Chapter 4: Anatomy of a Learning Algorithm

Dr. Sultan Alfarhood

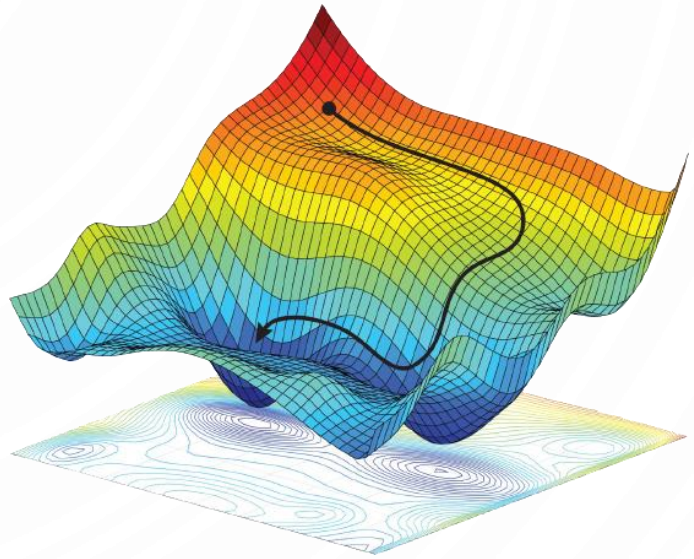
Building Blocks of a Learning Algorithm

1. A **loss function**
   - A function defined on one data point, prediction and label, and measures the penalty.
2. An optimization criterion based on the loss function (i.e., a **cost function** such as MSE)
   - A sum of loss functions over all training set.
3. An **optimization function**/routine leveraging training data to find a solution to the optimization criterion (i.e., Gradient Descent)
   - General term for any function that you optimize during training.

Optimization function/algorithm

By reading modern literature on machine learning, you often encounter references to:

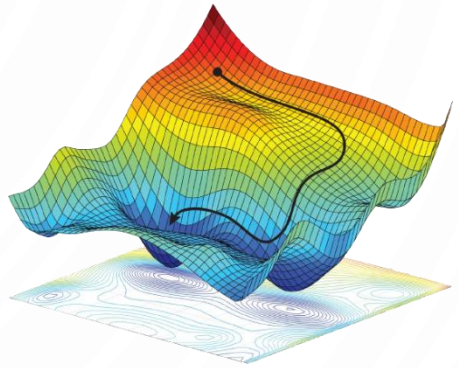
- Gradient Descent
- Stochastic Gradient Descent (SGD)



Gradient Descent

Gradient Descent

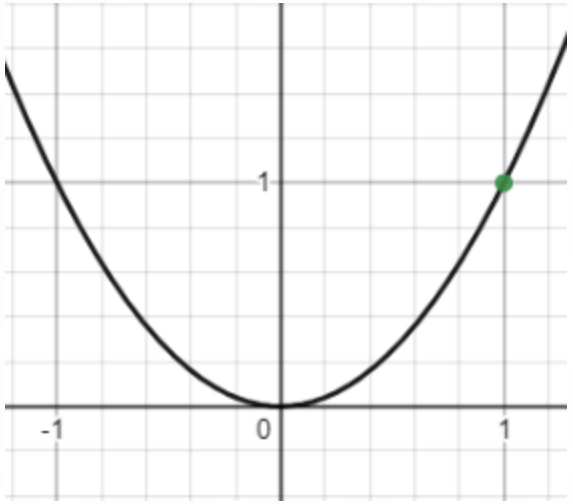
- An iterative optimization algorithm used to find local minima of a function (typically, a cost function.)
- Gradient descent can be used to find optimal parameters for linear and logistic regression, SVM, and neural networks.



Derivative and Gradient

- A derivative f' of a function f is a function or a value that describes **how fast f grows** (or decreases).
  - If the derivative is a **constant** value, like 5 or −3, then the function grows (or decreases) **constantly** at any point x of its domain.
  - If the derivative f' is a **function**, then the function f can grow at a **different pace** in different regions of its domain.
  - If the derivative f' is **positive** at some point x, then the function f **grows** at this point.
  - If the derivative of f is **negative** at some x, then the function **decreases** at this point.
  - The derivative of **zero** at x means that the function's slope at x is horizontal.
- A **gradient** of a function is a vector of partial derivatives.

$$f(x) = x^2$$
$$f' = 2x$$



How Gradient Descent Works (Linear Regression)

- **Goal:** Find w and b values that minimize the **Mean Squared Error (MSE)** $l \stackrel{\text{def}}{=} \frac{1}{N}\sum_{i=1}^{N}(y_i - (wx_i + b))^2.$

1. Start with random or 0 values for w and b.
2. Calculate gradient (partial derivative for every parameter):

$$\frac{\partial l}{\partial w} = \frac{1}{N}\sum_{i=1}^{N} -2x_i(y_i - (wx_i + b));$$

$$\frac{\partial l}{\partial b} = \frac{1}{N}\sum_{i=1}^{N} -2(y_i - (wx_i + b)).$$

3. Take a step in the opposite direction to the gradient:
   - The learning rate α controls the size of an update.

$$w \leftarrow w - \alpha\frac{\partial l}{\partial w};$$

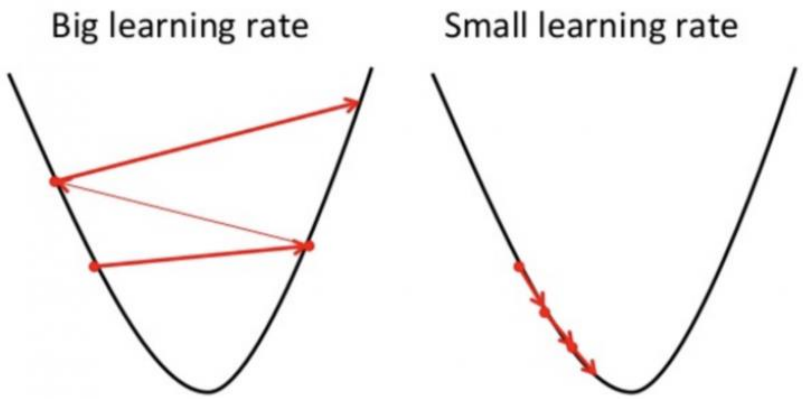
$$b \leftarrow b - \alpha\frac{\partial l}{\partial b}.$$

4. Repeat steps 2 and 3 until:
   - Local minima achieved (or approximated within a set tolerance)
   - Or max number of iterations reached.
     - Every repetition is called an **epoch**.

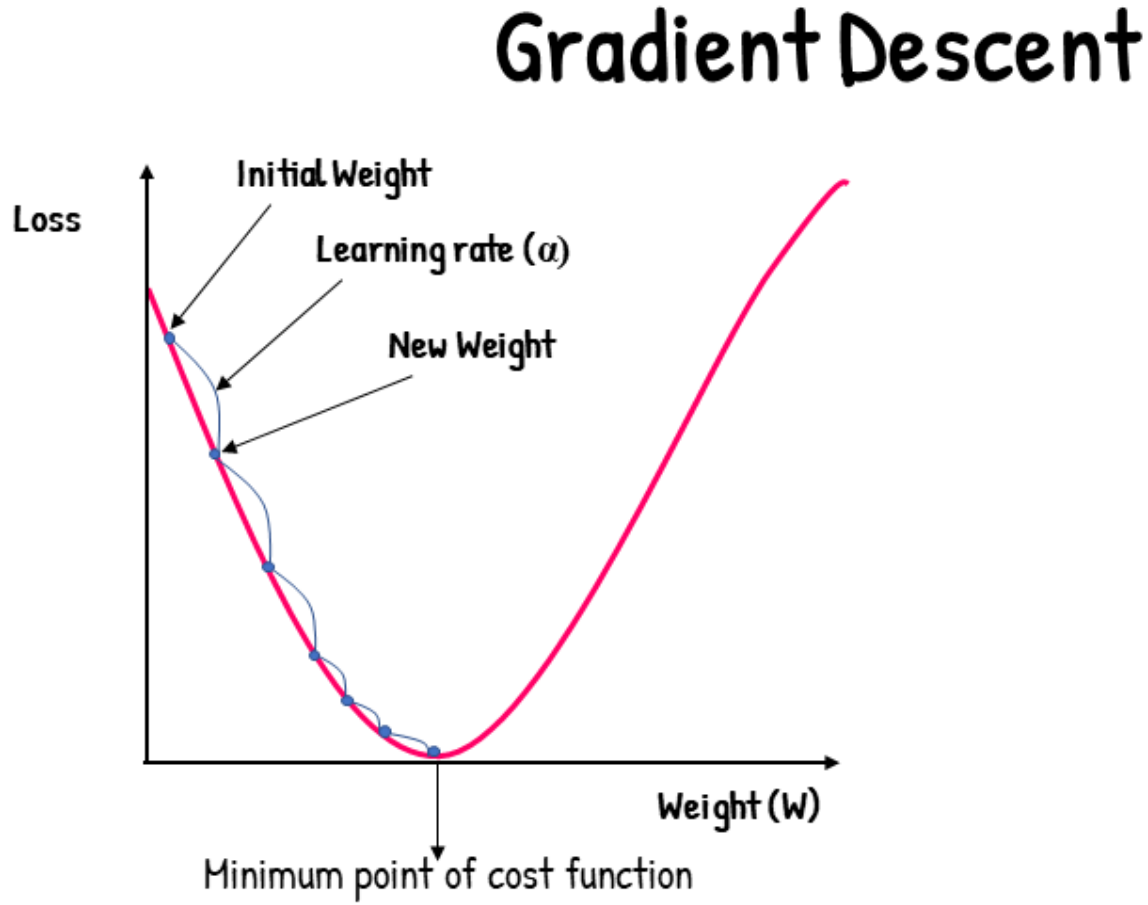
Learning Rate

- Learning Rate α
  - A parameter that controls how big the steps are in gradient descent.
  - Usually, a small value like 0.001.

$$w \leftarrow w - \alpha \frac{\partial l}{\partial w};$$
$$b \leftarrow b - \alpha \frac{\partial l}{\partial b}.$$

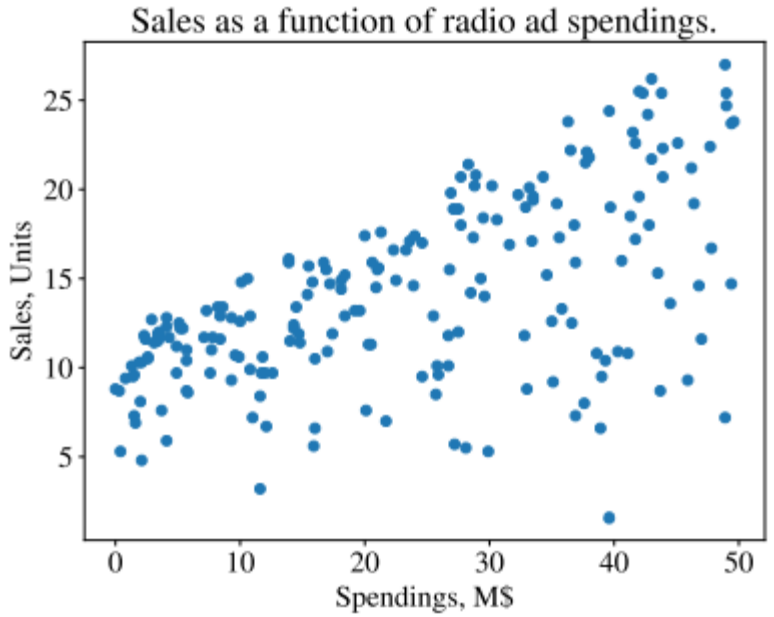


Gradient Descent



Example: Sales as a function of radio ad spendings

- Data
  - **Y** corresponds to the **sales** in units (the quantity we want to predict)
  - **X** corresponds to our feature: the **spendings** on radio ads in M$.
  - 200 Companies
- We want to build a regression model that we can use to predict units sold based on how much a company spends on radio advertising.



Python Example Code

```
df = pd.read_csv('Advertising.csv')
x = df['radio']
y = df['sales']

w,b = train(x,y,0.0,0.0,0.001,15000)
print("loss: "+ str(loss(x,y,w,b)))

x_new = 23.0
y_new = predict(x_new, w, b)
print(y_new)
```

Code Link: https://colab.research.google.com/drive/1S6odHBHsjvoODjKDLdP9iEGuNPoZKfXi?usp=sharing

Training (Fitting)

```
def train(featuer,target,w,b,alpha,epochs):
    for e in range(epochs):
        w, b = update_w_and_b(featuer,target,w,b,alpha)
    return w,b

def update_w_and_b(featuer,target,w,b,alpha):
    dl_dw = 0.0
    dl_db = 0.0
    N = len(featuer)
    for i in range(N):
        dl_dw += -2*featuer[i]*(target[i] - (w*featuer[i] + b))
        dl_db += -2*(target[i] - (w*featuer[i] + b))
    # update w and b
    w = w - (dl_dw/float(N))*alpha
    b = b - (dl_db/float(N))*alpha
    return w,b
```

$$\frac{\partial l}{\partial w} = \frac{1}{N}\sum_{i=1}^{N} -2x_i(y_i - (wx_i + b));$$
$$\frac{\partial l}{\partial b} = \frac{1}{N}\sum_{i=1}^{N} -2(y_i - (wx_i + b)).$$

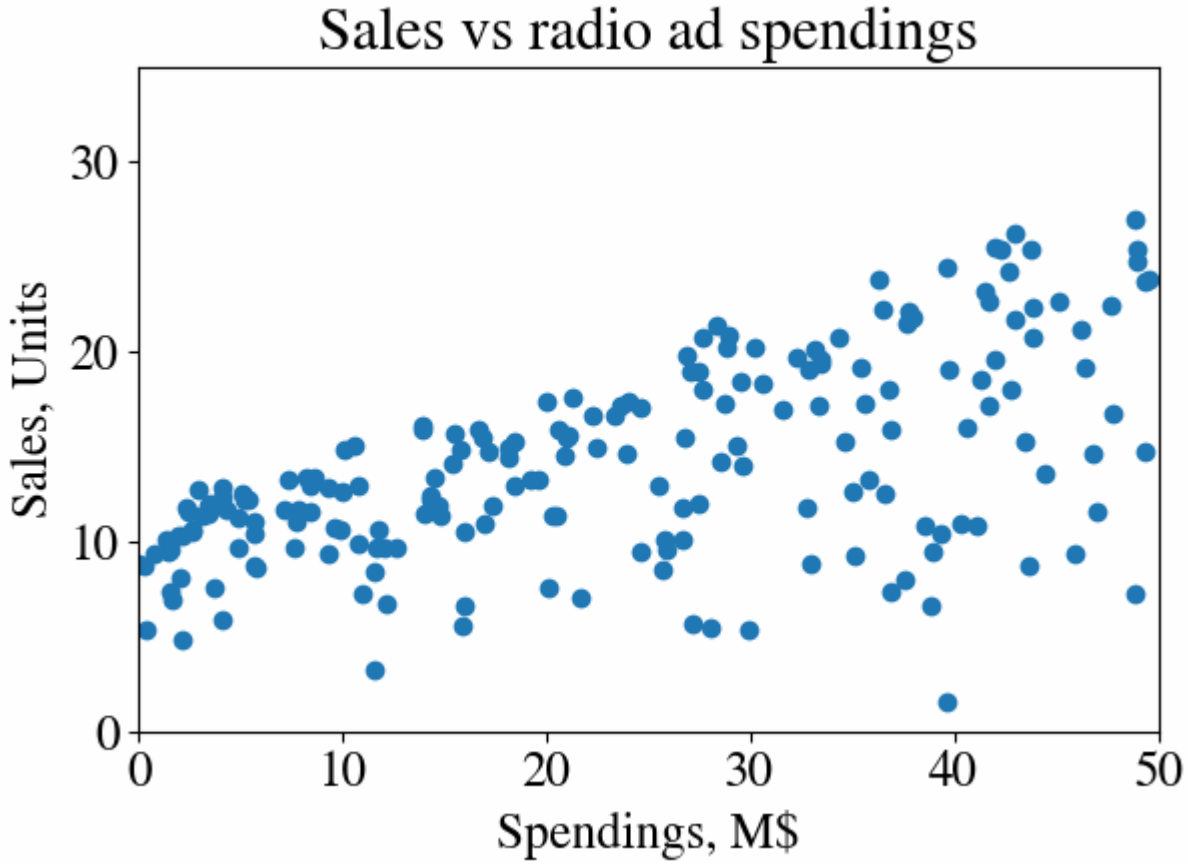
$$w \leftarrow w - \alpha\frac{\partial l}{\partial w};$$
$$b \leftarrow b - \alpha\frac{\partial l}{\partial b}.$$

Predicting a new value

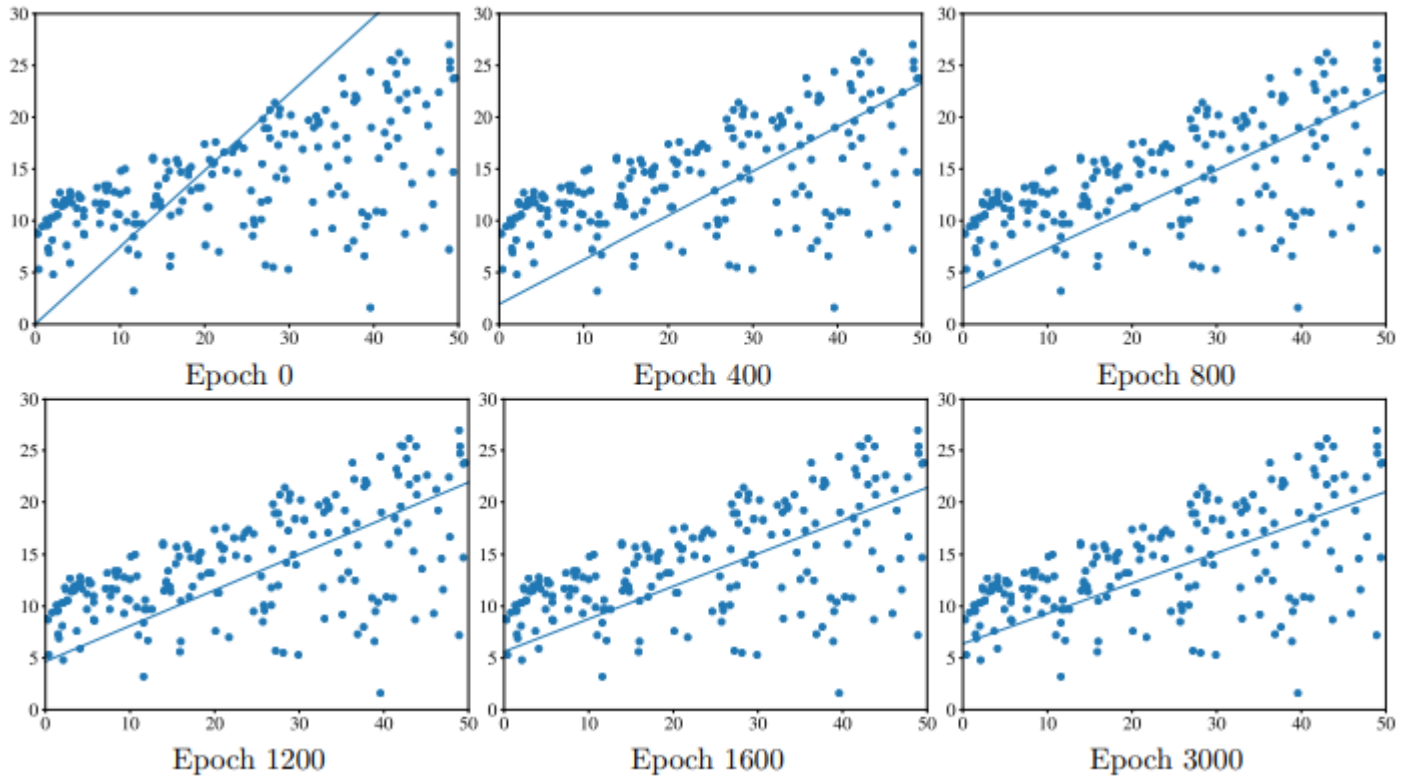
```
def loss(featuer,target,w,b):
    N = len(featuer)
    total_error = 0.0
    for i in range(N):
        total_error += (target[i] - (w*featuer[i] + b))**2
    return total_error/float(N)

def predict(x,w,b):
    return w*x + b
```

$$l \stackrel{\text{def}}{=} \frac{1}{N}\sum_{i=1}^{N}(y_i - (wx_i + b))^2.$$

Example Results (Animated)

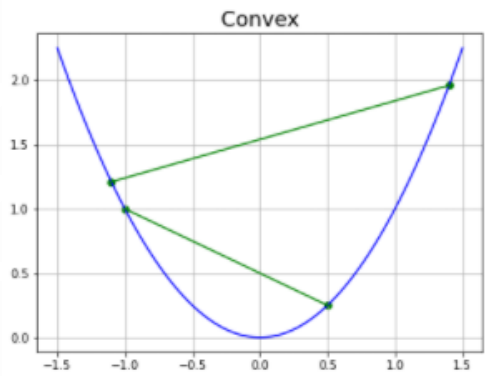


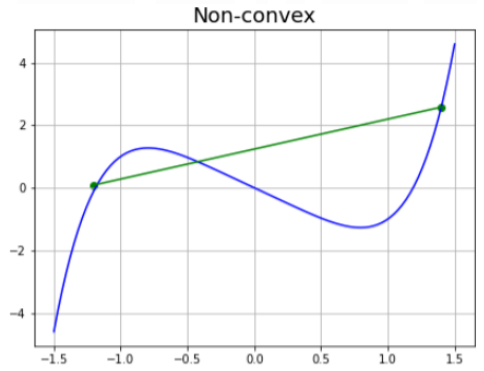
Example Results

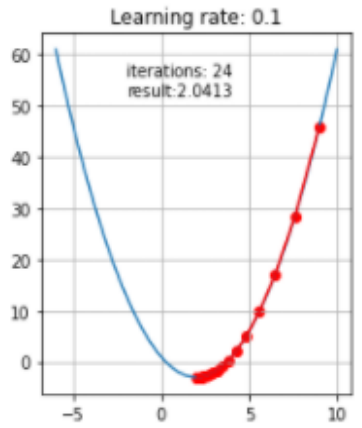


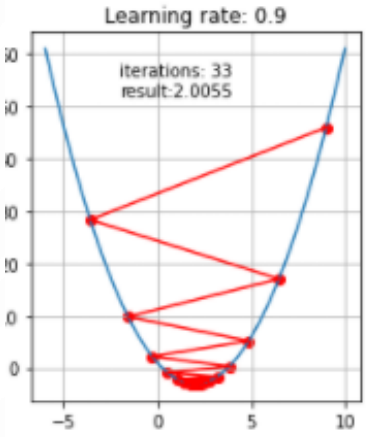
Problems with Gradient Descent

- Cost function may not be differentiable.
- Function may not be convex.
- Slow with larger datasets.
- Requires a lot of memory.



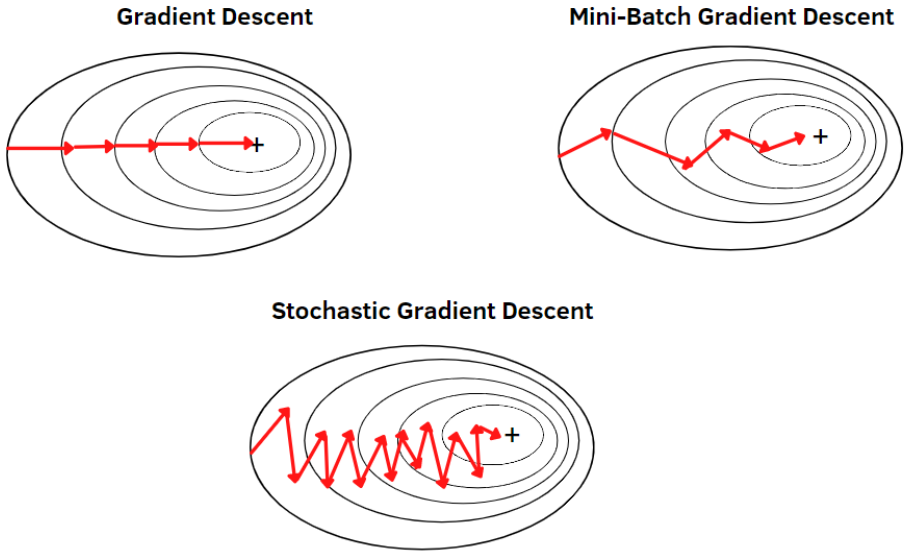




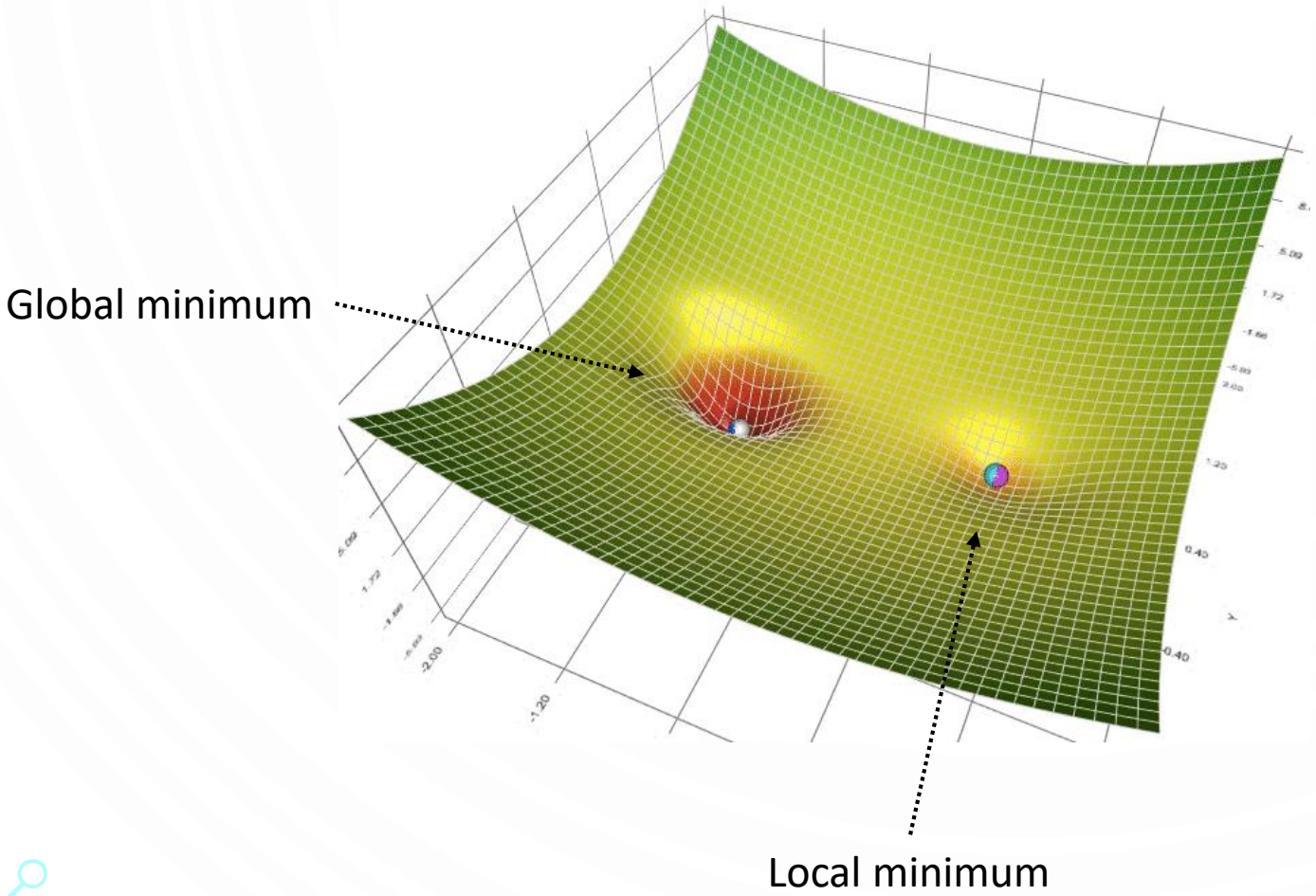


Gradient Descent Variants

- Minibatch SGD
  - Uses only random **subset** (mini-batch) of the training sample from training set to do the update for a parameter in a particular iteration
  - The most common type of gradient descent.
- Stochastic Gradient Descent (SGD)
  - Uses only **one** random training sample from training set to do the update for a parameter in a particular iteration
  - Faster updates.
  - SGD often converges much faster compared to GD but the error function is not as well minimized as in the case of GD.
- Other variants
  - Adagrad
  - Momentum
  - RMSprop
  - Adam



Animation of 5 gradient descent methods on a surface



- Gradient Descent (cyan)
- Momentum (magenta)
- AdaGrad (white)
- RMSProp (green)
- Adam (blue)

Optimizers Comparison

Optimizer	State Memory [bytes]	# of Tunable Parameters	Strengths	Weaknesses
SGD	0	1	Often best generalization (after extensive training)	Prone to saddle points or local minima Sensitive to initialization and choice of the learning rate α
SGD with Momentum	$4n$	2	Accelerates in directions of steady descent Overcomes weaknesses of simple SGD	Sensitive to initialization of the learning rate α and momentum β
AdaGrad	$\sim 4n$	1	Works well on data with sparse features Automatically decays learning rate	Generalizes worse, converges to sharp minima Gradients may vanish due to aggressive scaling
RMSprop	$\sim 4n$	3	Works well on data with sparse features Built in Momentum	Generalizes worse, converges to sharp minima
Adam	$\sim 8n$	3	Works well on data with sparse features Good default settings Automatically decays learning rate α	Generalizes worse, converges to sharp minima Requires a lot of memory for the state
AdamW	$\sim 8n$	3	Improves on Adam in terms of generalization Broader basin of optimal hyperparameters	Requires a lot of memory for the state
LARS	$\sim 4n$	3	Works well on large batches (up to 32k) Counteracts vanishing and exploding gradients Built in Momentum	Computing norm of gradient for each layer can be inefficient

Summary of popular optimizers highlighting their strengths and weaknesses. The column state memory denotes the number of bytes which is required by the optimizer — additional to the memory required for the gradient. Hereby, n is the number of parameters of the machine learning model. For example, SGD without momentum will only require the memory to store the gradient but SGD with momentum needs to store the moving average of the gradients as well.
Source: https://www.lightly.ai/post/which-optimizer-should-i-use-for-my-machine-learning-project

Note

- Notice that gradient descent and its variants are not machine learning algorithms.
- They are solvers of minimization problems in which the function to minimize has a gradient (in most
- points of its domain).

How Machine Learning Engineers Work

- Unless you are a research scientist or work for a huge corporation with a large R&D budget, you usually don't implement machine learning algorithms yourself.
  - You don't implement gradient descent or some other solver either.
- You use libraries, most of which are open source.
  - A library is a collection of algorithms and supporting tools implemented with stability and efficiency in mind.
- The most frequently used in practice open-source machine learning library is scikit-learn.

Machine Learning Algorithms Particularities

Hyperparameters

- Each algorithm has special hyperparameters that can differentiate it from other algorithms.

Categorical & Numerical acceptance

- Some algorithms, like decision tree learning, can accept both categorical and numerical features.
- Some accept only numerical data.
  - Examples, SVM, logistic and linear regression, as well as kNN (with cosine similarity or Euclidean distance metrics)

Classes Weights

- Some algorithms, like SVM, allows providing weightings for each class.
- Could be important if instances of some class are in the minority in your training data, but you would like to avoid misclassifying examples of that class as much as possible.

Confidence

- Some classification models, like SVM and kNN, given a feature vector only output the class.
- Others, like logistic regression, can also return the score between 0 and 1 which can be interpreted as either how confident the model is about the prediction or as the probability that the input example belongs to a certain class.

Machine Learning Algorithms Particularities

Incremental Learning

- Some classification algorithms (like decision tree learning, logistic regression, or SVM) build the model using the whole dataset at once.
  - If you have got additional labeled examples, you have to rebuild the model from scratch.
- Other algorithms can be trained iteratively, **one batch at a time**.
  - Once new training examples are available, you can **update** the model using only the new data.
  - Incremental estimators in scikit-learn:
    - **Classification**
      - sklearn.naive_bayes.MultinomialNB
      - sklearn.naive_bayes.BernoulliNB
      - sklearn.linear_model.Perceptron
      - sklearn.linear_model.SGDClassifier
      - sklearn.linear_model.PassiveAggressiveClassifier
    - **Regression**
      - sklearn.linear_model.SGDRegressor
      - sklearn.linear_model.PassiveAggressiveRegressor