ML NOTES:

- In **unsupervised learning**, the goal is to identify meaningful patterns in the data. To accomplish this, the machine must learn from an unlabeled data set. In other words, the model has no hints how to categorize each piece of data and must infer its own rules for doing so.

- For example,
  Here, we have two clusters. (Note that the number of clusters is arbitrary). What do these clusters represent? It can be difficult to say. Sometimes the model finds patterns in the data that you don't want it to learn, such as stereotypes or **bias**.

- **Note: While it is very common, clustering is not the only type of unsupervised learning.**

- In **Reinforcement Learning** you don't collect examples with labels. Imagine you want to teach a machine to play a very basic video game and never lose. You set up the model (often called an agent in RL) with the game, and you tell the model not to get a "game over" screen. During training, the agent receives a reward when it performs this task, which is called a reward function.

- The lack of a data requirement makes RL a tempting approach. However, designing a good reward function is difficult, and **RL models are less stable and predictable** than supervised approaches.
-

Type of ML Problem	Description	Example
Classification	Pick one of N labels	Cat, dog, horse, or bear
Regression	Predict numerical values	Click-through rate
Clustering	Group similar examples	Most relevant documents (unsupervised)
Association rule learning	Infer likely association patterns in data	If you buy hamburger buns, you're likely to buy hamburgers (unsupervised)
Structured output	Create complex output	Natural language parse trees, image recognition bounding boxes
Ranking	Identify position on a scale or status	Search result ranking

- Many machine learning systems produce models that encode knowledge and intelligence by interpreting signals differently than humans do. A neural network might interpret a word via an **embedding**, so "tree" is understood as something like, [0.37, 0.24, 0.2] and "car" as [0.1, 0.78, 0.9]. The neural network might use these representations to do accurate translations or sentiment analysis, but a human looking at the embeddings would find them very hard to understand.

- **Hard ML Problems and Why is Hard?**

    - ♣ Clustering: How to cluster if you have 1000-dim data and you have no idea about the shape of data.
    - ♣ Anomaly Detection: How to decide a sample of data is anomaly?
    - ♣ Causation: ML can identify correlations—mutual relationships or connections between two or more things. Determining causation (one event or factor causing another) is much harder. In other words, it is easy to see that something happened, but much harder to understand why it happened. In general, you need to intervene in the world—run an experiment—to determine causation; you can't see it in purely observational data.

- **Supervised ML:** ML systems learn how to combine input to produce useful predictions on never-before-seen data.

- **Training** means creating the model. That is, you show the model labeled examples and enable the model to gradually learn the relationships between features and label.

- A **Model** defines the relationship between features and label.

- **Inference** means applying the trained model to unlabeled examples. **Inference = make predictions.**

- A **regression** model predicts continuous values.

- A **classification** model predicts discrete values.

- **Linear regression** is a method for finding the straight line or hyperplane that best fits a set of points.
- **L2 Loss (Squared Loss)** = $(\text{observation} - \text{prediction})^2$

$$L_2 Loss = \sum_{(x,y)\in D} (y - prediction(x))^2$$

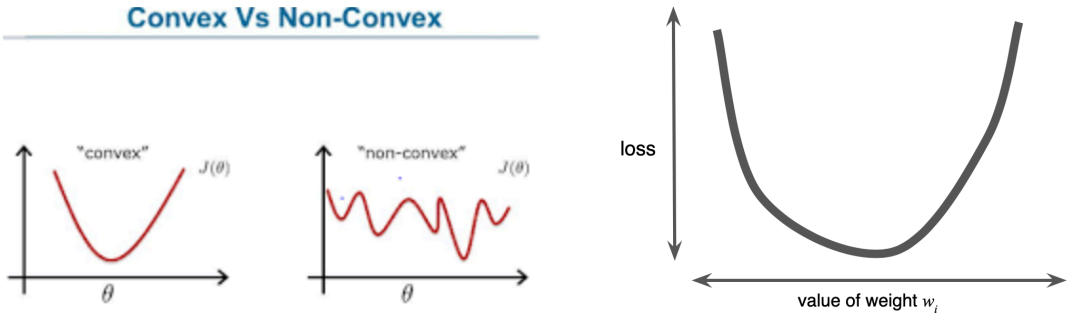
$\sum$:We're summing over all examples in the training set.
D: Sometimes useful to average over all examples, so divide by $\|D\|$.

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

- In general, **Linear Regression** uses **L2 Loss.**

- **NOTE: In practice, finding a "perfect" (or near-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.**

- **Convex Problems:** They're shaped like a giant bowl. Convex problems have only one minimum; that is, only one place where the slope is exactly 0. That minimum is where the loss function converges. Local minima = global minima. You can initiate the weights anywhere; it does converge somehow. However, many ML problems are **non-convex.** For example, neural networks. For each local minima might not be equal the global minima. Strong dependency on initial weight values.



- Calculating the loss function for every conceivable value of w_i over the entire data set would be an inefficient way of finding the convergence point. Let's

examine a better mechanism—very popular in machine learning—called **gradient descent**.

- The **gradient** always points in the direction of steepest increase in the **loss** function.
-
- In **gradient descent**, a **batch** is the total number of examples you use to calculate the gradient in a single iteration.

  Stochastic Gradient Descent / Online Gradient Descent: Compute loss & gradients over one sample at a time.

  Batch Gradient Descent: Compute loss & gradients over entire training data set. (Loss & gradients are averaged over size of the data set.)

  Mini-batch Gradient Descent: Compute loss & gradients over more than 1 and less than total number of samples in the training set. (Loss & gradients are averaged over the mini-batch.)

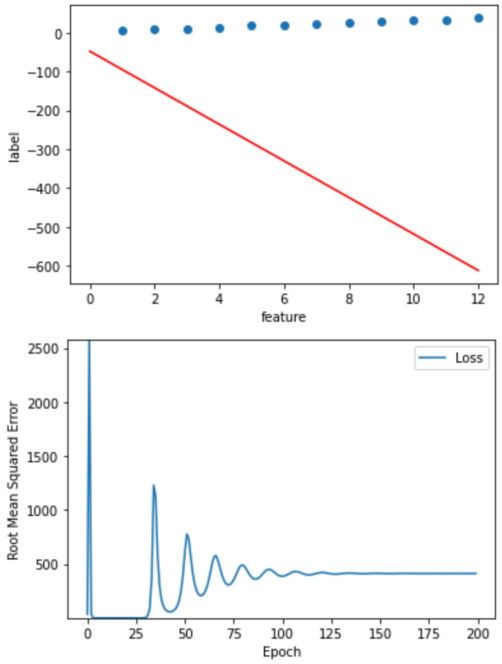
- **Why to use SGD or Mini-Batch GD instead of BGD?**

  - ♣ A very large batch may cause even a single iteration to take a very long time to compute. (Real world data sets often contain billions or hundreds of billion samples and huge number of features for each sample.)

  - ♣ Even if you choose samples from the data set randomly, a large data set with randomly sampled examples probably contains redundant data. In fact, redundancy becomes more likely as the batch size grows. Some redundancy can be useful to smooth out noisy gradients, but enormous batches tend not to carry much more predictive value than large batches.

- **Why to use Mini-Batch GD instead of SGD?**
  - ♣ Given enough iterations, SGD works but is very noisy.

- Training loss should steadily decrease, steeply at first, and then more slowly. Eventually, training loss should eventually stay steady (zero slope or nearly zero slope), which indicates that training has converged.

- 

The resulting model is terrible; the red line doesn't align with the blue dots. Furthermore, the loss curve oscillates like a roller coaster. An oscillating loss curve strongly suggests that the learning rate is too high.

- **Lowering the learning rate** while **increasing the number of epochs or the batch size** is often a good combination.

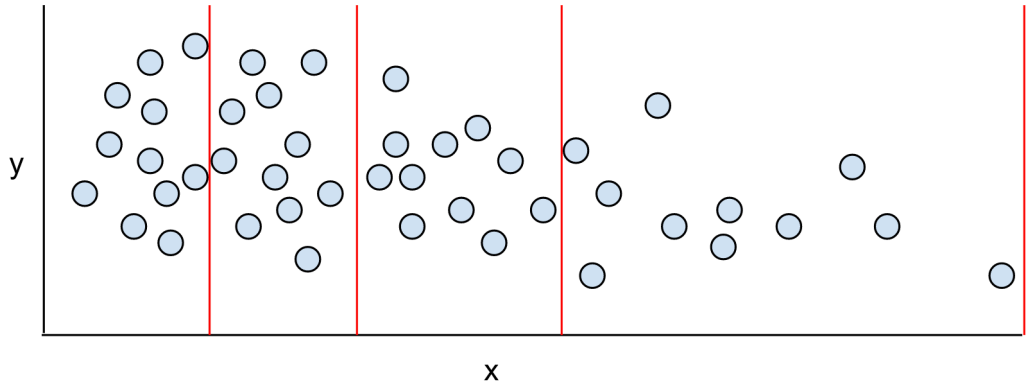
- Setting the batch size to a very **small batch number** can also cause **instability.** First, try large batch size values. Then, decrease the batch size until you see degradation.

- **Remember:** the ideal combination of hyperparameters is data dependent, so you must always experiment and verify.

- **Quantile bucketing:**

  quantile bucketing

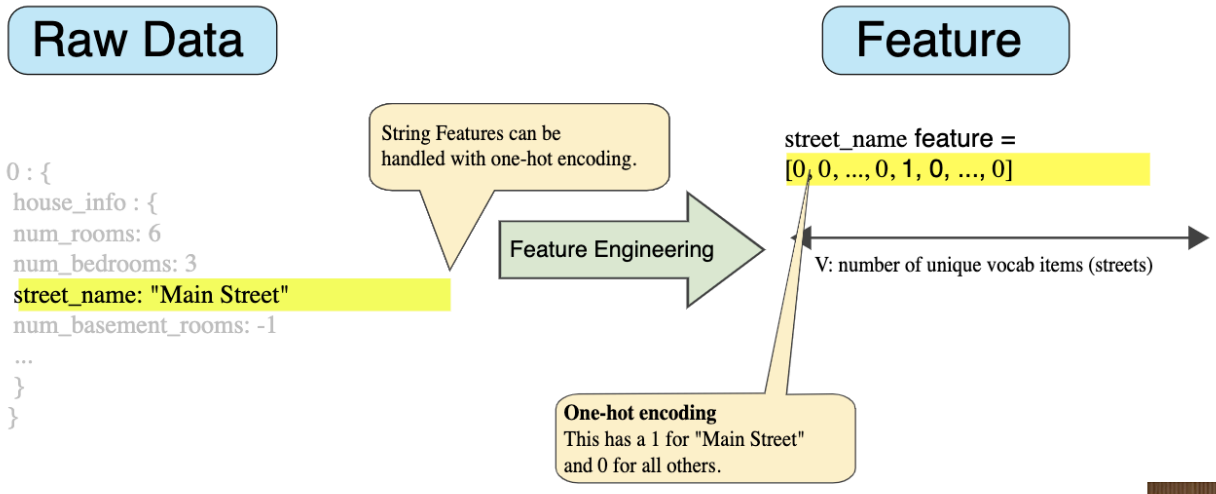
  Distributing a feature's values into **buckets** so that each bucket contains the same (or almost the same) number of examples. For example, the following figure divides 44 points into 4 buckets, each of which contains 11 points. In order for each bucket in the figure to contain the same number of points, some buckets span a different width of x-values.



- An **overfit** model gets a low loss during training but does a poor job predicting new data. Overfitting is caused by making a model more complex than necessary.
- **generalization bounds:**
  - the complexity of the model
  - the model's performance on training data
- The following three basic assumptions guide **generalization:**
  - We draw examples **independently and identically** (**i.i.d**) at random from the distribution. In other words, examples don't influence each other. (An alternate explanation: i.i.d. is a way of referring to the randomness of variables.)
  - The distribution is **stationary**; that is the distribution doesn't change within the data set.
  - We draw examples from partitions from the **same distribution.**
- **To prevent Overfitting:**
  - Apply Cross-Validation
  - Training with more data
  - Remove some features
  - Stop training early
  - Apply regularization
  - Apply ensemble methods.

- **One Hot Encoding:**

Translate string value into a feature vector. For example, if we have a one hot encoding for street names, we'll have a unique coefficient for each possible street. One hot encoding is extremely handy for sparse, categorical data.



- Dictionary maps each street name to an int in `{0, ...,V-1}`
- Now represent one-hot vector above as <*i*>

Tensorflow:

- TensorFlow, as the name indicates, is a framework to define and run computations involving tensors
- A tensor is a generalization of vectors and matrices to potentially higher dimensions.
- Colab is Google's version of Jupyter Notebook.

- **Tensorflow vs Keras**

  TensorFlow is an end-to-end open source platform for machine learning. It's a comprehensive and flexible ecosystem of tools, libraries and other resources that provide workflows with high-level APIs.

  Keras, on the other hand, is a high-level neural networks library which is running on the top of TensorFlow, CNTK, and Theano. Using Keras in deep learning allows for easy and fast prototyping as well as running seamlessly on CPU and GPU. This framework is written in Python code which is easy to debug and allows ease for extensibility.

  There are several differences between these two frameworks. Keras is a neural network library while TensorFlow is the open source library for a number of various tasks in machine learning. TensorFlow provides both high-level and low-level APIs while Keras provides only high-level APIs.