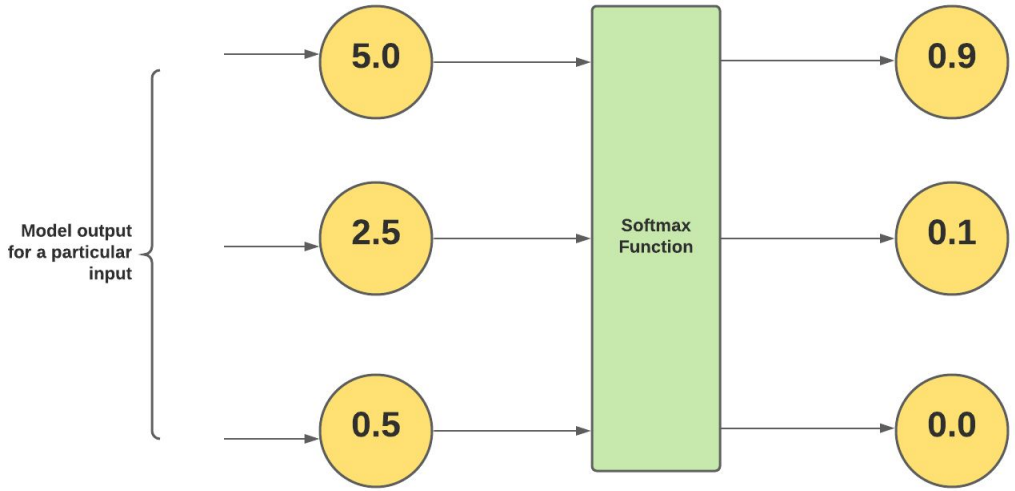
requires the user chose the percent to drop-out

- *(often ~50-80%)*

requires more epochs during training

Softmax: Let’s make the output a probability!

- Converts “raw” output to probabilities for classification tasks
- After the softmax, output values are between 0 and 1
- The class with the highest probability gives your answer



https://vitalflux.com/what-softmax-function-why-needed-machine-learning/

ANN considerations

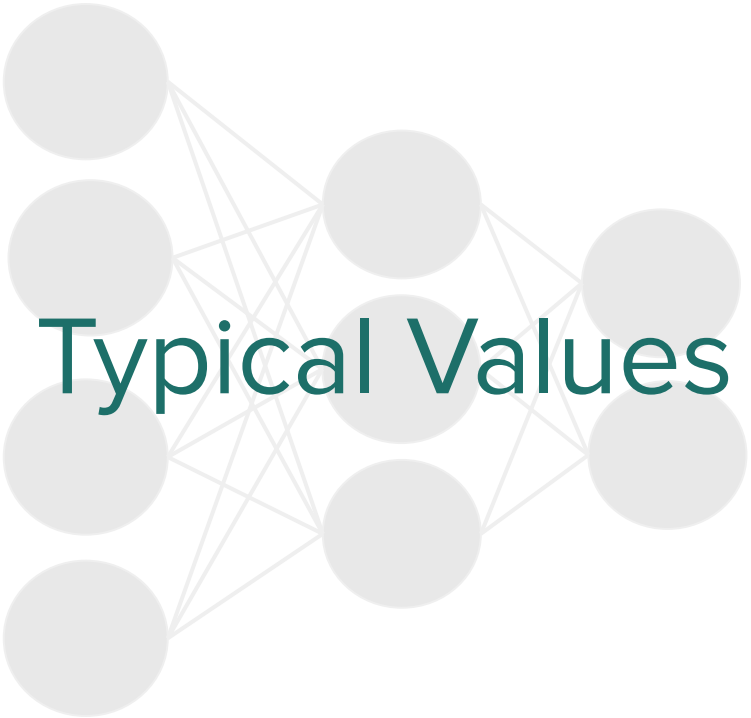
Train the data the right number of times (epochs)....
If we iterate too few times, our model will not have time to find the optimal fit. If we iterate too many times, we will overfit the training data.

Choosing the right number of layers and nodes (architecture)....
Fewer layers/nodes allow for easier interpretability. More layers/nodes allow for more nonlinearity.

Choose an activation function most appropriate for your solution....

Identify the right learning rate....
A learning rate too large may cause the solution to skip between local minima. A learning rate too small may get stuck in one local minimum, and also may take a longer to learn.

Others include… batch size, gradient descent algorithm, regularization, data preparation



Typical Values

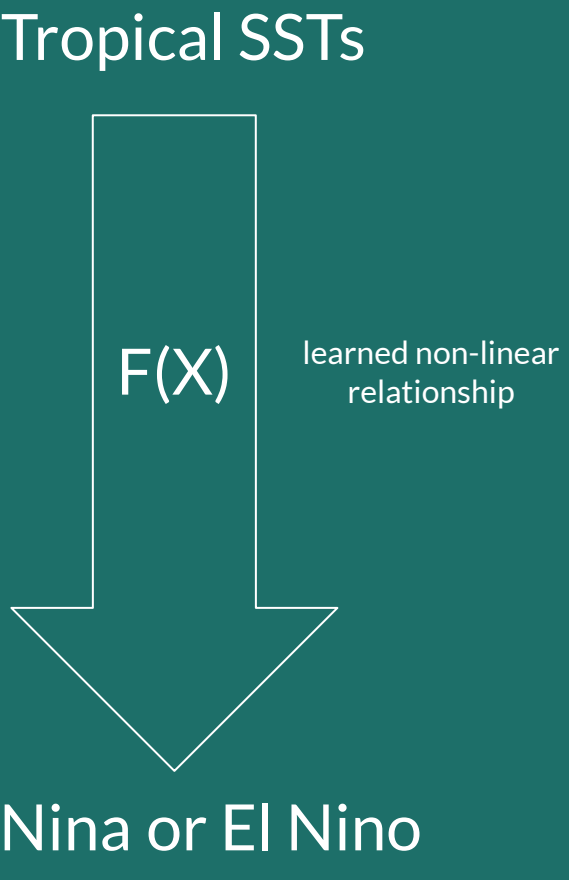
Learning Rate: 0.01, 0.001., 0.0001

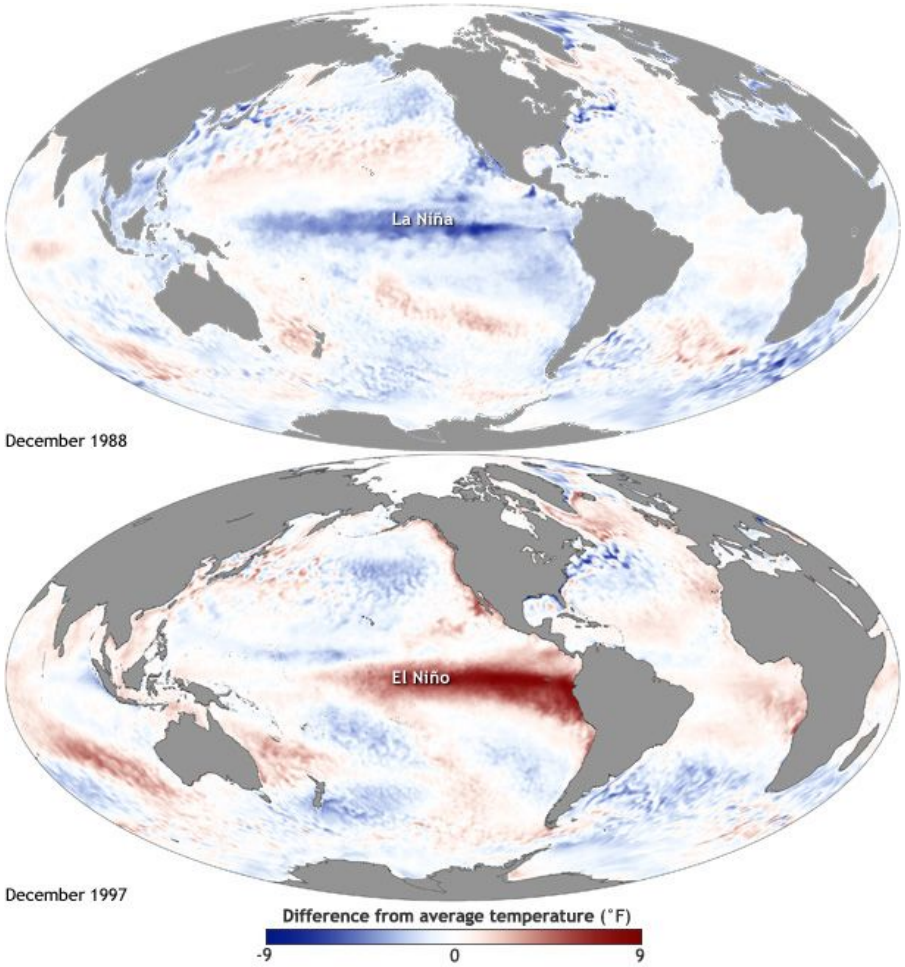
Batch Size: 32, 64, 128

Nodes: 4, 8, ..., 256

ENSO PREDICTION

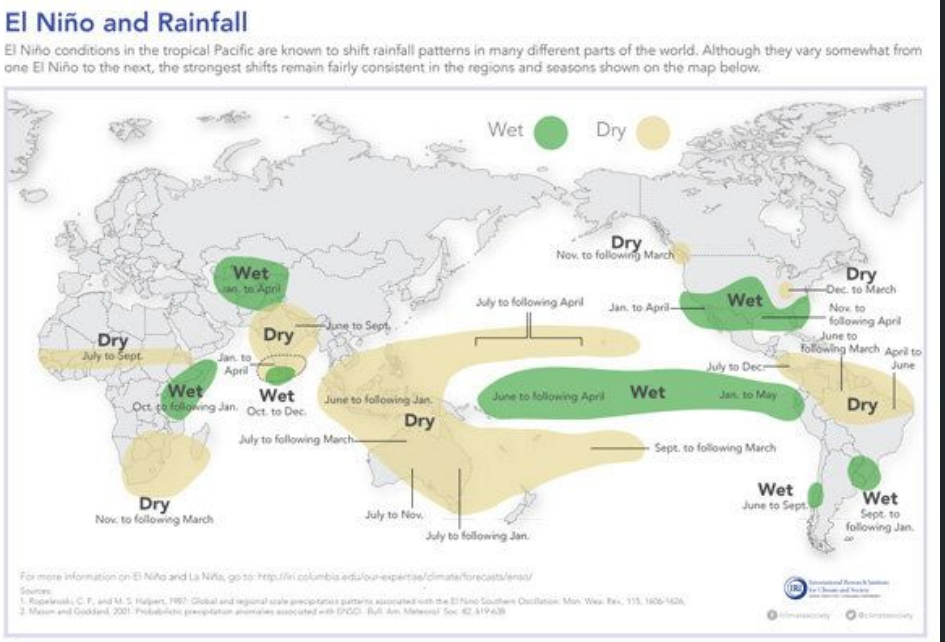
Find the **best function** that maps SSTs to the Nino3.4 index





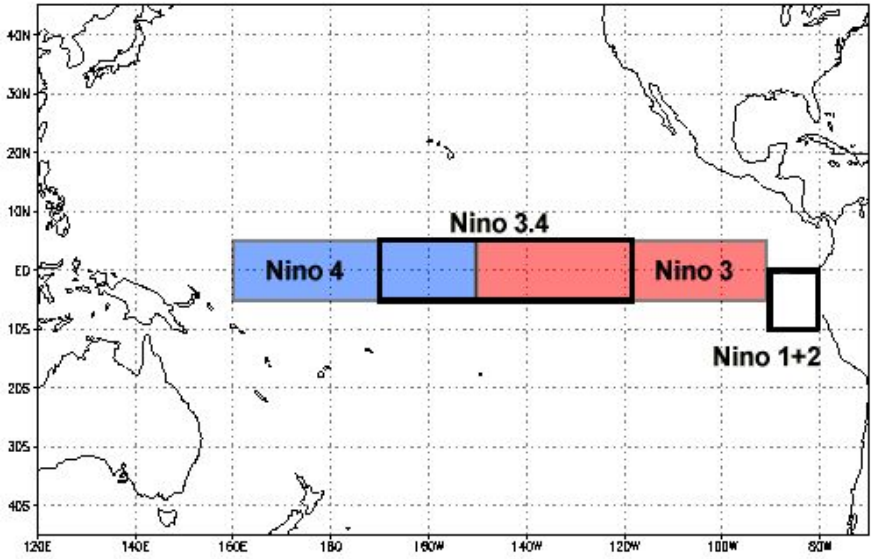
SST anomalies during a strong La Niña and El Niño. Maps by NOAA Climate.gov, based on data provided by NOAA View.

What is the **El Nino Southern Oscillation** (ENSO)?



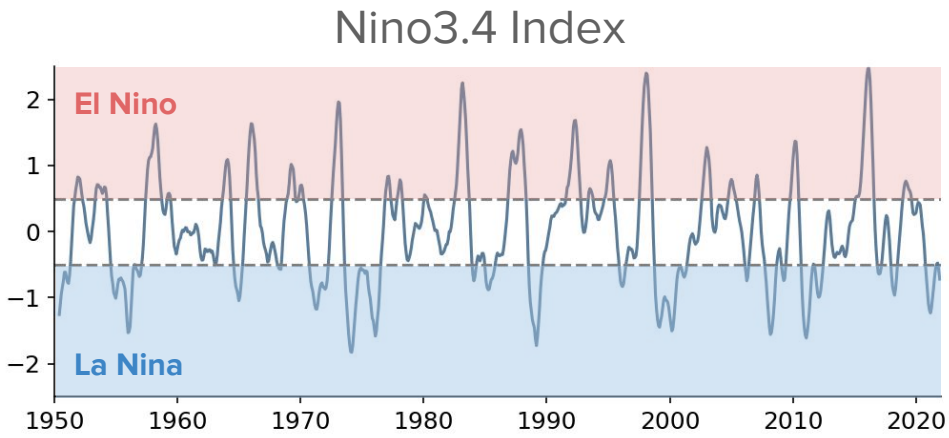
Typical rainfall patterns during El Niño events. Such teleconnections are *likely* during El Niño events, but not *certain*. Map by IRI.

Why do we care?

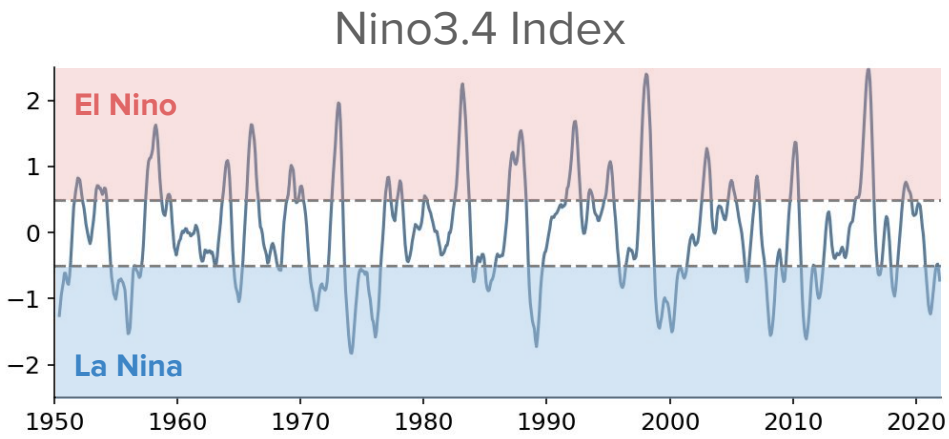


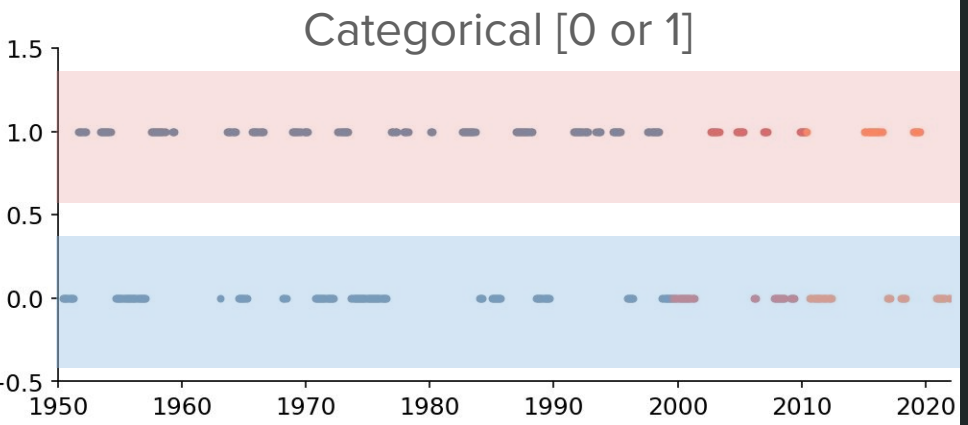
How do we define ENSO?

Source: https://www.ncei.noaa.gov/access/monitoring/enso/sst

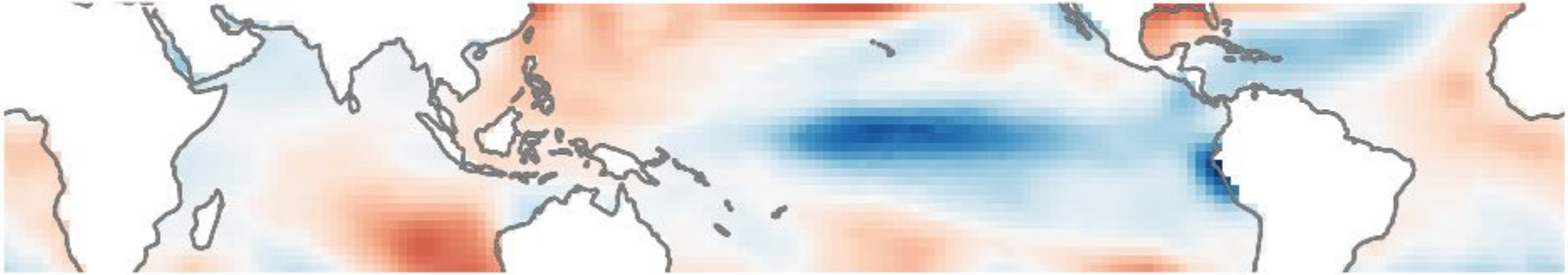


How do we define ENSO?





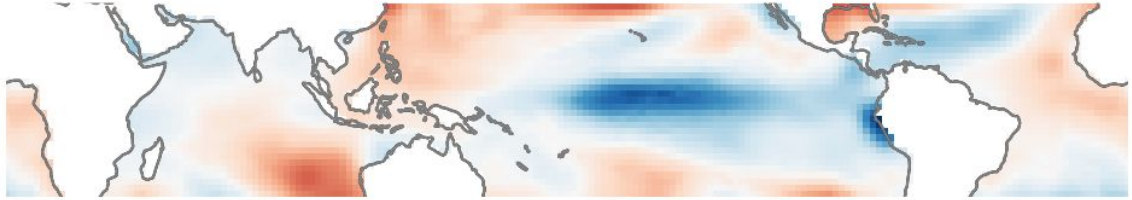
What will we predict?

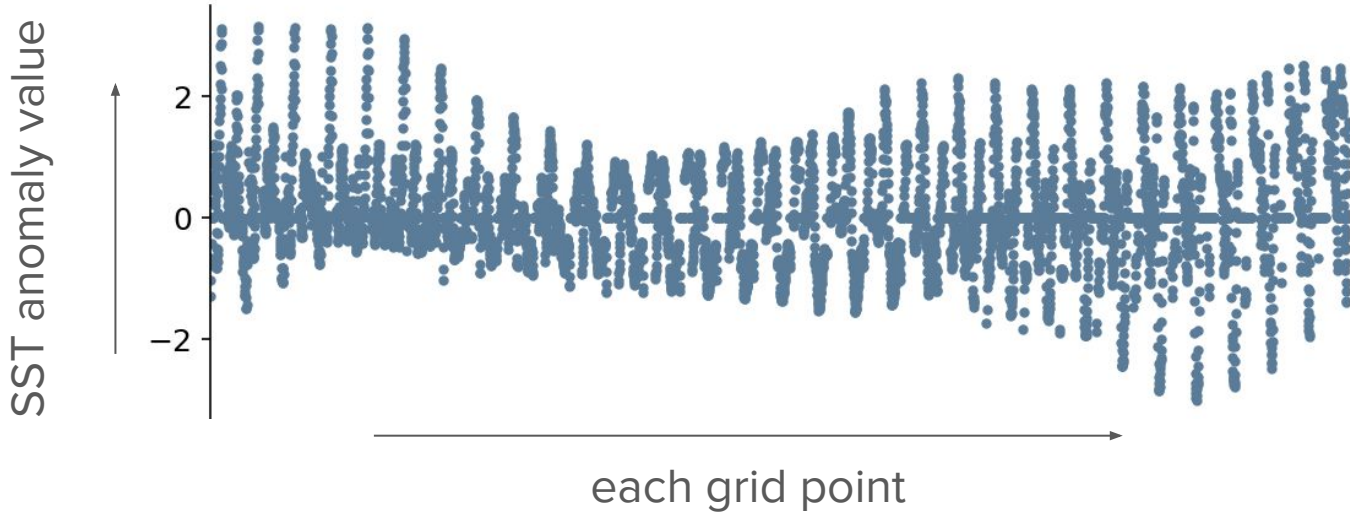


?

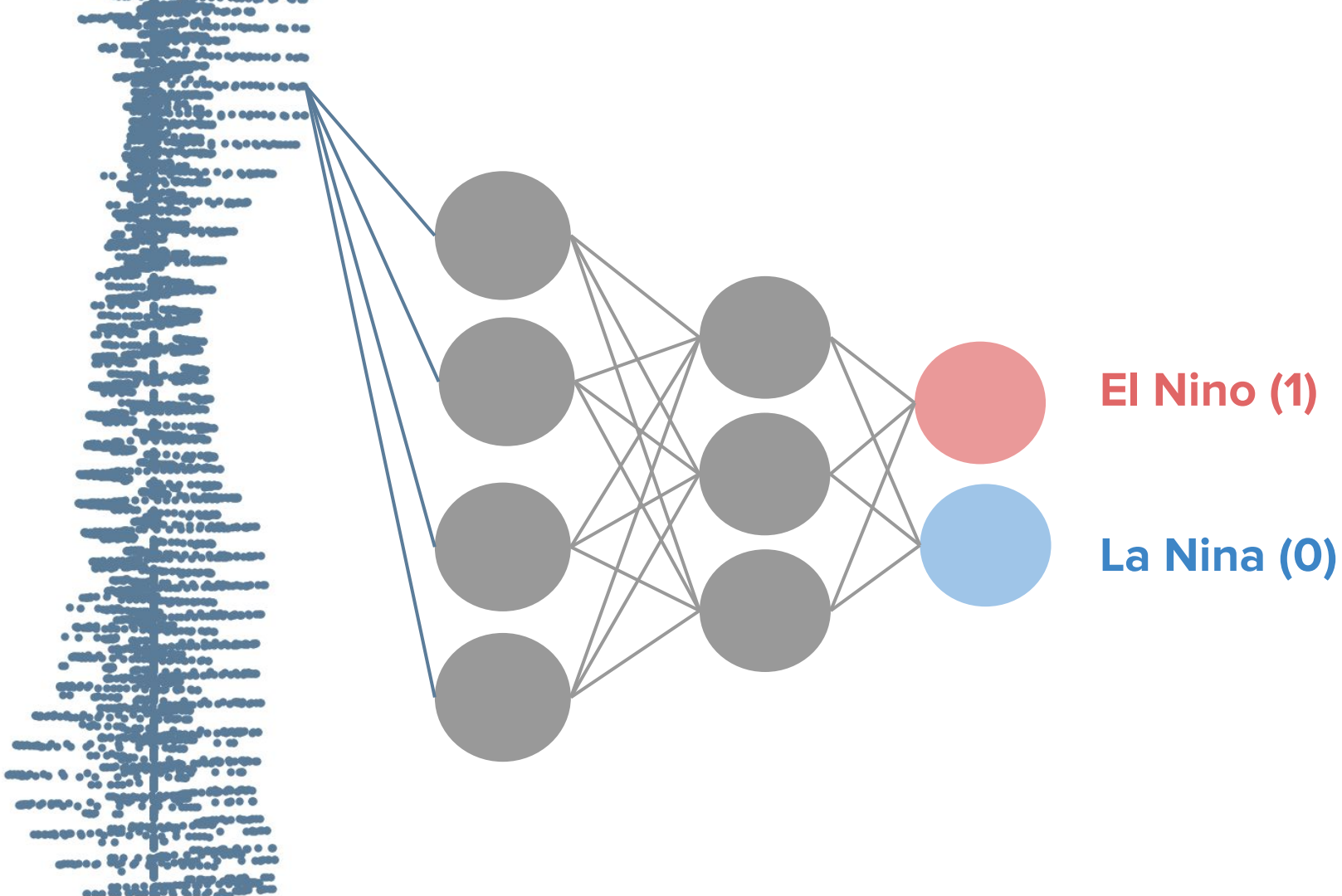
La Nina (0)

El Nino (1)





N = 5580 = 31 [lat] x 180 [lon]


El Nino (1)
La Nina (0)

model.summary( )

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 5580)]            0
_________________________________________________________________
dense_3 (Dense)              (None, 12)                66972
_________________________________________________________________
dense_4 (Dense)              (None, 12)                156
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 26
=================================================================
Total params: 67,154
Trainable params: 67,154
Non-trainable params: 0
_________________________________________________________________
```

How do we get the parameter #?

5580 x 12 = 66960

BUT we also need to add the bias terms for each node

66960 + 12 = **66972**

model.summary( )

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 5580)]            0
dense_3 (Dense)              (None, 12)                66972
dense_4 (Dense)              (None, 12)                156
dense_5 (Dense)              (None, 2)                 26
=================================================================
Total params: 67,154
Trainable params: 67,154
Non-trainable params: 0
_________________________________________________________________
```

How do we get the parameter #?

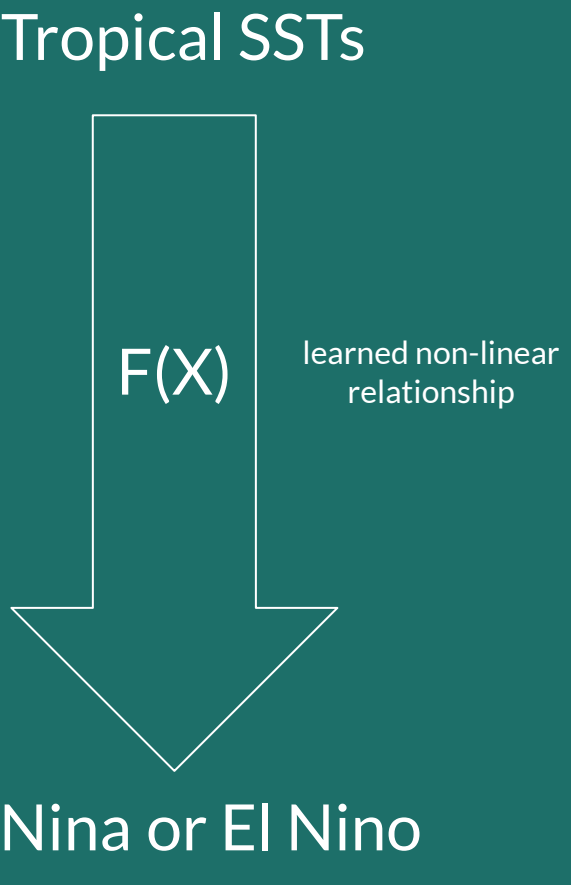
5580 x 12 = 66960

BUT we also need to add the bias terms for each node

66960 + 12 = **66972**

12*12 = 144 + 12 = **156**
12*2 = 24 + 2 = **26**

ENSO EXAMPLE



During this exercise, try to:

- Train a “good” model
  - I’ll leave the definition up to you
- Train a “bad” model
  - E.g. an overfit model = bad model
- Try some non-standard values for various hyperparameters
  - Does the ANN respond as you expected?

Some resources I recommend to get started!

3Blue1Brown Videos

Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow (book)

Blogs! (IBM, AWS, Towards Data Science)

Coursera ML Course by Andrew Ng

Tensorflow/Keras and Pytorch are the go-to python machine learning libraries

- their websites have example code
- Tensorflow has a smaller learning curve