ECE 469/ECE 568 Machine Learning

Textbook:
Machine Learning: a Probabilistic Perspective by Kevin Patrick Murphy

Southern Illinois University

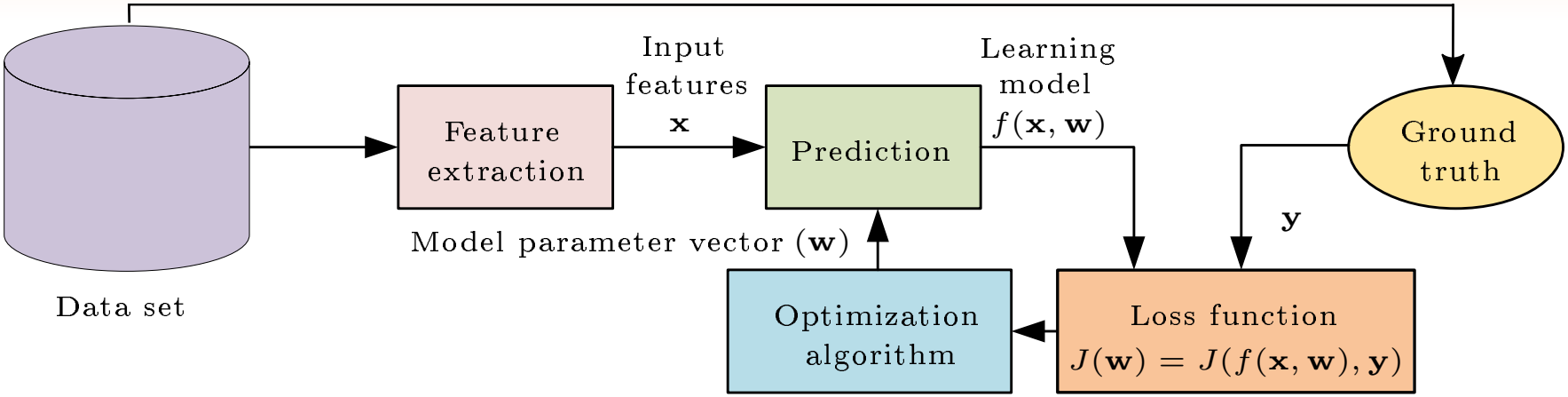
September 8, 2024

A recap for the last lecture

- Data preprocessing for machine learning:
  - **Data cleaning** : handling missing features and correcting outliers.
  - **Feature scaling** : max-min normalization and statistical standardization.
  - **Feature encoding** : one-hot encoding, ordinal encoding, target-mean encoding, and frequency encoding.
- Practical considerations of gradient descent algorithm
  - The role of data preprocessing in gradient descent algorithm
  - Learning rate on the convergence of gradient descent algorithm
- Gradient descent algorithm for large data sets
  - Issues of batch gradient descent algorithm for large data sets

Supervised machine learning architecture

- The main processes in supervised machine learning is depicted below.



- In this lecture, we are going to discuss optimizers for machine learning.
- Recall the linear regression learning model: $f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x}$, where $\mathbf{w}$ is model parameter/weight vector and $\mathbf{x}$ is input feature vectors.
- Recall that in supervised learning, the loss function is parameterized by the input feature vectors, model parameters, and output vector.
- For instance, mean square error (MSE) loss function is defined as

$$J(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2,$$

where $\mathbf{x}_i$ is the ith feature vector and y_i is the corresponding output.

Gradient descent algorithm - Summary

- In gradient descent algorithm, the model parameter vector is updated in each iteration as follows:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \nabla J(\mathbf{x}, \mathbf{w}^{(t)}),$$

where the terms are defined as

$\mathbf{w}^{(t+1)}$	:	updated model parameter vector
$\mathbf{w}^{(t)}$	:	current model parameter vector
α	:	learning rate
$J(\mathbf{w})$	:	loss function as a function of model parameter vector
$\nabla J(\mathbf{w})$	:	gradient of loss function wrt to model parameter vector
$\mathbf{x}$	:	input feature vector

- In batch gradient descent algorithm, the exact gradient of the loss function is computed in each iteration as

$$\nabla J(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \nabla J(\mathbf{x}_i, \mathbf{w}),$$

where N is the size of training data-set.

Batch gradient descent algorithm - Summary

- This gradient descent algorithm looks at every example in the entire training set on every step, and is called batch gradient descent.

Batch gradient descent algorithm

Repeat until concergence{

$$\text{temp}_j := w_j - \frac{\alpha}{N} \sum_{i=1}^{N} \frac{\partial J(\mathbf{x}^{(i)}, \mathbf{w})}{\partial w_j}$$

continue for all $j = 0, 1, \cdots, n$

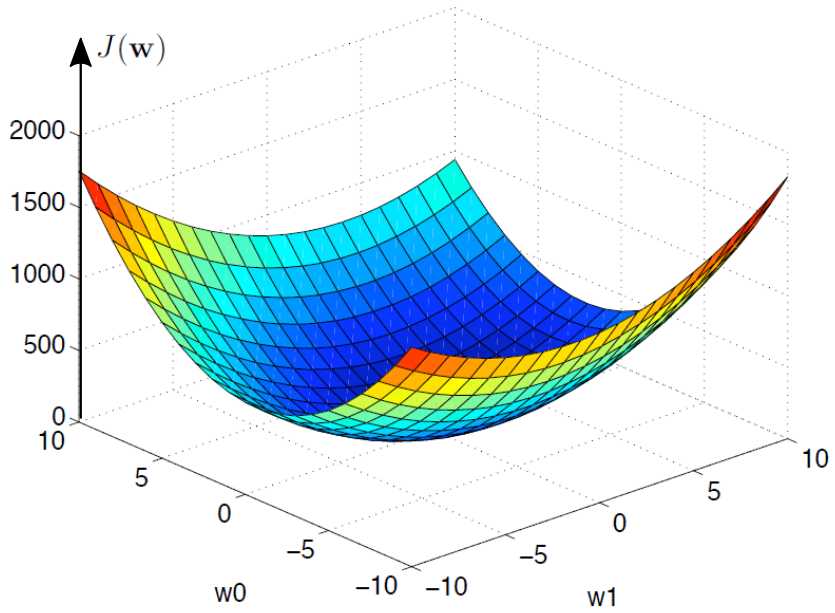
$w_j := \text{temp}_j \quad \forall j$

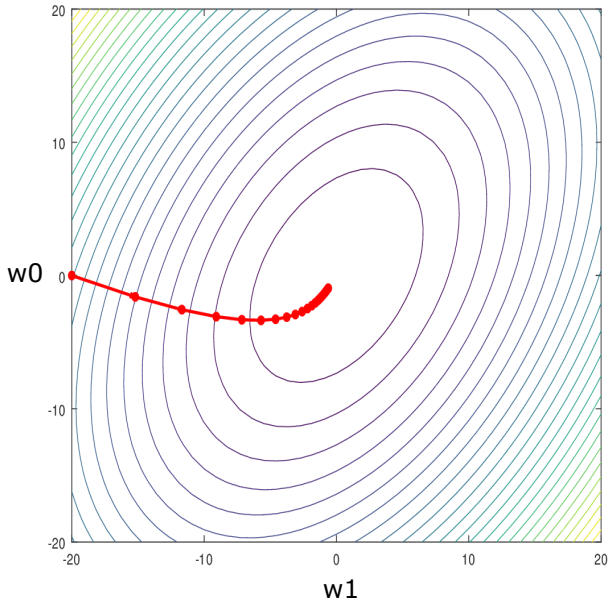
}

- Here, $\mathbf{w} = [w_0, w_1, \cdots, w_n]$ is the model parameter/weight vector, $\mathbf{x}_i$ is the ith feature vector, and $J(\mathbf{x}_i, \mathbf{w})$ is the loss function.
- Thus, the batch gradient descent algorithm has to scan through the entire training set before taking a single step.
- This is a costly operation if the number of examples in the training data set (N) is very large.

Batch gradient descent algorithm

- Batch gradient descent algorithm:
  - computes exact gradients of the loss function
  - attains global optimum without many oscillations
  - computationally complicated
  - runs slowly





Stochastic gradient descent algorithm for large data sets

- There exists an alternative to batch gradient descent algorithm that works very well with much larger data sets.
- This modified algorithm is termed as the stochastic gradient descent algorithm.
- Here, an approximation to the gradient of the loss function is used:

$$\nabla J(\mathbf{w}) \approx \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla J(\mathbf{x}_i, \mathbf{w}),$$

  where $\mathcal{B} \subseteq \{1, \cdots, N\}$ is a random subset of the data and $|\mathcal{B}|$ the cardinality/size of the set $\mathcal{B}$.
- In machine learning terminology, we term $|\mathcal{B}|$ as the batch size.

Stochastic gradient descent algorithm for large data sets

- In classical stochastic gradient descent algorithm, we repeatedly run through every entry in training set, and hence, we set $|\mathcal{B}| = 1$.
- Each time we encounter a training example, we update the parameters according to the gradient of the loss function with respect to that single training example only.

Stochastic gradient descent algorithm

Repeat until concergence $\{$

for $i = 1$ to N $\{$

$$w_j := w_j - \alpha \frac{\partial J(\mathbf{x}_i, \mathbf{w})}{\partial w_j} \quad \forall j$$

$\}$

$\}$

Stochastic gradient descent algorithm

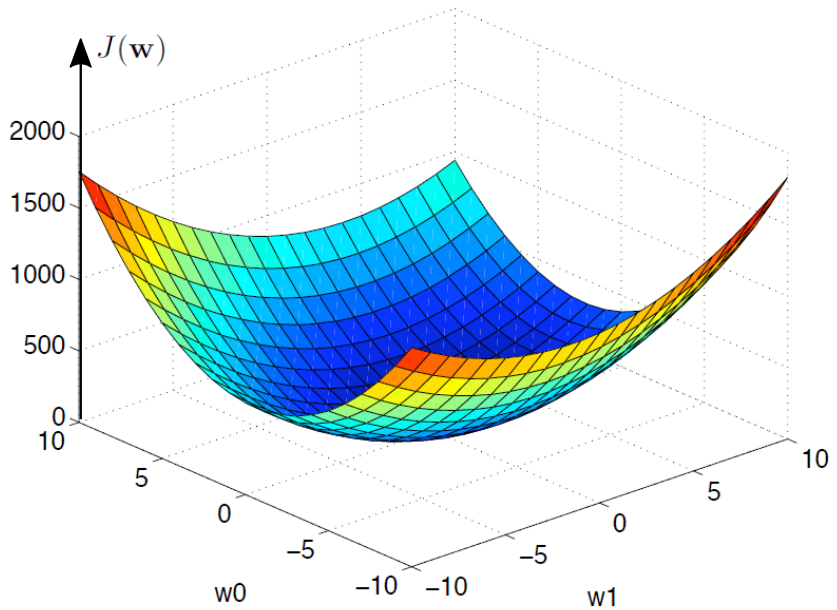
- Note that stochastic gradient descent algorithm selects a single instance randomly in the training set at every step and computes the gradients based only on that single instance.
- Working on a single instance at a time makes the algorithm much faster because it has very little data to manipulate at every iteration.
- It also makes stochastic gradient descent possible to train on huge training sets, since only one instance needs to be in memory at each iteration.
- Hence, the stochastic gradient descent algorithm can start making progress right away, and continues to make progress with each example it looks at.

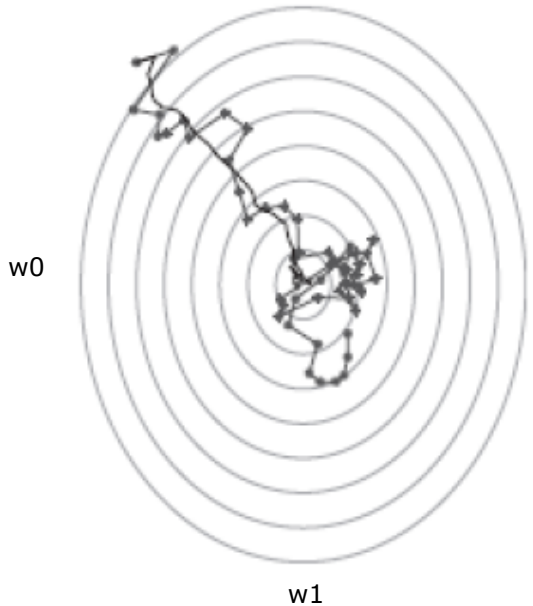
Stochastic gradient descent algorithm

- Stochastic gradient descent algorithm:
  - computes approximate gradients of the loss function
  - may converge in the vicinity of global optimum with many oscillations
  - computationally simpler than batch gradient descent algorithm
  - runs faster than batch gradient descent algorithm
  - provides a better trade-off between accuracy and efficiency
- When the training set is very large, stochastic gradient descent algorithm is often preferred over batch gradient descent algorithm.

Stochastic gradient descent algorithm

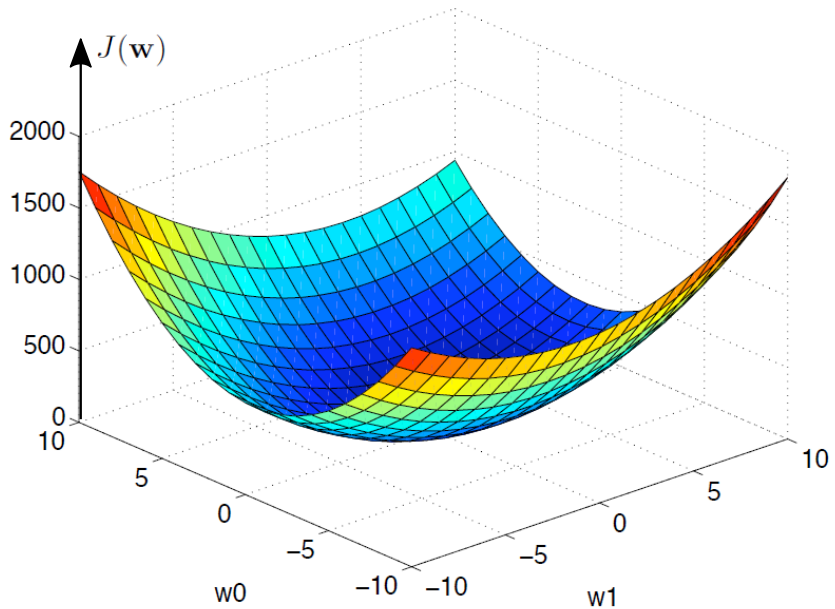
- Stochastic gradient descent algorithm attains a weight vector $\mathbf{w}$ close to the optimum much faster than batch gradient descent algorithm.
- Note that it may never converge to the minimum, and the parameters $\mathbf{w}$ will keep oscillating around the minimum of the loss function $J(\mathbf{w})$.
- However, in practice, most of the values near the minimum will be reasonably good approximations to the true minimum.

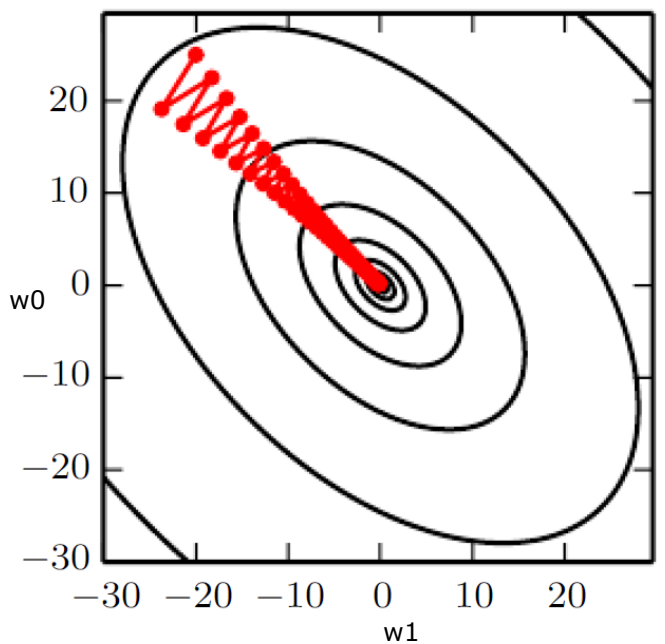




Stochastic gradient descent algorithm

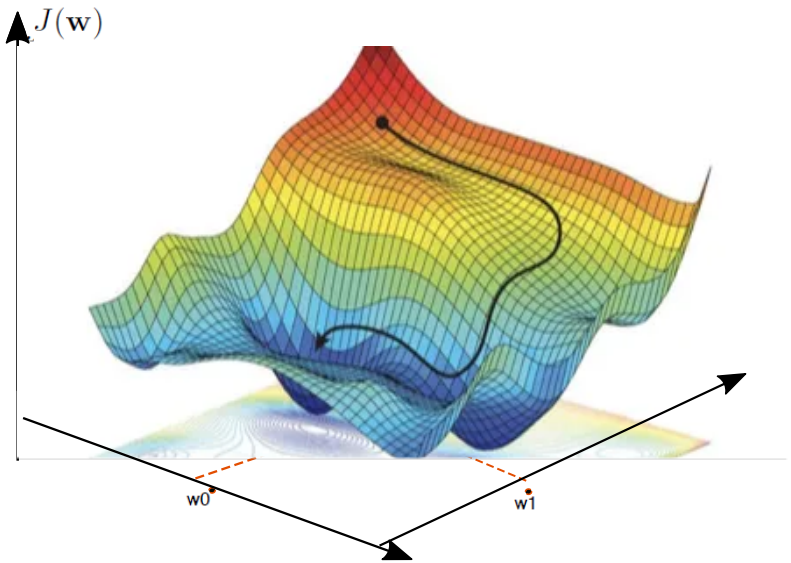
- Stochastic gradient descent algorithm can be run with a fixed learning rate α: –> may not always converge to the global minimum of loss function.
- Nevertheless, by letting learning rate α decrease to the vicinity of zero as the algorithm runs, the oscillation near the optima can be mitigated.
- Hence it is also possible to ensure that the parameters will converge to the global minimum rather than merely oscillate around the minimum.





Stochastic gradient descent algorithm

- The oscillations of stochastic gradient descent algorithm may sometimes be useful when the loss function is non-convex.
- If the loss function $J(\mathbf{x}, \mathbf{w})$ is non-convex, this oscillation around a local minima can actually help the algorithm jump out of that local minima.
- Thus, stochastic gradient descent algorithm has a better chance of finding the global minimum than batch gradient descent algorithm does for non-convex functions.



Mimi-batch gradient descent algorithm

- There exists an algorithm that provides a better trade-off between accuracy and efficiency than that of batch and stochastic versions of gradient descent algorithms.
- Mini-batch gradient descent algorithm computes the gradients on small random sets of instances called mini-batches, instead of computing the gradients based on the full training set (as in batch gradient descent) or based on just one instance (as in stochastic gradient descent).
- Thus, mini-batch gradient descent algorithm lies in between the batch gradient descent and stochastic gradient descent algorithms.
  - Batch gradient descent: Batch size = Size of training set
  - Stochastic gradient descent: Batch Size = 1
  - Mini-batch gradient descent: 1 < Batch size < Size of training set

Mimi-batch gradient descent algorithm - An example

- Let us define the mini-batch size to be $b = 10$ for instance and the total number of rows in the training date set to be $m = 1000$.

Mini-batch gradient descent

Repeat until concergence $\{$

for $i = \{1, 11, \cdots, 991\}$ $\{$

$$w_j := w_j - \frac{\alpha}{10} \sum_{k=i}^{i+9} \frac{\partial J(\mathbf{x}_k, \mathbf{w})}{\partial w_j} \quad \forall j$$

$\}$

$\}$

- The main benefit of mini-batch gradient descent algorithm over stochastic gradient descent algorithm is that we can get a performance boost from hardware optimization acceleration of operations by using parallel processing through GPUs.

Other variants of gradient descent algorithm

- The method of momentum is designed to accelerate learning.
- Momentum algorithm accumulates an decaying moving average of past gradients and continues to move in their direction.
- Gradient descent with momentum updates the model parameters as

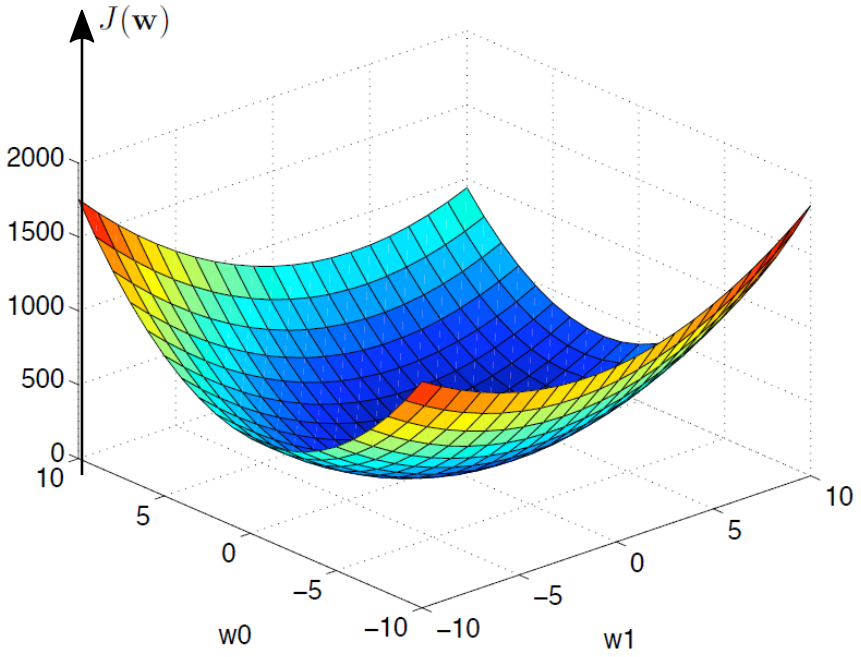
$$\begin{aligned} \text{Update velocity vector:} \quad & \mathbf{v} \leftarrow \beta\mathbf{v} - \alpha\nabla J(\mathbf{w}) \\ \text{Update model parameter vector:} \quad & \mathbf{w} \leftarrow \mathbf{w} + \mathbf{v}, \end{aligned}$$

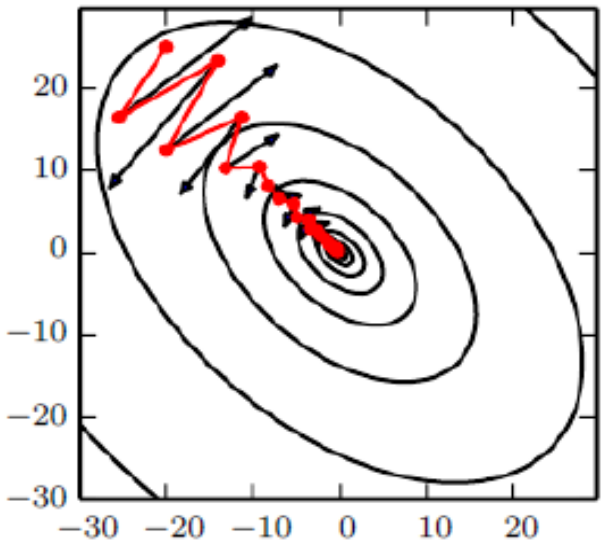
where the terms are defined as

$$\begin{aligned} \mathbf{w} &: \text{model parameter vector} \\ \alpha &: \text{global learning rate} \\ \beta \in [0,1) &: \text{hyperparameter} \\ J(\mathbf{w}) &: \text{loss function as a function of model parameter vector} \\ \nabla J(\mathbf{w}) &: \text{gradient of loss function wrt to model parameter vector} \\ \mathbf{v} &: \text{velocity vector} \end{aligned}$$

- The hyperparameter $\beta \in [0,1)$ determines how quickly the contributions of previous gradients decay.

Other variants of gradient descent algorithm





- Recall that the classical stochastic gradient descent exhibits oscillation.
- But, gradient descent with momentum tends to keep traveling in the same steepest direction, preventing many oscillations.
- This momentum method can be applied to both gradient descent and stochastic gradient descent algorithms.

Other variants of gradient descent algorithm

- There is a class of gradient descent algorithm that leverages adaptive learning rate optimization.
  - AdaGrad optimizer - Adaptive Gradient algorithm:
    - A modified stochastic gradient descent algorithm with an adaptive learning rate.
    - Leverages cumulative summation of past squared gradients.
    - Adapts learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all of their historical squared values.
    - The model parameters are updated as follows:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \mathbf{G}_t^{-1/2} \nabla J\left(\mathbf{w}^{(t)}\right),$$

where $\mathbf{G}_t$ is the sum of outer product of all previous gradients defined by

$$\mathbf{G}_t = \sum_{\tau=1}^{t} \left[\nabla J\left(\mathbf{w}^{(\tau)}\right) \nabla J^T\left(\mathbf{w}^{(\tau)}\right)\right].$$

    - Note that the effective learning rate is now adapted by considering past subgradients: $\alpha_{\text{eff}} = \alpha \mathbf{G}_t^{-1/2}$.

Other variants of gradient descent algorithm

- Note that AdaGrad shrinks learning rate according to entire history of squared gradient.
- This may cause the learning rate to become too small before arriving at such a convex structure.
- RMSProp optimizer - Root Mean Square Propagation:
  - An adaptive optimization algorithm that is an improved version of AdaGrad.
  - Performs better in non-convex setting by changing gradient accumulation into a weighted moving average.

$$\mathbf{G}_{t+1} = \beta \mathbf{G}_t + (1-\beta)\left[\nabla J\left(\mathbf{w}^{(t)}\right) \nabla J^T\left(\mathbf{w}^{(t)}\right)\right],$$

where β is the decay rate.

Other variants of gradient descent algorithm

- Adam optimizer - Adaptive Moment:
  - A modification to RMSProp combining it with adaptive moments.
  - Adam optimizer updates the model parameters as

$$\begin{aligned} \text{Update first moment} &: \mathbf{s} \leftarrow \beta_1 \mathbf{s} + (1-\beta_1)\mathbf{g} \\ \text{Update second moment} &: \mathbf{r} \leftarrow \beta_2 \mathbf{r} + (1-\beta_2)\mathbf{g} \odot \mathbf{g} \\ \text{Update parameter vector} &: \mathbf{w} \leftarrow \mathbf{w} - \alpha \Delta \mathbf{w} \end{aligned}$$

where the terms are defined as

$$\Delta \mathbf{w} = \frac{\mathbf{s}/(1-\beta_1)}{\sqrt{\mathbf{r}/(1-\beta_2)}} \quad \text{(operations applied element-wise)}$$

$$\begin{aligned} \mathbf{w} &: \text{model parameter vector} \\ \alpha &: \text{global learning rate} \\ \beta_1 \in [0,1) &: \text{hyperparameter} \\ \beta_2 \in [0,1) &: \text{hyperparameter} \\ J(\mathbf{w}) &: \text{loss function} \\ \mathbf{g} = \nabla J(\mathbf{w}) &: \text{gradient of loss function wrt } \mathbf{w} \end{aligned}$$

Implementation of an optimizer in ML frameworks

1. **Scikit – Learn** :

   Stochastic gradient descent can be implemented using 'SGDRegressor'.

```
sklearn.linear_model.SGDRegressor(loss='squared_loss',
  alpha=0.001 )
```

   The details of arguments to 'SGDRegressor' method can be found here.

Implementation of an optimizer in ML frameworks

2. **Keras** :
   - All the variants of gradient descent is available in Keras.
   - More details on the optimizers available in Keras can be found here
   - These optimizers must be specified in 'model.compile()'.

   Keras : **compile** method

   ```
   Model.compile(optimizer="rmsprop", loss=..., ...)
   ```

   More details on the arguments to the methods can be found here.

- Mini-Batch Gradient Descent can be implemented in Keras by specifying the batch_size parameter in the model.compile() method.

Implementation of an optimizer in ML frameworks

3. **PyTorch** :
   - In pytorch 'torch.optm' is a package implementing various optimization algorithms.
   - To construct an Optimizer, you have to give it an iterable containing the parameters (all should be Variables) to optimize.
   - Then, you can specify optimizer-specific options such as the learning rate, weight decay, etc.

   Example:

```
optimizer = optim.SGD(model.parameters(), lr=0.01,
momentum=0.9)
optimizer = optim.Adam([var1, var2], lr=0.0001)
```

   The basics of PyTorch implementations can be found here.

Implementation of optimizing algorithms in different ML frameworks

3. **PyTorch** :
   - PyTorch provides a package 'DataLoader' for easy implementation of Mini-Batch Gradient Descent.
   - DataLoader handles data loading and preprocessing, streamlining the training process.
   - DataLoader in PyTorch is a powerful utility that automates the process of dividing the data set into batches.
   - It ensures that each mini-batch is correctly fed into the model during the training phase, optimizing the learning process.

```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(training_data, batch_size=64,
shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=64,
shuffle=True)
```