

***“Quantum Computers
Could Solve Countless
Problems—And Create a
Lot of New Ones”***

Feb 2023

Source:

<https://time.com/6249784/quantum-computing-revolution/>



Model Comparisons

Classification of mathematical models

	Empirical Data-driven	Mechanistic
Deterministic	machine learning (ML)	equation-based models (EBM): differential equations & difference equations
Stochastic	statistical models	agent-based models (ABM)

Navier-Stokes (meteorology)

The diagram illustrates the Navier-Stokes equation with annotations for each term. Red curly braces group the terms under their respective category labels.

DENSITY
of fluid

ACCELERATION
of fluid

FORCES
acting on fluid

$$\rho \left(\underbrace{\frac{\partial V}{\partial t}}_{\text{Change in fluid velocity over time}} + \underbrace{V \cdot \nabla V}_{\text{The speed and direction in which fluid is moving}} \right) = \underbrace{\nabla P}_{\text{Internal pressure gradient of fluid}} + \underbrace{\rho g}_{\text{External forces on fluid (gravity)}} + \underbrace{\mu \nabla^2 V}_{\text{Internal stress forces acting on fluid (e.g. viscosity)}}$$

Change in **fluid velocity** over time

The speed and direction in which fluid is moving


Internal pressure gradient of fluid

External forces on fluid (gravity)

Internal stress forces acting on fluid (e.g. viscosity)

Einstein Field Equations (physics)

CURVING
of spacetime
by matter & energy



MOVEMENT
through spacetime
by matter & energy



$$R_{uv} - \frac{1}{2}R g_{uv} + \Lambda g_{uv} = \frac{8\pi G}{c^4} T_{uv}$$

Black-Scholes (finance)

price of **call** or
put option

stock volatility

$$rV = \frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S}$$

change in
call/option
price over time

price of **stock**

Black-Scholes Solution (call option)

$$C = N(d_1)S_t - N(d_2)Ke^{-rt}$$

$$\text{where } d_1 = \frac{\ln \frac{S_t}{K} + (r + \frac{\sigma^2}{2})t}{\sigma\sqrt{t}}$$

$$\text{and } d_2 = d_1 - \sigma\sqrt{t}$$

C = call option price

N = CDF of the normal distribution

S_t = spot price of an asset

K = strike price

r = risk-free interest rate

t = time to maturity

σ = volatility of the asset



Any time

Since 2023

Since 2022

Since 2019

Custom range...

Sort by relevance

Sort by date

Any type

Review articles

☐ include patents☒ include citations☒ Create alert**Predicting the stock price of frontier markets using machine learning and modified Black–Scholes Option pricing model**[R Chowdhury](#), [MRC Mahdy](#), [TN Alam...](#) - *Physica A: Statistical ...*, 2020 - Elsevier

... obtained using **Black–Scholes** equations, we have used data mining and **machine learning** ... , neural net, linear regression and ensemble **learning** method to predict the future price of ...

☆ Save Cite Cited by 39 Related articles All 10 versions

[PDF] Machine learning in finance: the case of deep learning for option pricing[R Culkin](#), [SR Das](#) - *Journal of Investment Management*, 2017 - [srdas.github.io](#)

... We begin with the specific exercise of **learning** the **Black-Scholes** option pricing model from simulated data. If this is possible to achieve with high accuracy, then it suggests that market ...

☆ Save Cite Cited by 115 Related articles

Application of Neural Network Machine Learning to Solution of Black-Scholes Equation[MV Klibanov](#), [KV Golubnichiy](#), [AV Nikitin](#) - *arXiv preprint arXiv:2111.06642*, 2021 - [arxiv.org](#)

... **Machine Learning** to the solution of the **Black-Scholes** ... -Reversibility and **Machine Learning** method are compared in ... We apply **Machine Learning** to reduce the probability of non...

☆ Save Cite Cited by 2 Related articles All 2 versions

Computing Black Scholes with Uncertain Volatility—A Machine Learning Approach[K Hellmuth](#), [C Klingenberg](#) - *Mathematics*, 2022 - [mdpi.com](#)

... to the **Black Scholes** equation with uncertain volatility and interest rates, when there are doubts concerning its true value, or to familiar equations such as the **Black Scholes** equation for ...

Feynman's Approach to Theoretical Physics



Source: https://video.ucdavis.edu/media/Richard+Feynman+on+New+Laws+of+Physics/1_1kljdbq2

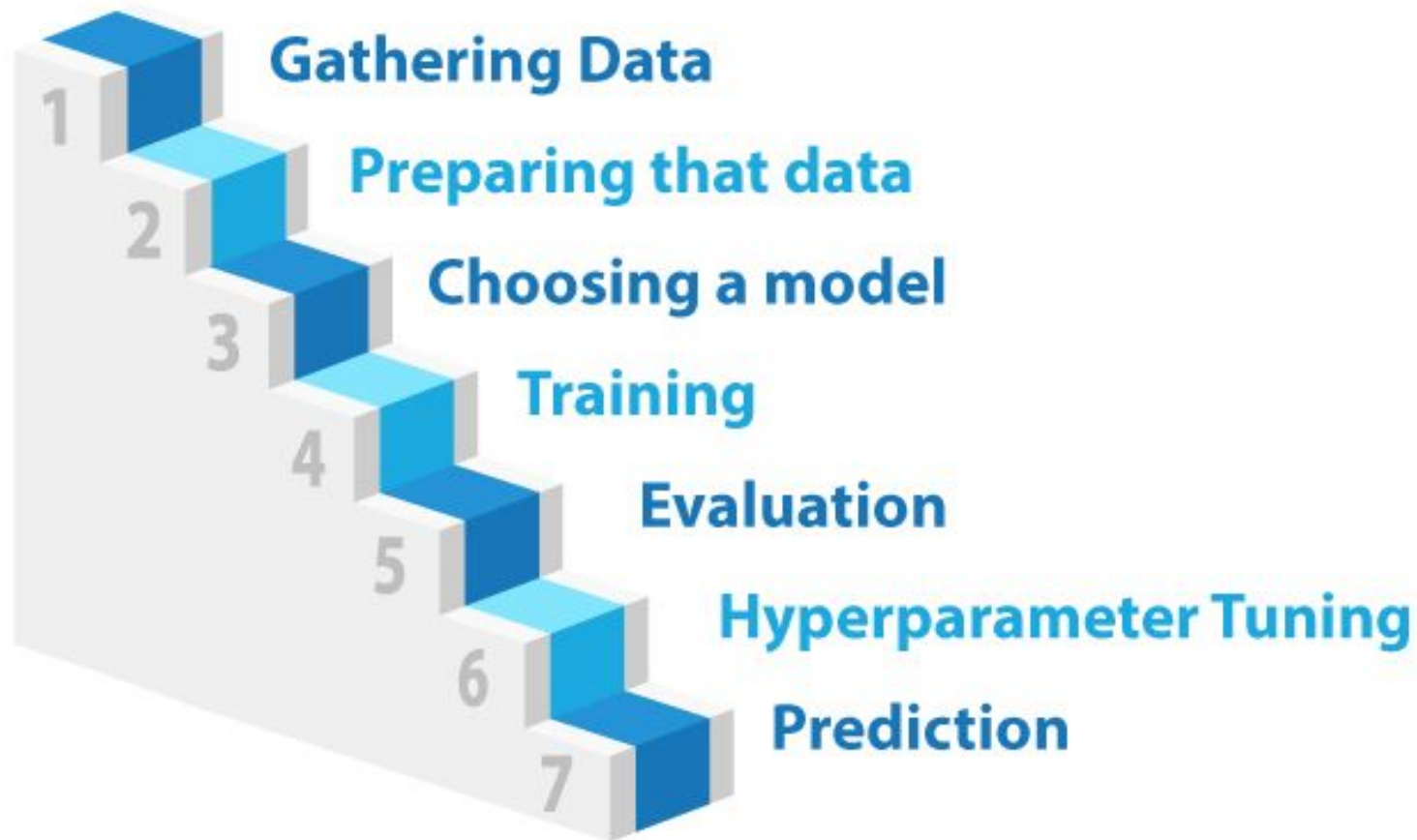
Traditional Programming vs ML

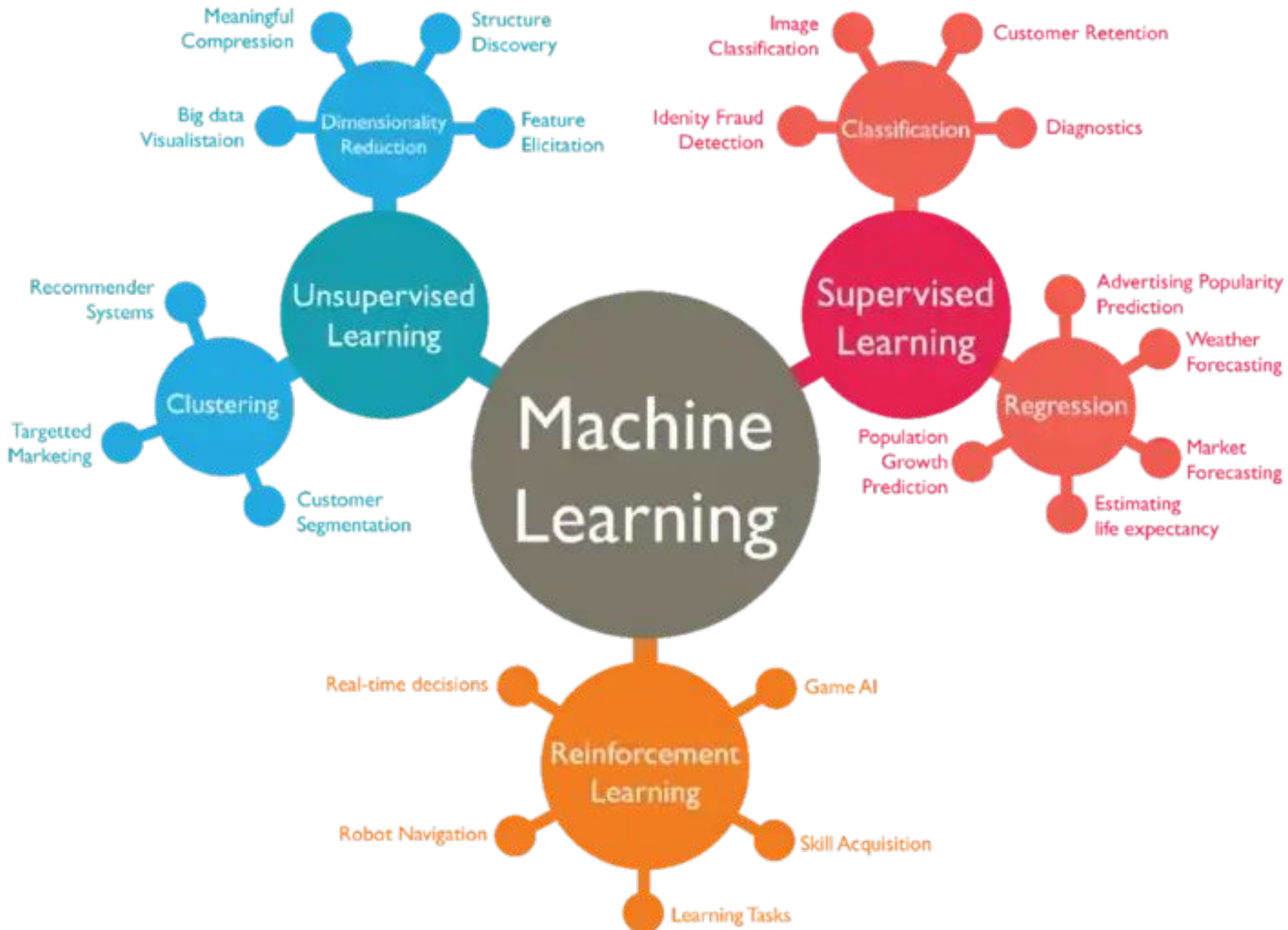


Difference Between AI, ML, and Deep Learning

Concept	Definition
Artificial intelligence	The field of computer science aims to create intelligent machines that can think and function like humans.
Machine learning	A subfield of artificial intelligence that focuses on developing algorithms and models that can learn from data rather than being explicitly programmed.
Deep learning	A subfield of machine learning that uses multi-layered artificial neural networks to learn complex patterns in data.

7 steps of Machine Learning





Source: Image source: Types of Machine Learning Algorithms by [Shrikumar Shankar](#).

Feature Engineering

Feature Engineering

- Extracting features from raw data
 - Raw data comes from logs, databases, protocol buffers, images, etc.
 - Can take up to ~ 80% of your time
- How do we deal with number data?
- How do we deal with string data?

Feature Engineering

- Transforms raw data into a feature vector of real numbers.

Raw Data

```
0 : {  
  house_info : {  
    num_rooms: 6  
    num_bedrooms: 3  
    street_name: "Shorebird Way"  
    num_basement_rooms: -1  
    ...  
  }  
}
```

Raw data doesn't come to us as feature vectors.

Feature Engineering

Feature Vector

```
[  
  6.0,  
  1.0,  
  0.0,  
  0.0,  
  0.0,  
  9.321,  
  -2.20,  
  1.01,  
  0.0,  
  ...,  
]
```

Process of creating features from raw data is **feature engineering**.

What to transform (encode)?

- Anything not represented as floating point number
 - Floating point numbers are used directly
 - Integers are easily converted
 - Beware of numerical data not intended to be multiplied by weights, e.g. zip codes
- String representation
 - A set of strings is a **vocabulary**, e.g. categorical features {red, orange, yellow, green, blue, indigo, violet}
 - A string not belonging to the set is **Out-of-Vocabulary (OOV)**
 - Can we assign a number to each string and be done?

Hot Encoding

- One-hot encoding - A binary vector for all strings in the set, with one element set to 1, all others set to 0
 - OOV can be represented by one of the elements
- Multi-hot encoding – A binary vector where multiple elements can be set to 1
- Effectively creates a Boolean value for every feature
 - The model uses only the weight for hot features
- Sparse representation – for large string sets, e.g. 10^6 strings, binary vector string representation is inefficient to store

Example: One-Hot Encoding

Human-Readable

Machine-Readable

Pet
Cat
Dog
Turtle
Fish
Cat



Cat	Dog	Turtle	Fish
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

-> 4 features

Need vector of size n classes (cat, dog, ...), which can be huge!

Example: Multi-Hot Encoding

- First label-encode your classes, thus having only a single number which represents the presence of a class (e.g. 1 for 'dog') and then convert the numerical labels to binary vectors of size $\log_2 n$, where n is number of classes.
 - Cat = 0 = [0,0]
 - Dog = 1 = [0,1]
 - Turtle = 2 = [1,0]
 - Fish = 3 = [1,1]

-> 2 features
- Can introduce false relationships, like dog + turtle = fish.
- Further reading
 - <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html>

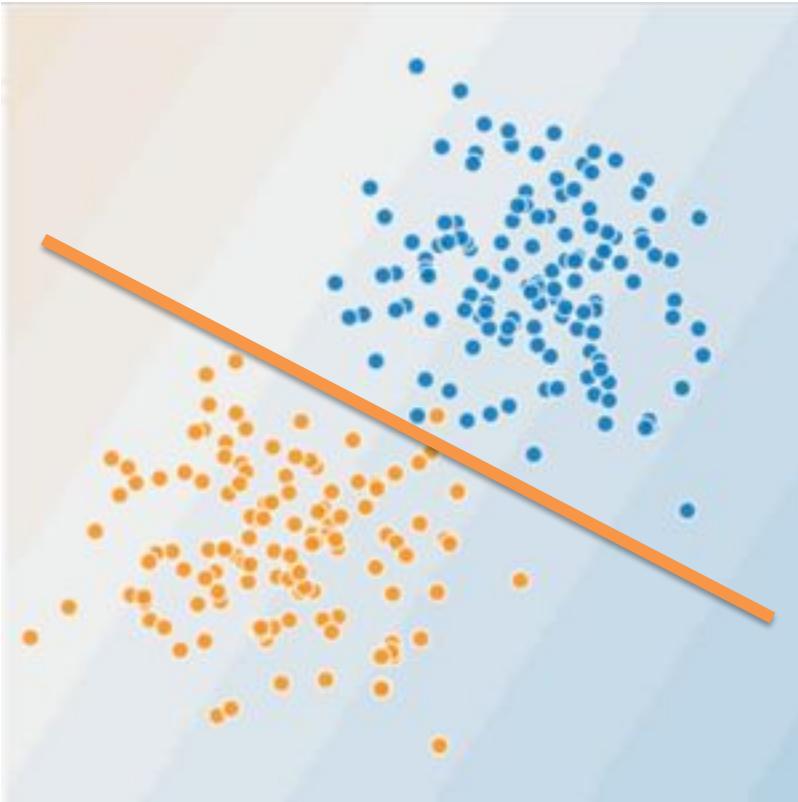
Qualities of Good Features

- Avoid rarely used discrete feature values
 - Feature values should appear at least 5 times in the data for model to see the feature with different labels
- Use clear and obvious feature names
 - Should be clear and obvious to anyone on the project
 - E.g. house_age_years
- Don't mix "magic" values with actual data
 - E.g using -1 for data not available for user_age_years
 - Instead, create Boolean feature user_age_available
 - For discrete values, creating a missing value, e.g. OOV
- The feature definition should not change over time
 - Watch out for dependencies on other models, systems, etc.
 - E.g. city_id = 123 (from another system) vs. city_id = "ny/new_york"

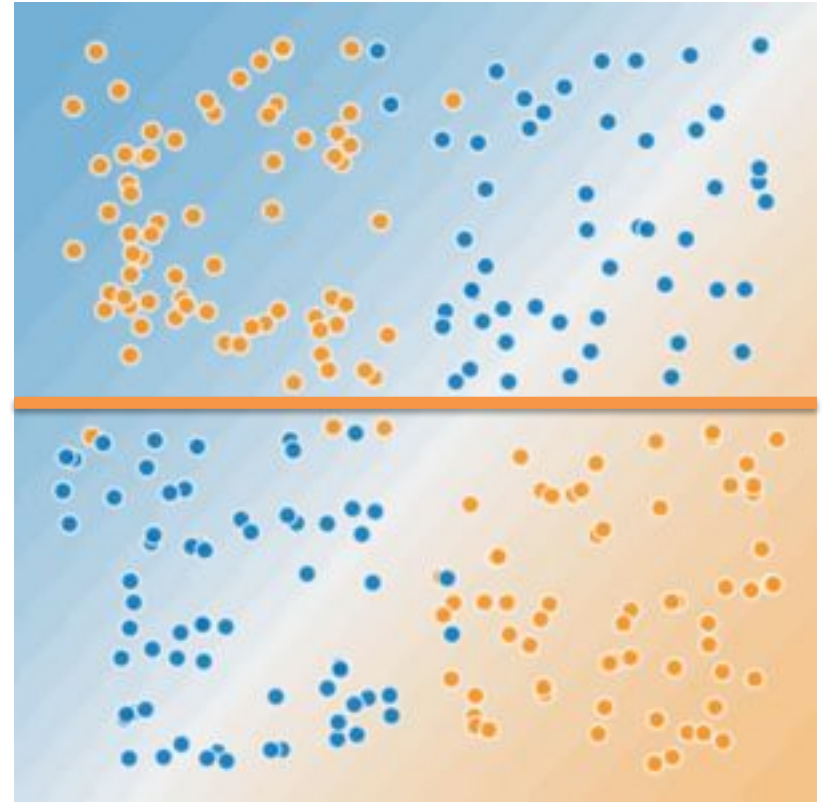
FEATURE CROSSES

Linear problem?

Can you draw a line that neatly separates the orange and blue data points?



Linear



Non-Linear

How to solve non-linear problems?

- One way is we turn them into a linear problem using synthetic features
- Create a feature cross (product)
 - a **synthetic feature** that encodes nonlinearity in the feature space by multiplying two or more input features together
 - “cross” comes from cross product
- For example, let's create a feature cross named x_3 by crossing x_1 and x_2 :
$$x_3 = x_1 x_2$$
- Treat this new feature like any other:
$$y = b + w_1 x_1 + w_2 x_2 + w_3 x_3$$
- The model training will determine w_3 like any other weight

Why do we do this?

- Linear learners use linear models
- Linear learners scale well to massive data
 - Vowpal Wabbit - Vowpal wabbit has been used to learn a tera-feature (10^{12}) data-set on 1000 nodes in one hour.
 - Sofia-ml
- Feature crosses + massive data = efficient strategy for highly complex models
 - Neural nets provide another strategy

Crossing One-Hot Vectors

- In practice, you are more likely to cross one-hot feature vectors, not continuous features
 - Like a logical conjunction
- For example, suppose you bin latitude and longitude
 - `binned_latitude` = [0, 0, 0, 1, 0]
 - `binned_longitude` = [0, 1, 0, 0, 0]
 - feature cross of these two feature vectors is a 25-element one-hot vector (24 zeroes and 1 one)
 - The single 1 in the cross identifies a particular conjunction of latitude and longitude.
 - Your model can then learn particular associations about that conjunction.
- Neural networks are a more sophisticated version of feature crosses
 - In essence, neural networks learn the appropriate feature crosses for you.

Feature Crossing Demo

- <https://developers.google.com/machine-learning/crash-course/feature-crosses/playground-exercises>

Feynman Technique

- Step 1 – Study
- Step 2 – Teach
- Step 3 – Fill the Gaps
- Step 4 – Simplify
- Good technique to practice in your study or project groups



Richard Feynman

“If you can’t explain it to a six-year-old, you don’t understand it yourself.”
– Albert Einstein



LOSS

- To build an ML model we need
 - A problem T
 - A performance measure P
 - An experience E

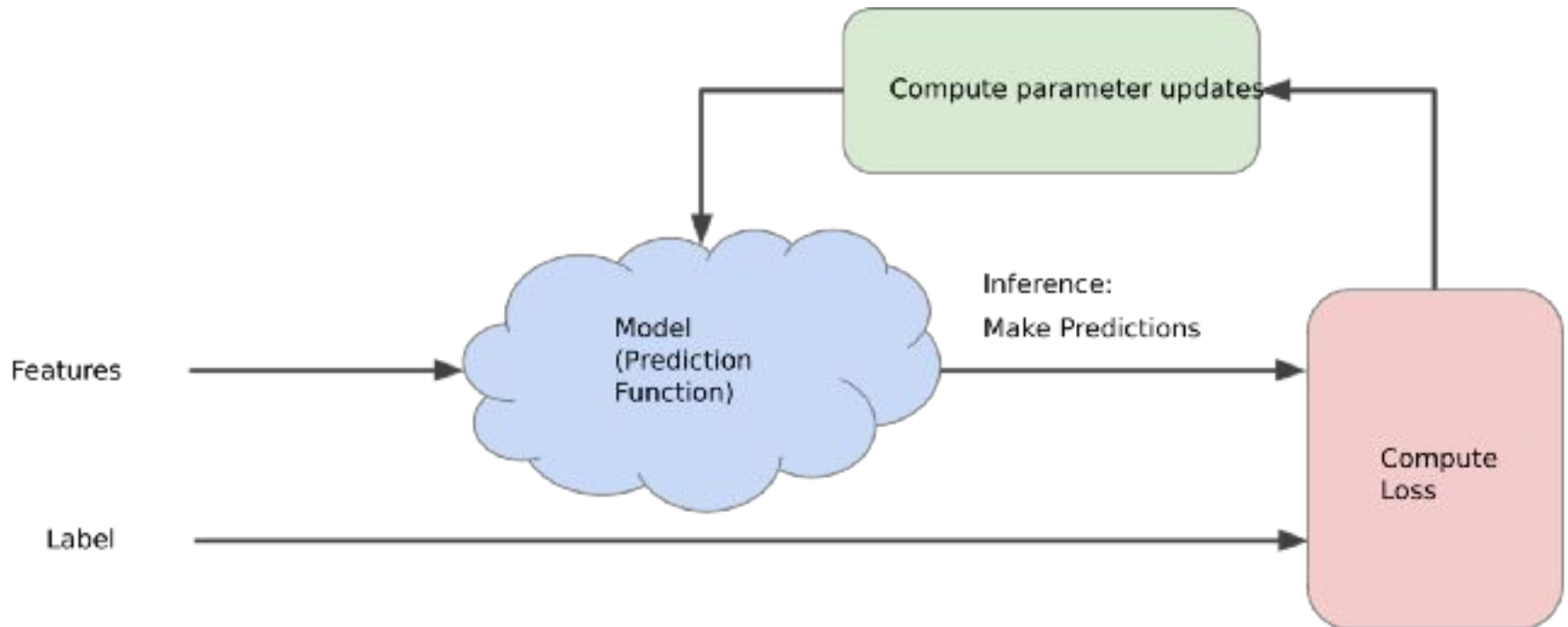
Training and Loss

- Training = learning (determining) good values (on average) for all the weights and the bias from (many) labeled examples (in supervised learning)
- “good values” => minimizes loss
 - Loss = number indicating how bad a single prediction is
 - Process of minimizing loss = **empirical risk minimization**

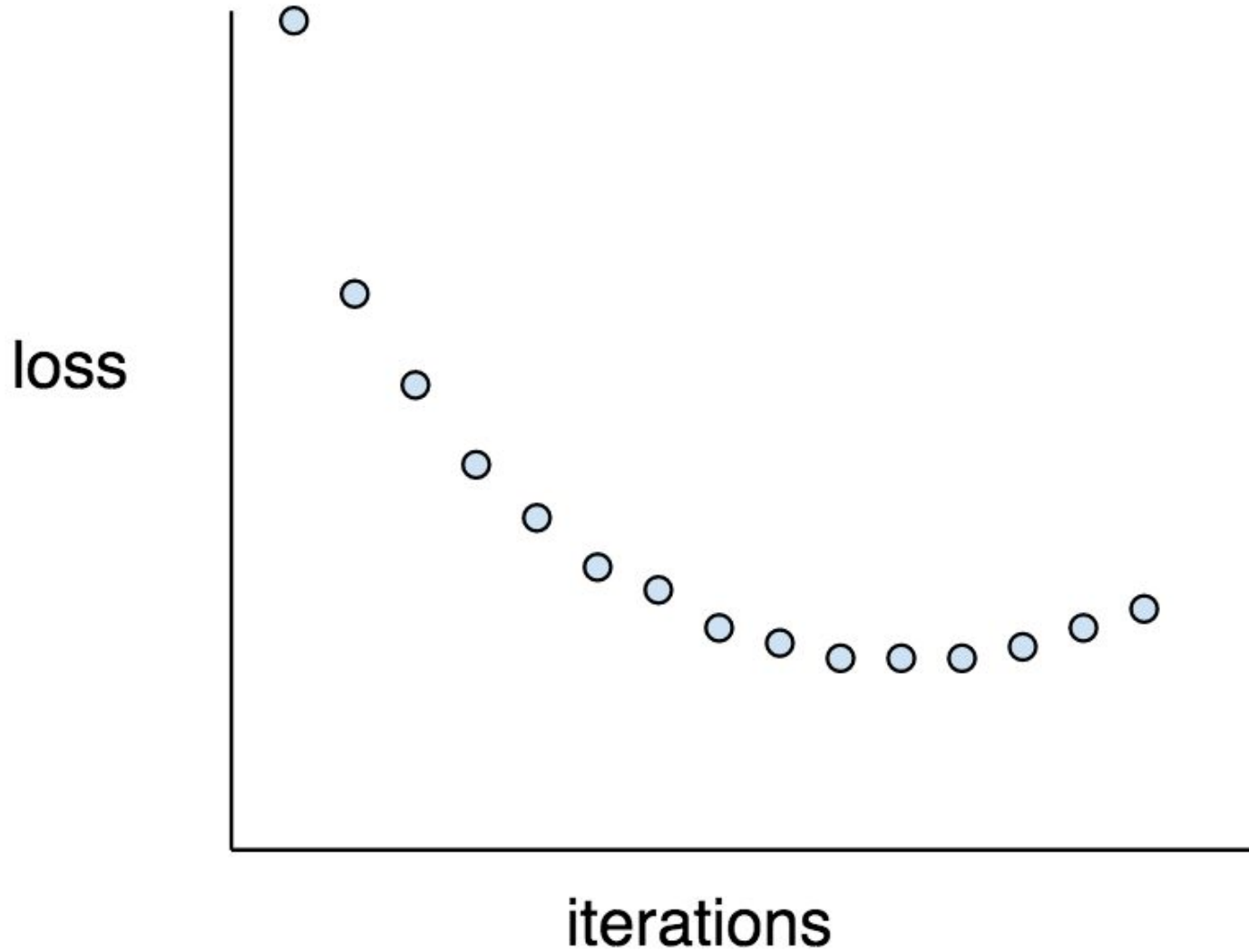
What is ML Loss?

- **A measure of how “bad” the model is**
- In other words, how far are a model's predictions from its labels
- Can use an iterative approach to reduce loss and determine the best model

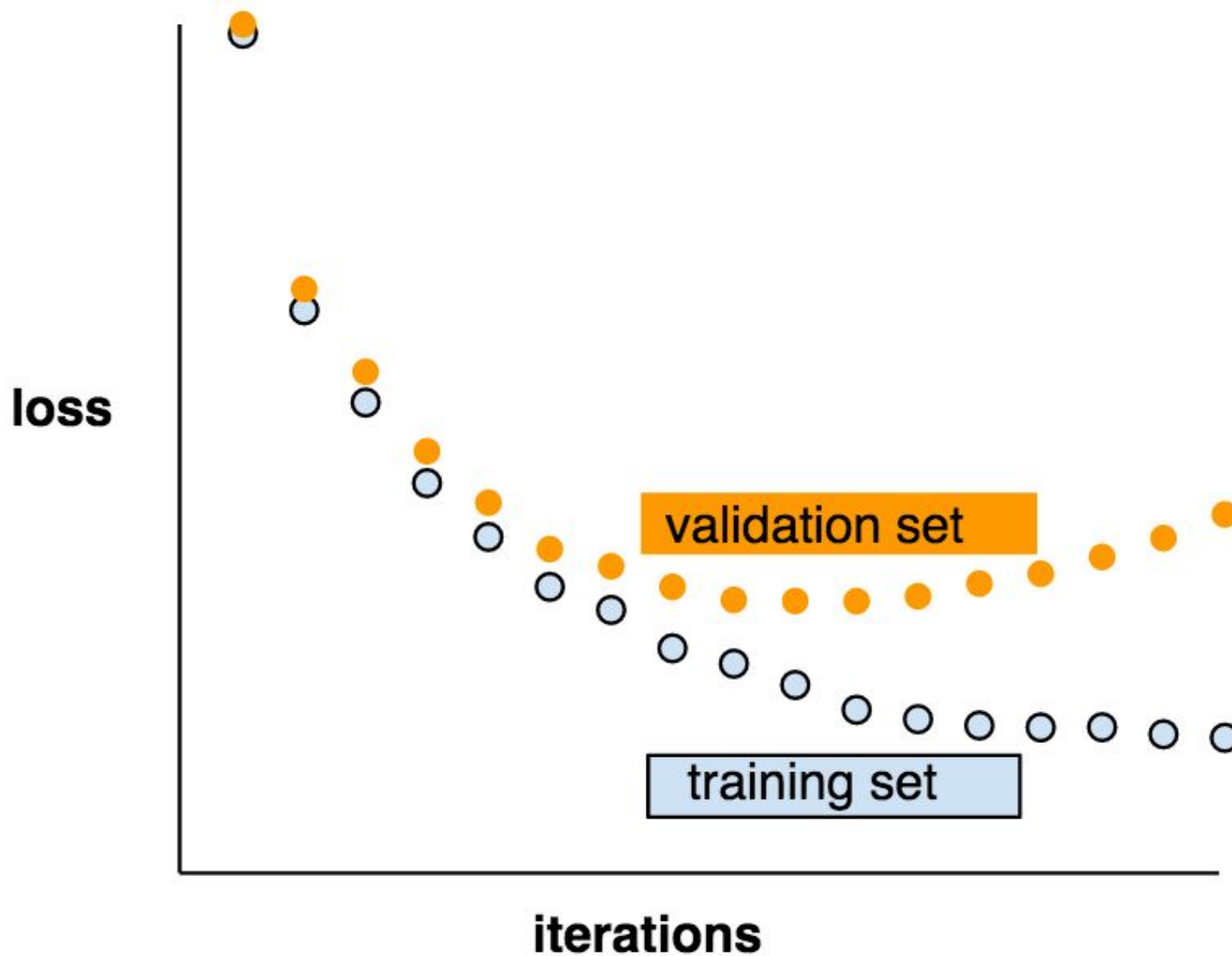
Iterative Approach



Loss Curve



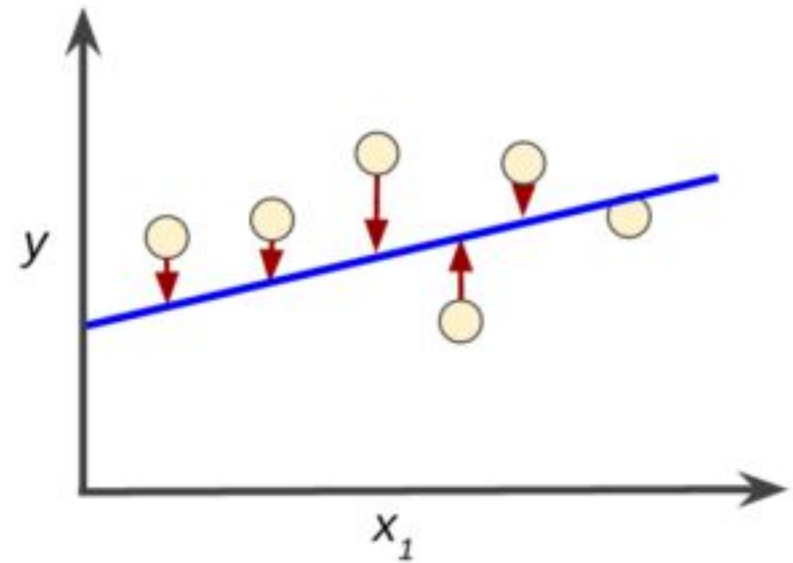
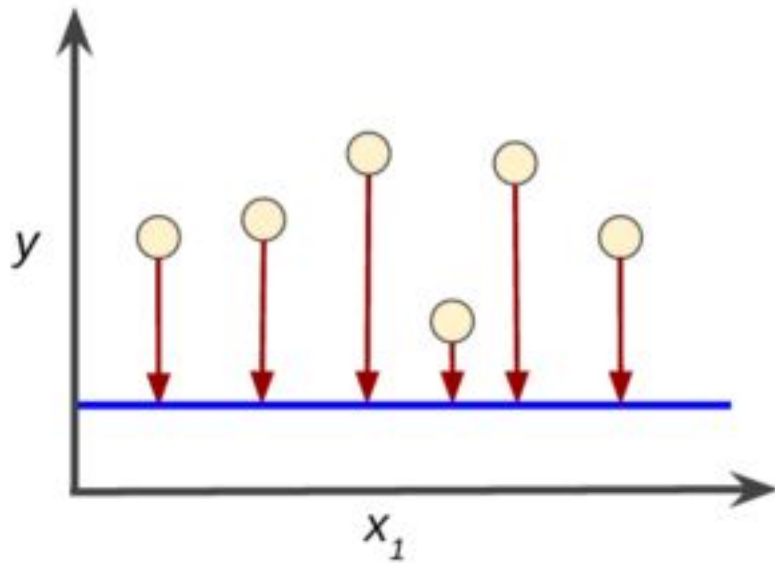
What's happening here?



Linear Regression Loss

- Curve fitting is not ML!
 - ML is looking for generalization that can be applied to new data
- $y = mx + b$
- In ML
 - $y' = b + w_1 x_1$
 - y' = predicted label
 - b = the bias (y-intercept), aka w_0
 - w_1 = weight of feature 1 (like slope m)
 - x_1 = feature (known input)
 - More features: $y' = b + w_1 x_1 + w_2 x_2 + w_3 x_3 + \dots$

High Loss vs Low Loss



What's your favorite loss function?

- Squared loss, or L_2 loss
- MSE = Mean square error is the average squared loss per example over the entire dataset

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

- (x, y) is an example in which x is the set of features (for example, Father's height, age, gender) that the model uses to make predictions and y is the example's label (for example, Son's height).
- $\text{prediction}(x)$ is a function of the weights and bias in combination with the set of features x .
- D is a data set containing many labeled examples, which are (x, y) pairs.
- N is the number of examples in D .

Other Loss functions

- Commonly used regression loss functions
 - MSE for linear regression models
 - Log Loss for Binary Logistic Regression

$$\text{Logloss}_i = -[y_i \ln p_i + (1 - y_i) \ln(1 - p_i)]$$

- Others
 - Mean Absolute Error (L1 Loss)
 - Mean Bias Error – calculates average bias (under/over estimate) in the model. Not usually used for training.

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n}$$

Mean absolute error

$$MBE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)}{n}$$

Mean bias error

Other Loss functions

- Classification losses
 - Hinge Loss/Multi class SVM Loss
 - Cross Entropy Loss/Negative Log Likelihood

$$SVM Loss = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

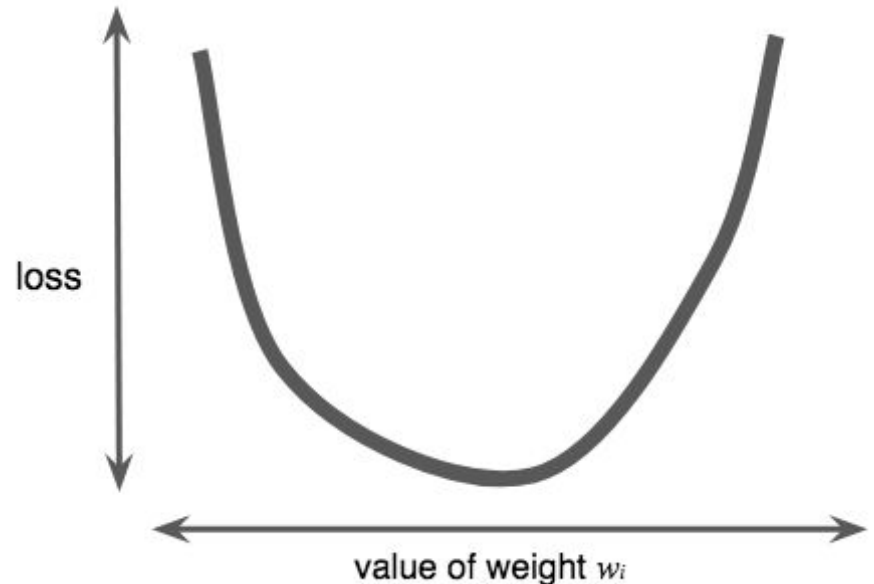
SVM Loss or Hinge Loss

$$CrossEntropyLoss = -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

Cross entropy loss

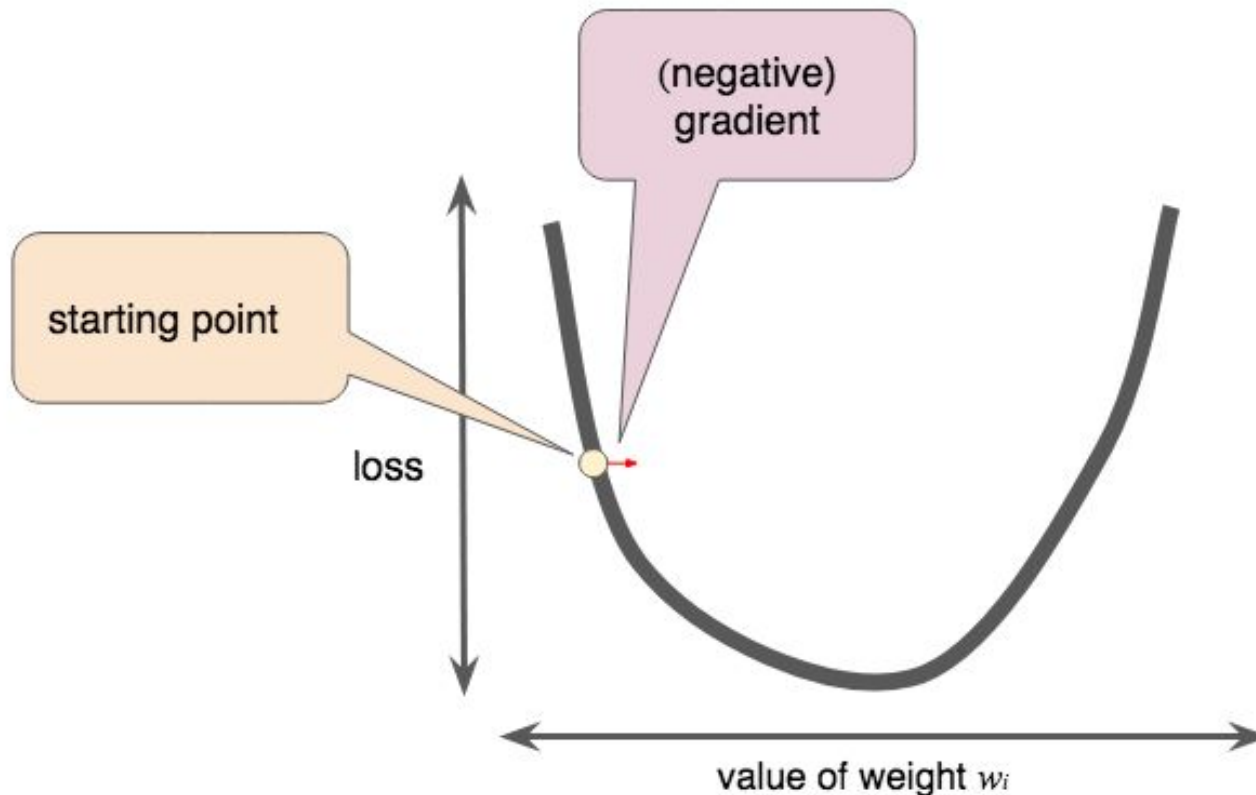
Let's revisit linear regression loss

- $y' = b + w_1x_1 + w_2x_2 + w_3x_3 + \dots = b + w_ix_i$
- We can calculate the loss for all values of w_i
 - ... but not efficient
- We know the loss is convex w.r.t. w_i , so what can we do that's more efficient?



Gradient Descent

- Minimize **loss** by computing the gradients of **loss** with respect to the model's parameter (e.g. weight)



Gradient Descent Procedure

- When performing gradient descent, we tune all the model parameters simultaneously.
- For example, to find the optimal values of both w_1 and the bias b , we calculate the gradients of the loss function with respect to both w_1 and b .
- Next, we modify the values of w_1 and b based on their respective gradients.
- Then we repeat these steps until we reach minimum loss.
- Note: Tools usually do this procedure for you automatically (e.g. TensorFlow)

Gradient Descent for Linear Regression Example

- Our model

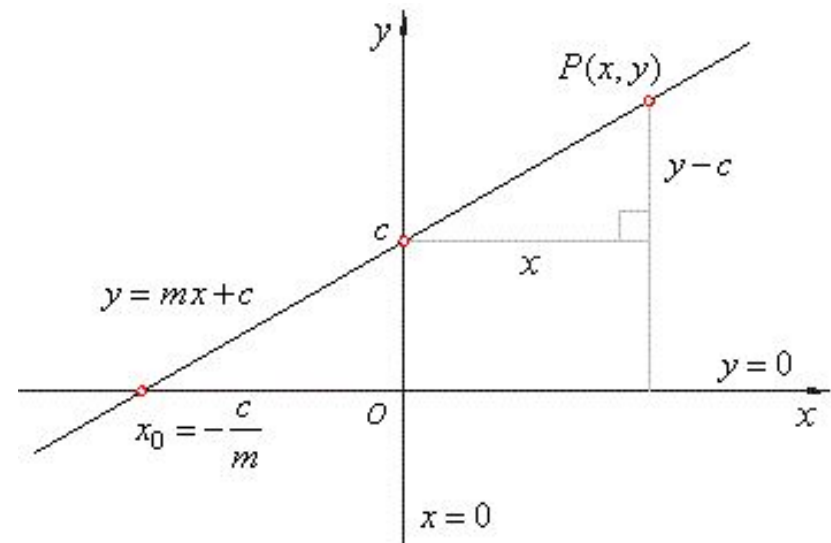
$$Y = mX + c$$

- Our loss function

$$E = \frac{1}{n} \sum_{i=0}^n (y_i - \bar{y}_i)^2$$

- with the predicted value in terms of weights

$$E = \frac{1}{n} \sum_{i=0}^n (y_i - (mx_i + c))^2$$



Gradient Descent for Linear Regression

Example

- Calculate the gradient
 - Partial derivative w.r.t. m
 - Partial derivative w.r.t. c

$$D_m = \frac{1}{n} \sum_{i=0}^n 2(y_i - (mx_i + c))(-x_i)$$

$$D_m = \frac{-2}{n} \sum_{i=0}^n x_i(y_i - \bar{y}_i)$$

$$D_c = \frac{-2}{n} \sum_{i=0}^n (y_i - \bar{y}_i)$$

- Update the parameters ([simulation](#))

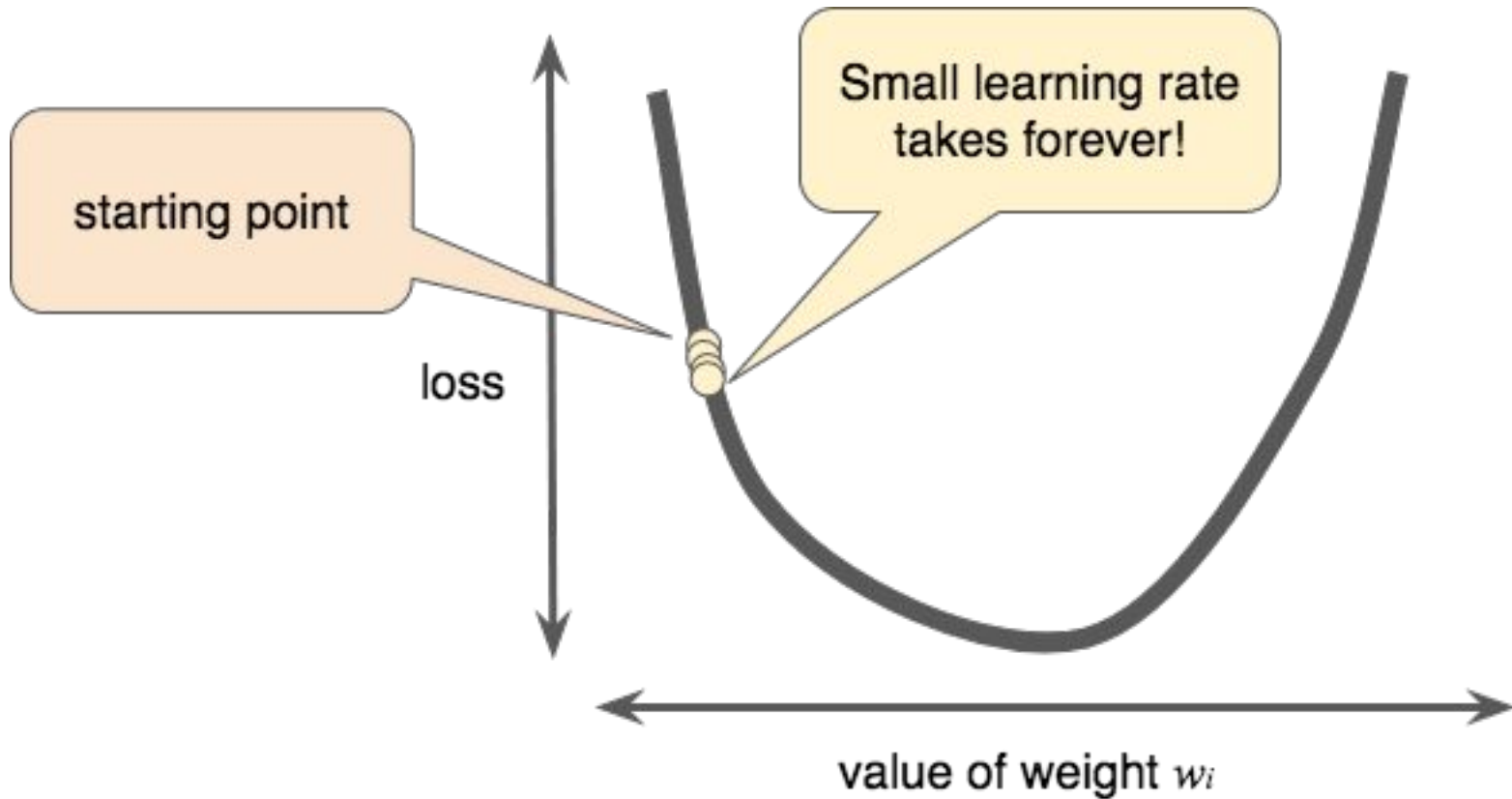
$$m = m - L \times D_m$$

$$c = c - L \times D_c$$

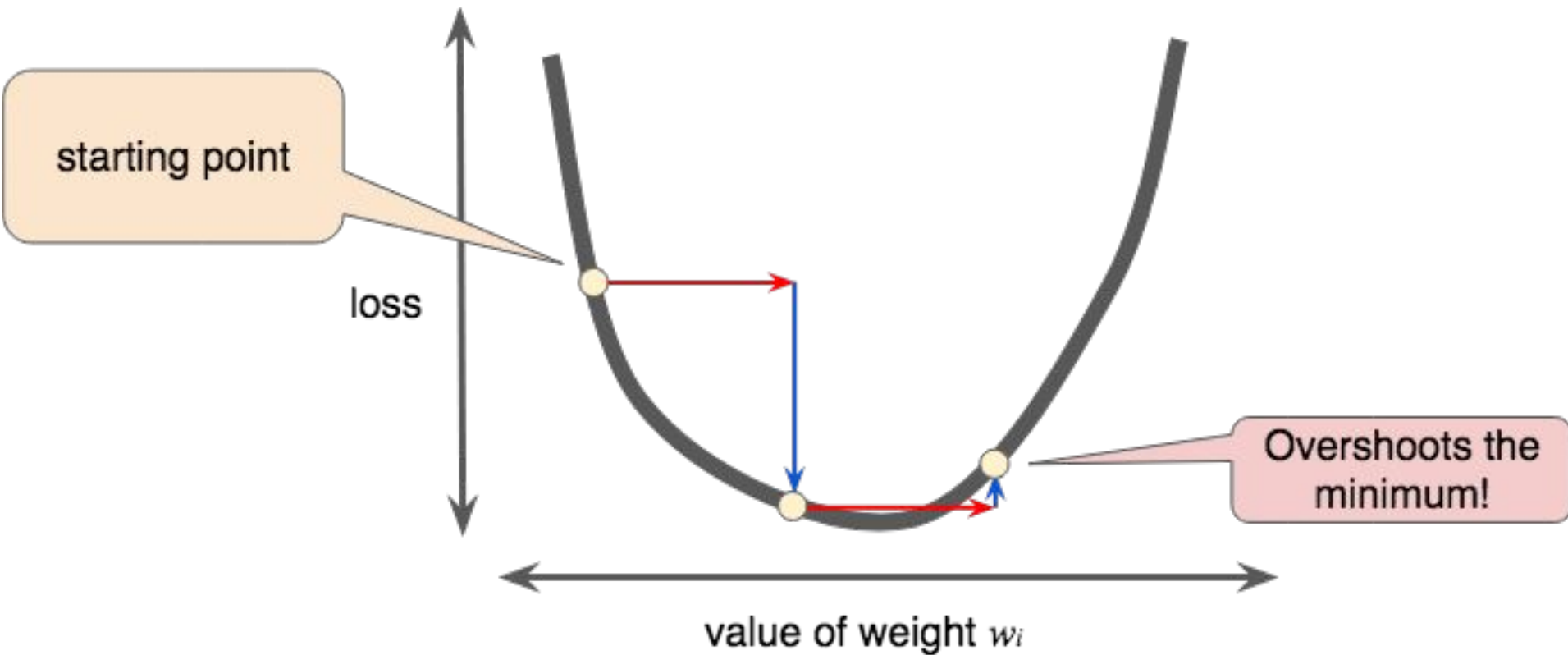
Learning Rate (L)

- The gradient vector has both a direction and a magnitude.
- Gradient descent algorithms multiply the gradient by a scalar known as the **learning rate** (also sometimes called step size) to determine the next point.
- For example, if the gradient magnitude is 2.5 and the learning rate is 0.01, then the gradient descent algorithm will pick the next point 0.025 away from the previous point.
- Any potential issues with the Learning rate we pick?
 - Think “Goldilocks & the Three Bears”

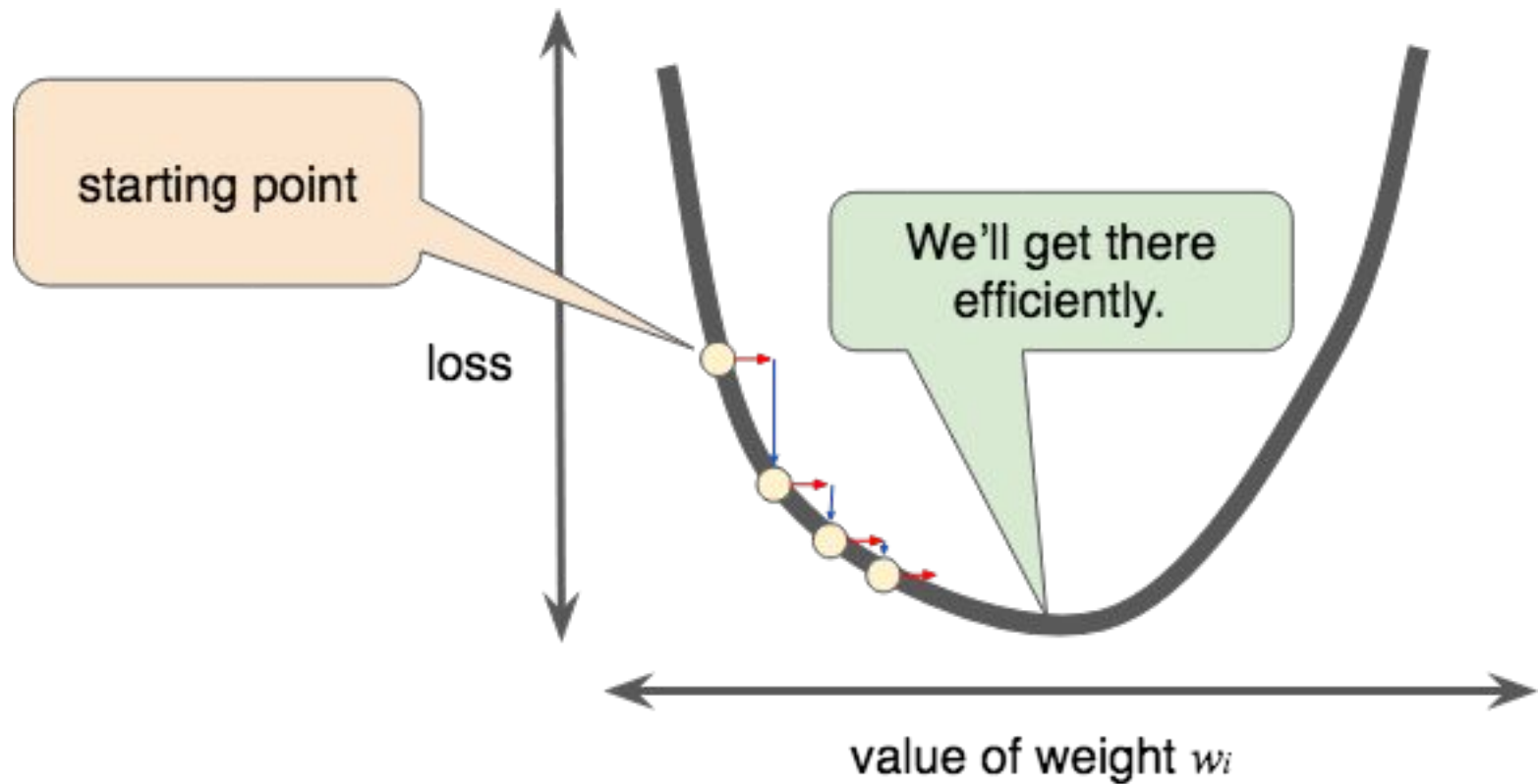
Learning Rate: Too Small



Learning Rate: Too Large



Learning Rate: Just Right



Optimal Learning Rate

- In one dimension, it's $\frac{1}{f''(x)}$
- For two or more dimensions, it's the **inverse** of the Hessian matrix

$$\mathbf{H}_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}, \quad (\mathbf{H}_f)_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}.$$

Gradient

- The partial derivative of a function is one of the most important and common math concepts in ML.
- Among other things, a partial derivative gives you
 - the direction (the sign of the gradient) and
 - magnitude (magnitude of gradient) of how to adjust constants in order to minimize some sort of error.

Gradient, Jacobian, Hessian, oh my!

- Gradient: Vector of first order derivatives of a scalar field
 - direction and rate of fastest increase

$$\nabla f = \frac{\partial f}{\partial x} \mathbf{i} + \frac{\partial f}{\partial y} \mathbf{j} + \frac{\partial f}{\partial z} \mathbf{k},$$

- Jacobian: Matrix of gradients for components of a vector field

$$\mathbf{J}_{i,j} = \frac{\partial F_i}{\partial x_j}$$

- The matrix of all first-order partial derivatives of a multivariate function
- Hessian: Matrix of second order mixed partials of a scalar field.

$$\mathbf{H}_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

ML and the Hessian

Suppose you want to minimize some error function E which depends on a set of weights: minimize $E(W)$ where $W = (w_1, w_2, w_3, \dots)$

Let a vector \mathbf{a} be guesses of the weight values that minimize E . And suppose the difference between the guesses \mathbf{a} and the solution weights \mathbf{W} is epsilon (Greek lower case looks-like-'e'). So:

$$\mathbf{a} = (a_1, a_2, \dots, a_n)$$

$$\mathbf{W} = (a_1 + \epsilon_1, a_2 + \epsilon_2, \dots, a_n + \epsilon_n)$$

We want to minimize, so we take the Calculus derivative and set to zero. This is a nasty problem so we estimate the derivatives (i.e., "gradient") using Taylor's formula like this:

$$\frac{\partial E(\mathbf{a} + \boldsymbol{\epsilon})}{\partial w_i} \approx \frac{\partial E(\mathbf{a})}{\partial w_i} + \sum_j \frac{\partial^2 E(\mathbf{a})}{\partial w_i \partial w_j} \epsilon_j + (\text{other terms in Taylor's formula})$$

Now, if we rewrite this equation using matrix notation, we get:

$$\nabla E(\mathbf{a} + \boldsymbol{\epsilon}) = \nabla E(\mathbf{a}) + H_f(\mathbf{a})\boldsymbol{\epsilon}_j$$

Solving for epsilon and abusing notation a bit:

$$\boldsymbol{\epsilon} = -1 * \left(H_f(\mathbf{a}) \right)^{-1} \nabla E(\mathbf{a})$$

ML and the Hessian

- Procedure: Add the ϵ to the guesses and repeat
 - evaluate the Hessian (all second derivatives) at the guesses \mathbf{a}
 - invert the matrix
 - then multiply by the gradient of the error function at \mathbf{a}
- Notice
 - For $n = 10$ weights, the Hessian is 10×10
 - If n gets very large, this direct approach will not work
 - There are several variations of this technique that estimate the Hessian without direct calculation
 - Finding a "perfect" learning rate is not essential for successful model training. The goal is to find a learning rate large enough that gradient descent converges efficiently, but not so large that it never converges. Convergence becomes more important the larger the data set.

Batch Size for Calculating the Gradient

- **Stochastic Gradient Descent:** relies on a **single** example chosen uniformly at random from a dataset to calculate an estimate of the gradient at each step
 - Batch size = 1
- **Mini-batch:** A small, randomly selected **subset** of the entire batch of examples run together in a single iteration of training or inference.
 - The batch size of a mini-batch is usually between 10 and 1,000.
 - It is much more efficient to calculate the loss on a mini-batch than on the full training data.
- **Full-batch:** **All** examples used in one iteration (that is, one gradient update) of model training.
- TensorFlow allows dynamic batch sizes

(Hyper)Parameters

- Parameters are estimated from the dataset. They are part of the *model equation*. Examples:
 - Weights
- Hyperparameters are set manually to help in the estimation of the model parameters. They are not part of the final *model equation*.
 - A parameter whose value is initialized before the learning takes place.
 - Examples:
 - Learning rate α
 - Regularization rate λ

When to tune the Learning Rate?

- **Learning rate trade-off: learning optimal parameters (e.g. weights) vs. time to train**
- Given a perfectly configured learning rate, the model will learn to best approximate the function given available resources in a given number of training epochs (passes through the training data).
- Generally, **a large learning rate allows the model to learn faster, at the cost of arriving on a sub-optimal final set of weights**. A smaller learning rate may allow the model to learn a more optimal or even globally optimal set of weights but may take significantly longer to train.
- At extremes, a learning rate that is too large will result in weight updates that will be too large and the performance of the model (such as its loss on the training dataset) will oscillate over training epochs. Oscillating performance is said to be caused by weights that diverge (are divergent). A learning rate that is too small may never converge or may get stuck on a suboptimal solution.
- Further Reading
 - <https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/>

Hyperparameter Optimization

- **Search Space** - Volume to be searched where each dimension represents a hyperparameter and each point represents one model configuration.
- **Random Search** - Define a search space as a bounded domain of hyperparameter values and randomly sample points in that domain.
- **Grid Search** - Define a search space as a grid of hyperparameter values and evaluate every position in the grid.
- More info:
 - <https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/>
 - <https://towardsdatascience.com/grid-search-for-model-tuning-3319b259367e>

Sparse Representation

- Matrices that contain mostly zero values are called **sparse**, distinct from matrices where most of the values are non-zero, called **dense**.
- It is computationally expensive to represent and work with sparse matrices as though they are dense, and much improvement in performance can be achieved by using representations and operations that specifically handle the matrix sparsity.
- The sparsity of a matrix can be quantified with a sparsity score:
number of zero elements / total number of elements
- Problems with Sparsity
 - **Space Complexity** - Very large matrices require a lot of memory, and some very large matrices that we wish to work with are sparse.
 - **Time Complexity** - Assuming a very large sparse matrix can be fit into memory, we will want to perform operations on this matrix.
 - Simply, if the matrix contains mostly zero-values, i.e. no data, then performing operations across this matrix may take a long time where the bulk of the computation performed will involve adding or multiplying zero values together.

Working with Sparse Matrices

- The solution to representing and working with sparse matrices is to use an alternate data structure to represent the sparse data.
- There are multiple data structures that can be used to efficiently construct a sparse matrix; three common examples are listed below.
 - Dictionary of Keys. A dictionary is used where a row and column index is mapped to a value.
 - List of Lists. Each row of the matrix is stored as a list, with each sublist containing the column index and the value.
 - Coordinate List. A list of tuples is stored with each tuple containing the row index, column index, and the value.
- There are also data structures that are more suitable for performing efficient operations; two commonly used examples are listed below.
 - Compressed Sparse Row. The sparse matrix is represented using three one-dimensional arrays for the non-zero values, the extents of the rows, and the column indexes.
 - Compressed Sparse Column. The same as the Compressed Sparse Row method except the column indices are compressed and read first before the row indices.
- More info
 - <https://machinelearningmastery.com/sparse-matrices-for-machine-learning/>