Introduction to Machine Learning in Geosciences GEO371T/398D.1

Optimization

Mrinal K. Sen

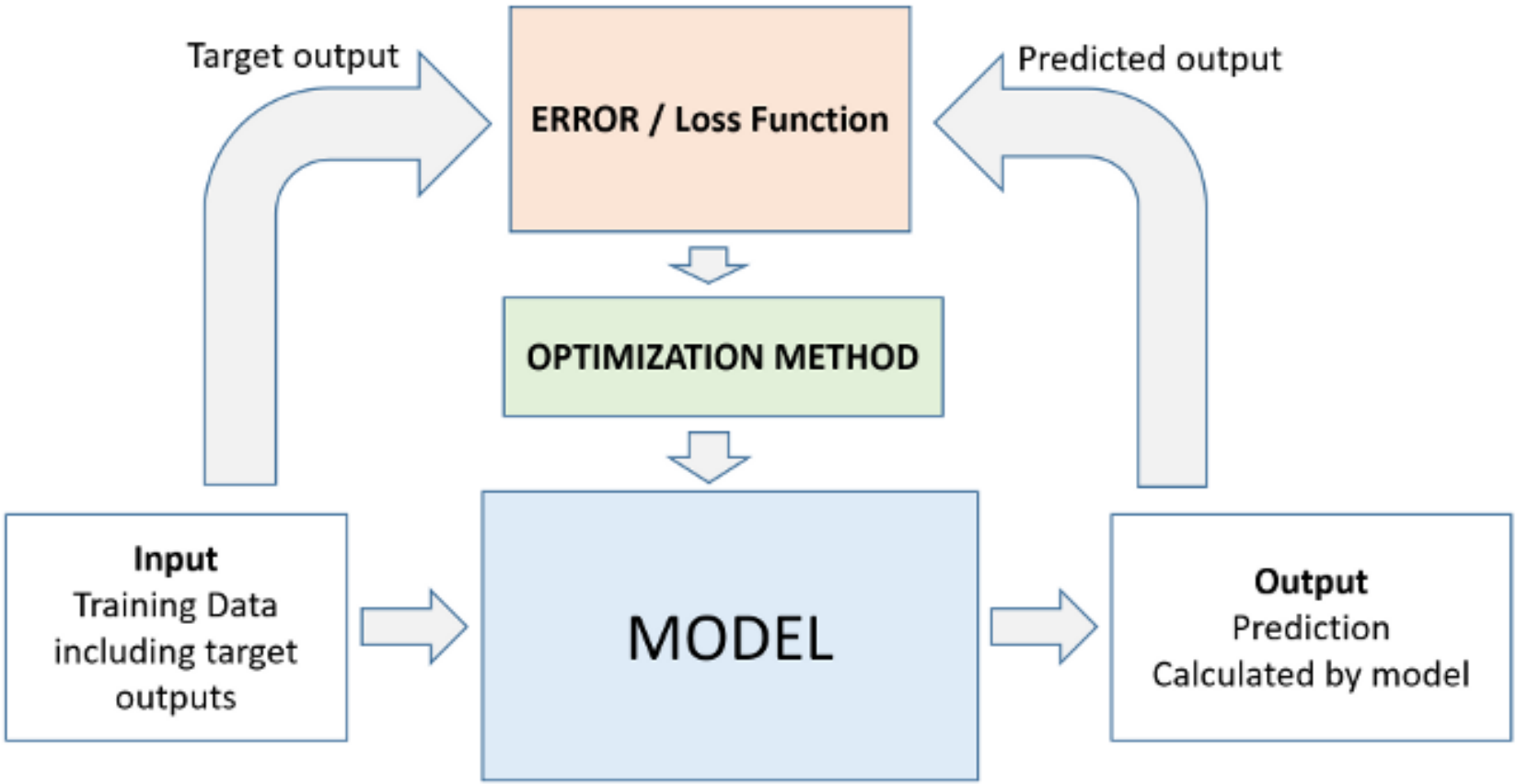
September 12, 2023

Content

- What is Optimization?
- Norms
- The Least Squares
- Regularization
- Optimization Methods
- Local Optimization: SD, CG, Newton, SGD, ADAM, Minibatch

Training a Machine Learning Model

- Features $\mathbf{x}$, Target $\mathbf{y}$
- Model $\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W})$
- Loss function $L(\mathbf{y}, \hat{\mathbf{y}})$
- Optimization: Minimize L

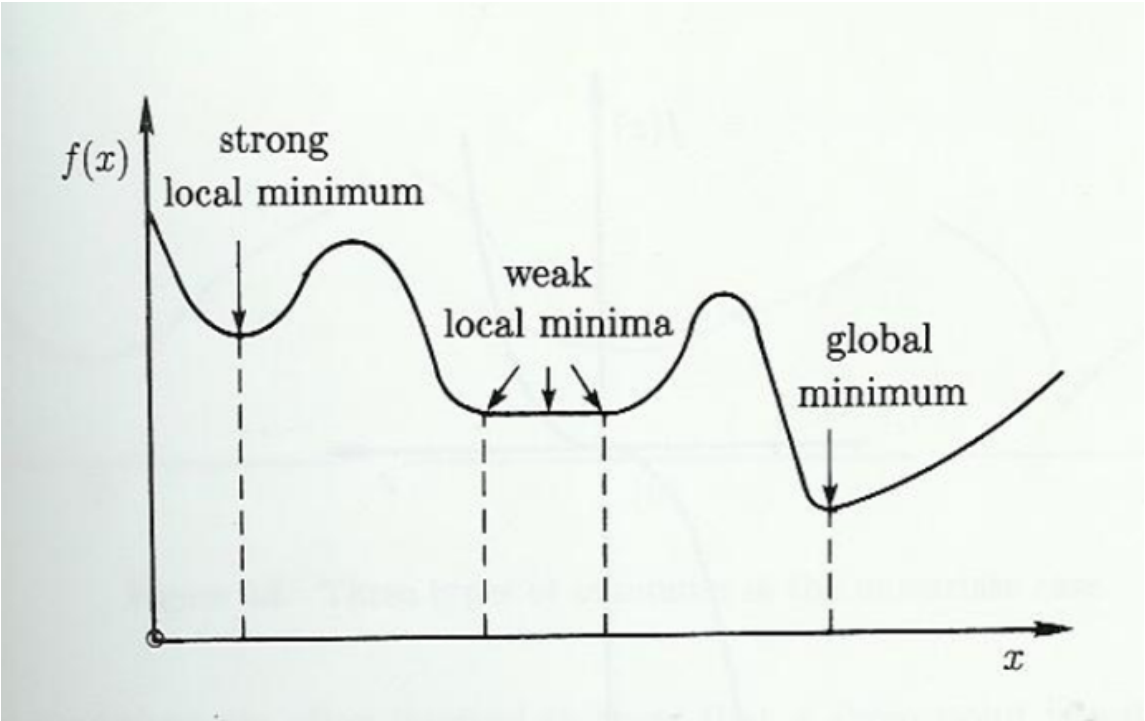


Optimization

- Non-linearly constrained optimization (NCP):

$$\text{minimize} \;\; L(\mathbf{w}); \;\; \mathbf{w} \in \Re^n$$

$$\text{subject to } c_i(\mathbf{W}) = 0; i = 1, 2, .., n_1; \;\; c_i(\mathbf{w}) \geq 0, i = n_1 + 1, n_1 + 2, ...$$



Optimization

- Any point that satisfies all the constraints of NCP is said to be **feasible**.
- the set of all feasible points is termed the **FEASIBLE REGION**.
- Let $\mathbf{w}^*$ be a feasible point and $N(\mathbf{w}^*, \delta)$ a set of feasible points contained in the δ- neighborhood of $\mathbf{w}^*$.

Definition A: The point $\mathbf{w}^*$ is a **strong local minimum** of NCP if there exists such that

- A1. $L(\mathbf{w})$ is defined on $N(\mathbf{w}^*, \delta)$; and
- A2. $L(\mathbf{w}^*) < L(\mathbf{y})$; $\forall \mathbf{y} \in N(\mathbf{w}^*, \delta), \mathbf{y} \neq \mathbf{w}^*$.

Definition B: The point $\mathbf{w}^*$ is a **weak local minimum** of NCP if there exists such that

- A1. $L(\mathbf{w})$ is defined on $N(\mathbf{w}^*, \delta)$; and
- A2. $L(\mathbf{w}^*)L(\mathbf{y})$; $\forall \mathbf{y} \in N(\mathbf{w}^*, \delta), \mathbf{y} \neq \mathbf{w}^*$.

Optimization

Univariate Case

- If $L(w)$ is twice continuously differentiable, and a local minimum of L exists at a finite point w^* , the following are the **necessary conditions**:
  - $L'(w^*) = 0$; and $L''(w^*) \geq 0$.
- **Sufficient conditions** are:
  - $L'(w^*) = 0$; and $L''(w^*) > 0$.

Multi-variate Case

- If $L(\mathbf{w})$ is twice continuously differentiable, and a local minimum of L exists at a finite point $\mathbf{w}^*$, the following are the **necessary conditions**:
  - $||L'(\mathbf{w}^*) = 0||$; and $L''(\mathbf{w}^*) = \mathbf{H}(\mathbf{w}^*)$ is positive semi-definite.
- **Sufficient conditions** are:
  - $||L'(\mathbf{w}^*) = 0||$; and $L''(\mathbf{w}^*) = \mathbf{H}(\mathbf{w}^*)$ is positive definite.

Linear Algebra

Topics

- Scalar, Vector, Matrix and Tensor
- Linear system of equations
- Matrix Algebra (transpose, adjoint, multiiplication, addition etc)
- Vector space
- Eigen Values and Eigen Vectors

Vector Norms

Measurement of fitness

Definition: A vector norm is a function from $\mathbb{R}^n$ to $\mathbb{R}$, with a certain number of properties. If $\mathbf{x} \in \mathbb{R}^n$, we denote its norm by $||\mathbf{x}||$, with the following properties:

- $||\mathbf{x}|| \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$, and $||\mathbf{x}|| = 0$ iff $\mathbf{x} = 0$.
- $||\alpha\mathbf{x}|| = |\alpha|||\mathbf{x}|| \quad \forall \alpha \in \mathbb{R}, \quad \mathbf{x} \in \mathbb{R}^n$
- $||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}|| \; \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$,

- The L_1-norm (Manhattan Distance):

$$||\mathbf{x}||_1 = \sum_{i=1}^{n} |x_i|.$$

- The L_2-norm (Euclidean Norm):

$$||\mathbf{x}||_2 = \sqrt{\sum_{i=1}^{n} x_i^2}.$$

- The L_∞-norm:

$$||\mathbf{x}||_\infty = \max |x_i|$$

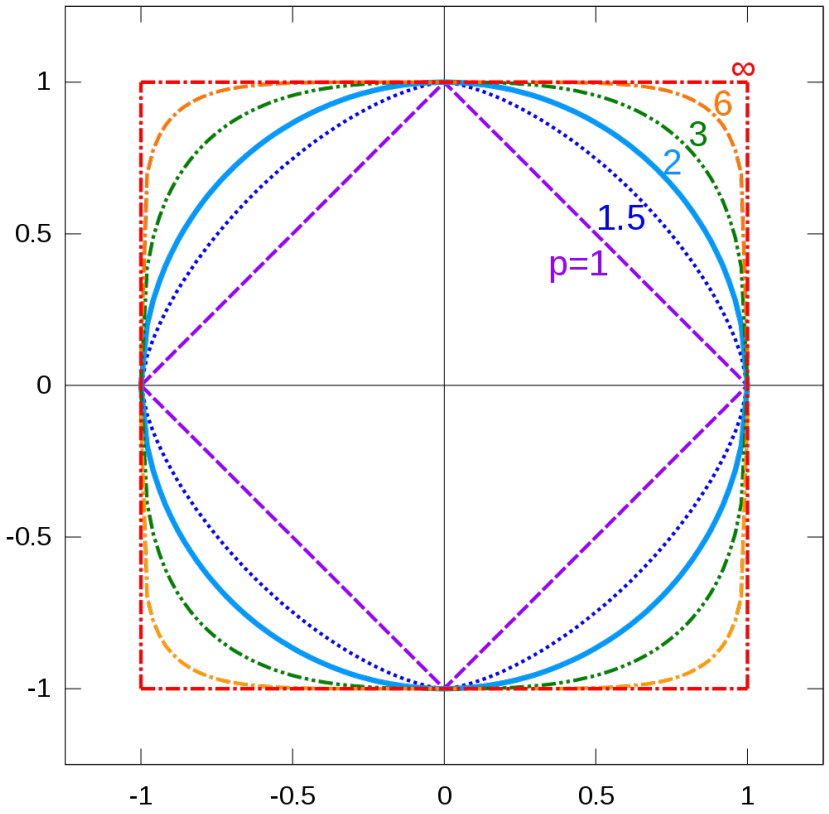
- The L_p-norm:

$$||\mathbf{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}, \quad \text{for} p \geq 1.$$

Vector Norms

Measurement of fitness

- $||\mathbf{x}||_2 \leq ||\mathbf{x}_1||$.
- $||\mathbf{x}||_{p+a} \leq ||\mathbf{x}||_p;\ p \geq 1,\ a \geq 0$.
- $0 < p < 1$ **does not define a norm.**
- when $p = 0$; the l_o "norm" : $|x_1|^0 + |x_2|^0 + \ldots |x_n|^0$; (define $0^0 = 0$) gives the **number of non-zero entries of the vector x**



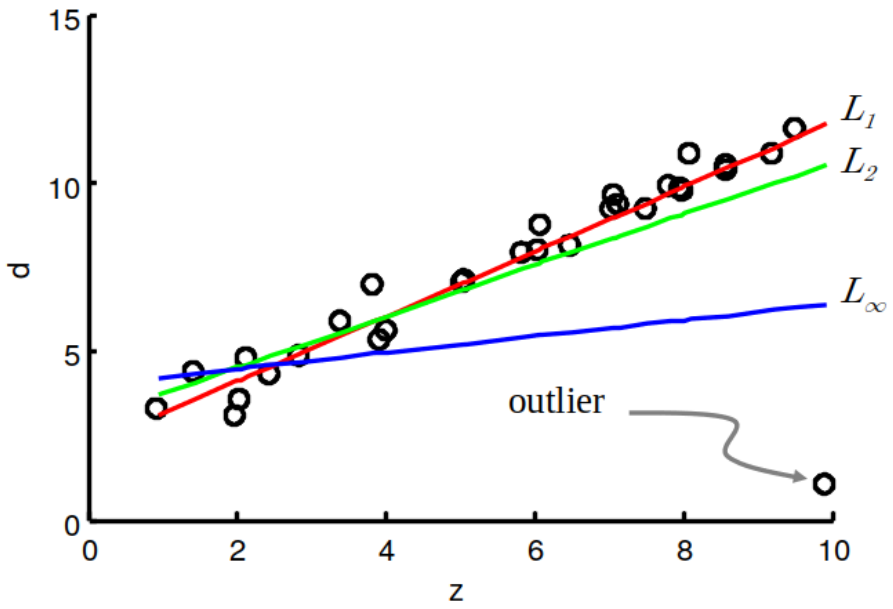
from Wikipedia

Vector Norms

Loss/Cost/Objective/Misfit

We need a measure of distance between two vectors $\mathbf{y}$, and $\hat{\mathbf{y}}$ ($\mathbf{y}, \hat{\mathbf{y}} \in \mathbb{R}^n$) in many ML applications, for example. cluster analysis, regression, cost or loss function in NN.

- **Euclidean Distance**: $L_2 norm$: $||\mathbf{y} - \hat{\mathbf{y}}||_2 = (\mathbf{y} - \hat{\mathbf{y}})^T(\mathbf{y} - \hat{\mathbf{y}})$.
- **Manhattan Distance**: $L_1 norm$: $||\mathbf{y} - \hat{\mathbf{y}}||_1$.
- **Minkowski Distance**: $L_p norm$: $||\mathbf{y} - \hat{\mathbf{y}}||_p$.
- **Mahalnabis Distance**: $D^2 = (\mathbf{y} - \hat{\mathbf{y}})^T \mathbf{C}^{-1}(\mathbf{y} - \hat{\mathbf{y}})$., $\mathbf{C}$: Covariance Matrix.

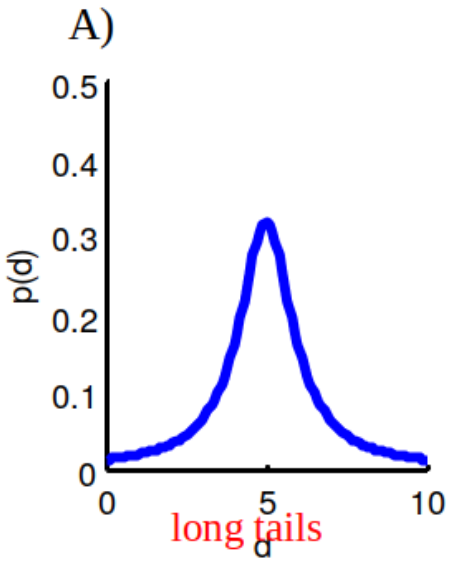


from Menke

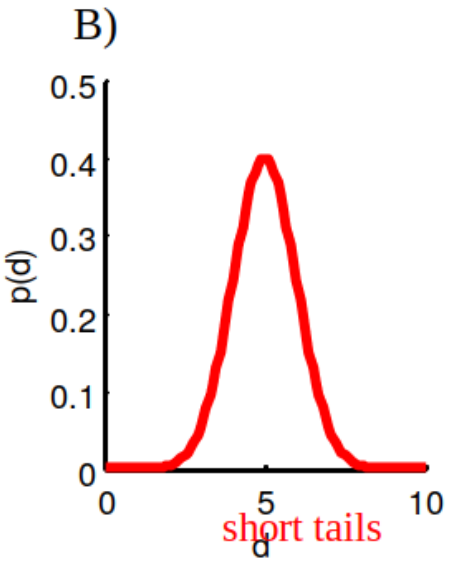
Vector Norms

Loss/Cost/Objective/Misfit

Answer is related to the distribution of the error. Are outliers common or rare?



long tails
outliers common
outliers unimportant
use low norm
gives low weight to outliers



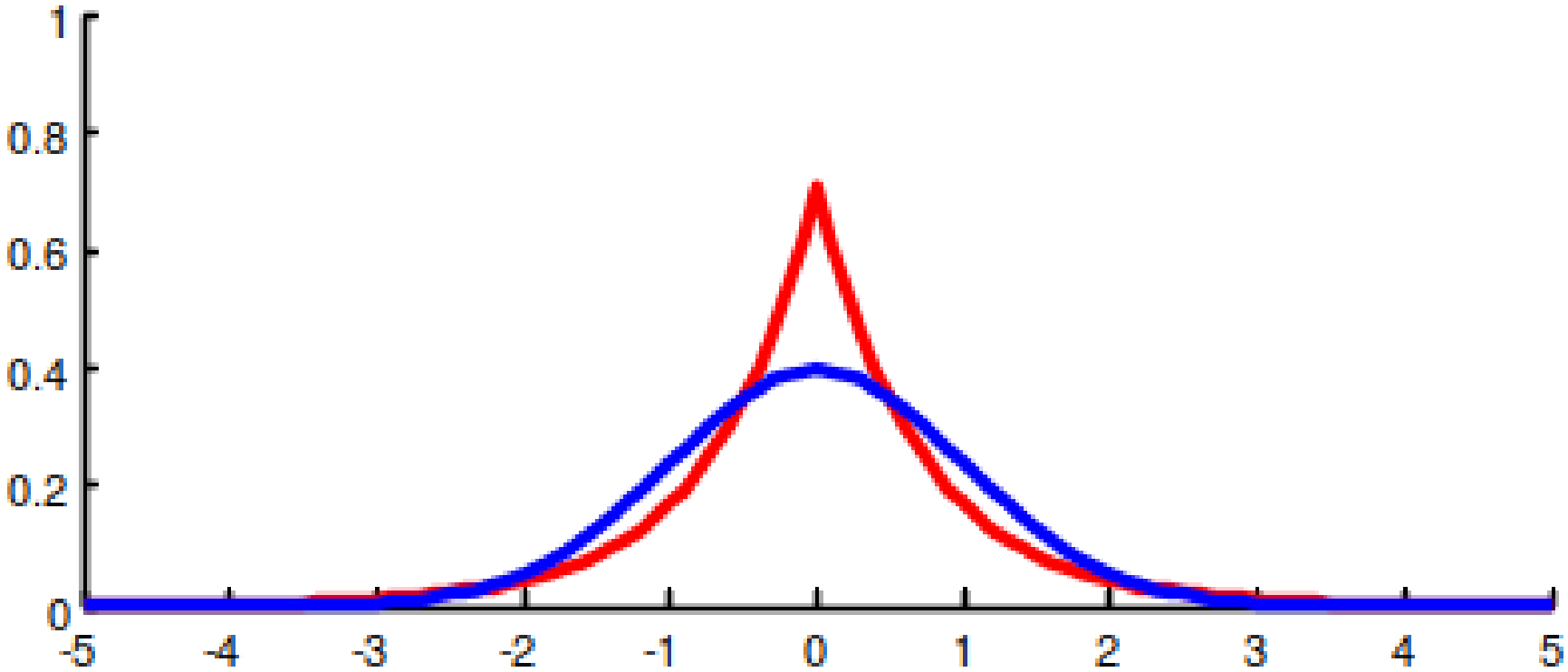
short tails
outliers uncommon
outliers important
use high norm
gives high weight to outliers

from Menke

Vector Norms

Distance Metrics

- L_2 : Gaussian Distribution
- L_1 : Laplace Distribution



from Menke

Loss Functions in Classification Tasks

- **The Hinge Loss function:** Used in binary Classification (e.g., SVM); Output is a single value $\hat{y}$ and the intended output y is in $\{+1, -1\}$.

$$L_{\text{hinge(binary)}}(y, \hat{y}) = \max(0, 1 - y.\hat{y}).$$

The loss is 0 when y and $\hat{y}$ share the same sign and $|\hat{y}| = 1$.

- **Entropy S**: We use entropy to indicate disorder or uncertainty. It is measured for a random variable X with probability distribution $p(X)$

$$S = -\sum_i p(x_i) \log p(x_i).$$

The negative sign is used to make the overall quantity positive. This makes binary cross-entropy suitable as a loss function – you want to minimize its value.

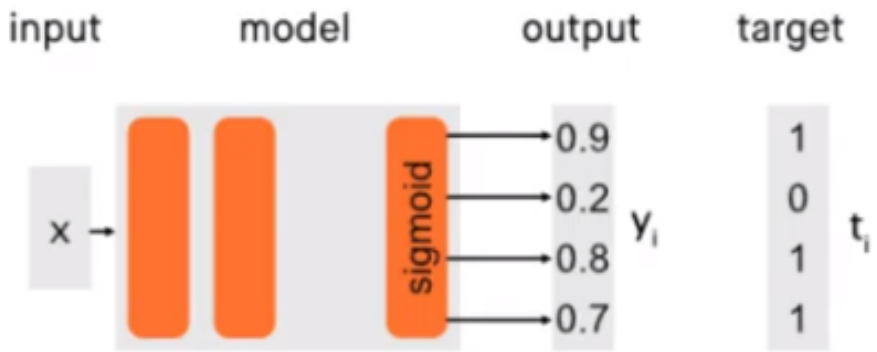
- **Binary Cross-Entropy Loss:** We use binary cross-entropy loss for classification models which output a probability p Then, the cross-entropy loss for output label y (can take values 0 and 1) and predicted probability p is defined as:

$$L = -y * \log(p) - (1 - y) * \log(1 - p) = \begin{cases} -\log(1 - p), & \text{if } y = 0 \\ -log(p) & \text{if } y = 1 \end{cases}$$

To calculate probability p, we can use the sigmoid function!

Binary Cross-Entropy

$$\text{Loss} = \sum_{\substack{\text{output} \\ \text{size}}} - t_i \log(y_i) - (1-t_i) \log(1-y_i)$$



The Least Squares : Analytic Approach

- We consider here a linear problem such that $\hat{\mathbf{y}} = \mathbf{Gm}$. We seek an optimal set of model parameters $\mathbf{m}^*$ such that the loss function $L = (\mathbf{y} - \mathbf{Gm})^T(\mathbf{y} - \mathbf{Gm})$ is minimum!
- Set $\frac{\partial L}{\partial \mathbf{m}} = 0$ to obtain:

$$\boxed{\mathbf{m}^* = \left[\mathbf{G}^T\mathbf{G}\right]^{-1}\mathbf{G}^T\mathbf{y}.}$$

- We assume that $\left[\mathbf{G}^T\mathbf{G}\right]^{-1}$ exits.
- Constraints/Regularization to address non-uniqueness and stability.

- We minimize L with the constraint the length of $\mathbf{m}$ is minimum.
- The modified Loss function is then $\phi(\mathbf{m}) = L + \epsilon^2 \mathbf{m}^T \mathbf{m}$, ϵ is a weight, called **regularization weight**.
- Set $\frac{\partial \phi}{\partial \mathbf{m}} = 0$ to obtain:

$$\boxed{\mathbf{m}^* = \left[\mathbf{G}^T \mathbf{G} + \epsilon^2 \mathbf{I}\right]^{-1} \mathbf{G}^T \mathbf{y}.}$$

- **The Damped Least Squares!**
- ϵ : Damping parameter.

- $\mathbf{m}$ is close to an a priori model$< \mathbf{m} >$. The modified Loss function is then $\phi(\mathbf{m}) = L + (\mathbf{m} - < \mathbf{m} >)^T (\mathbf{m} - < \mathbf{m} >)$.
- $\mathbf{m}$ varies slowly with position, i.e., $\tilde{\mathbf{m}} = \mathbf{D}\mathbf{m}$, where

$$\mathbf{D} = \begin{bmatrix} -1 & 1 & & & & & \\ & -1 & 1 & & & & \\ & & -1 & 1 & & & \\ & & & & & & \\ & & & & & & \\ & & & & & -1 & 1 \end{bmatrix}, \text{ or } \begin{bmatrix} 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & 1 & -2 & 1 & & \\ & & & & & & \\ & & & & & & \\ & & & & & 1 & -2 & 1 \end{bmatrix}$$

The loss function: $\phi(m) = L + \tilde{\mathbf{m}}^T \tilde{\mathbf{m}}$.

- The loss function: The modified Loss function is

$$\phi(\mathbf{m}) = (\mathbf{y} - \mathbf{Gm})^T \mathbf{W}_y (\mathbf{y} - \mathbf{Gm}) + \epsilon^2 (\mathbf{m} < \mathbf{m} >)^T \mathbf{W}_m (\mathbf{m} - < \mathbf{m} >).$$

- Solution:

$$[\mathbf{G}^T \mathbf{W}_y \mathbf{G} + \epsilon^2 \mathbf{W}_m] \mathbf{m}^* = \mathbf{G}^T \mathbf{W}_y \mathbf{y} + \epsilon^2 \mathbf{W}_m < \mathbf{m} > .$$

- A General Loss Function:

$$\boxed{\phi(\mathbf{m}) = ||\mathbf{y} - \hat{\mathbf{y}}||_p + \epsilon^2 ||\mathbf{m} - < \mathbf{m} > ||_p}$$

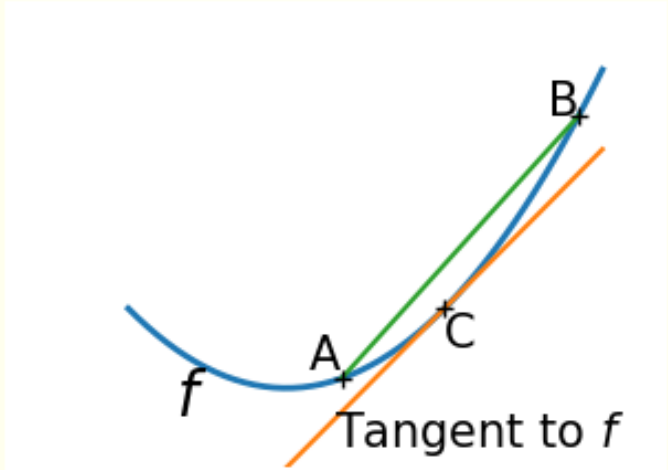
Optimization

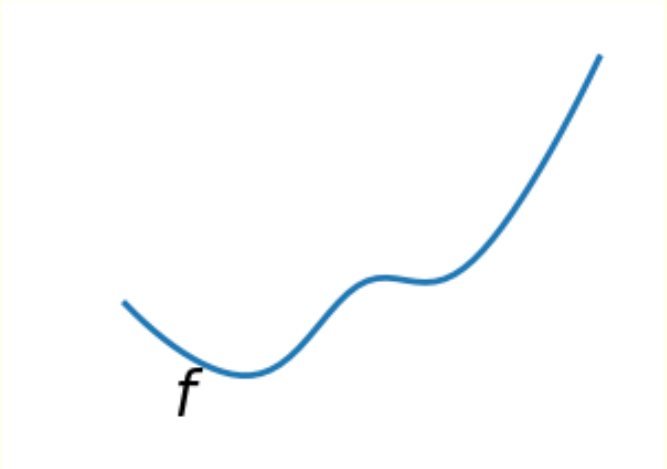
Introductions

- An optimization problem begins with a set of variables or parameters, and often includes conditions or restrictions that define acceptable values of the variables - **Constraints**.
- The other essential component of an optimization problem is a singlemeasure of goodness termed objective function that depends in some way on the variables.
- When using statistical models, the definition of the objective function also depends on the noise in the data and the pdf defining our prior knowledge of the model.
- The objective function has also been called
  - Loss function (in Machine Learning)
  - A fitness or misfit function (in Genetic Algorithms).
  - Energy function (in Simulated Annealing where one attempts to attain a minimum energy state).
  - Cost function (in traveling salesman problem).
- Solution: A set of allowed values of the variables for which the objective function assumes an optimal value.

Convex versus non-convex Optimization

2.7.1.1. Convex versus non-convex optimization





A convex function:

- f is above all its tangents.
- equivalently, for two point A, B, f(C) lies below the segment [f(A), f(B)]], if A < C < B

A non-convex function

Optimizing convex functions is easy. Optimizing non-convex functions can be very hard.

Note: It can be proven that for a convex function a local minimum is also a global minimum. Then, in some sense, the minimum is unique.

http://scipy-lectures.org/advanced/mathematical_optimization/

- A twice continuously differentiable function $L(\mathbf{w})$.
- We impose the requirement for a decrease in L at every iteration (**epoch**).

A Model Algorithm

- Guess $\mathbf{w}_0$, set $k \leftarrow 0$
- while $L(\mathbf{w}) \geq \epsilon$ do
- Compute a search direction $\Delta\mathbf{w}_k$
- Compute a step length (**learning rate**) α_k
- $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \alpha_k \ \Delta\mathbf{w}_k$
- $k \leftarrow k + 1$
- enddo
- return $\mathbf{w}_k$

- When we take a step, we choose the direction in which L decreases most quickly, which is the direction opposite to $L'(\mathbf{w}_i)$

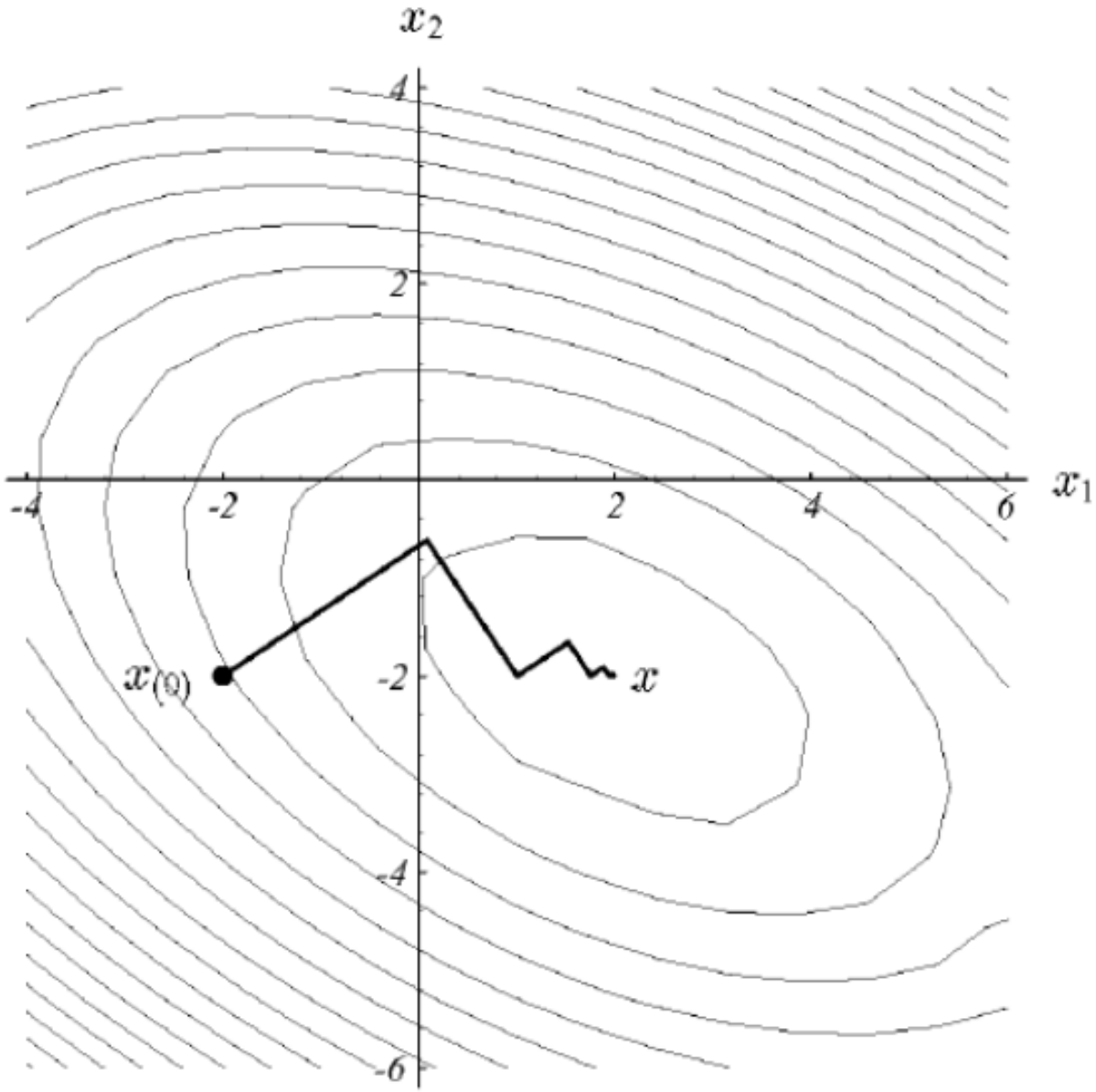
Steepest Descent Algorithm

- Guess $\mathbf{w}_0$, set $k \leftarrow 0$
- for k in range(0:n_epochs):
- $\quad \mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \alpha_k \nabla L(\mathbf{w}_k)$
- $\quad k \leftarrow k + 1$
- return $\mathbf{w}_k$

compute α_k [**learning rate**] by line search at each iteration.

Methods for multi-variate functions

Steepest Descent



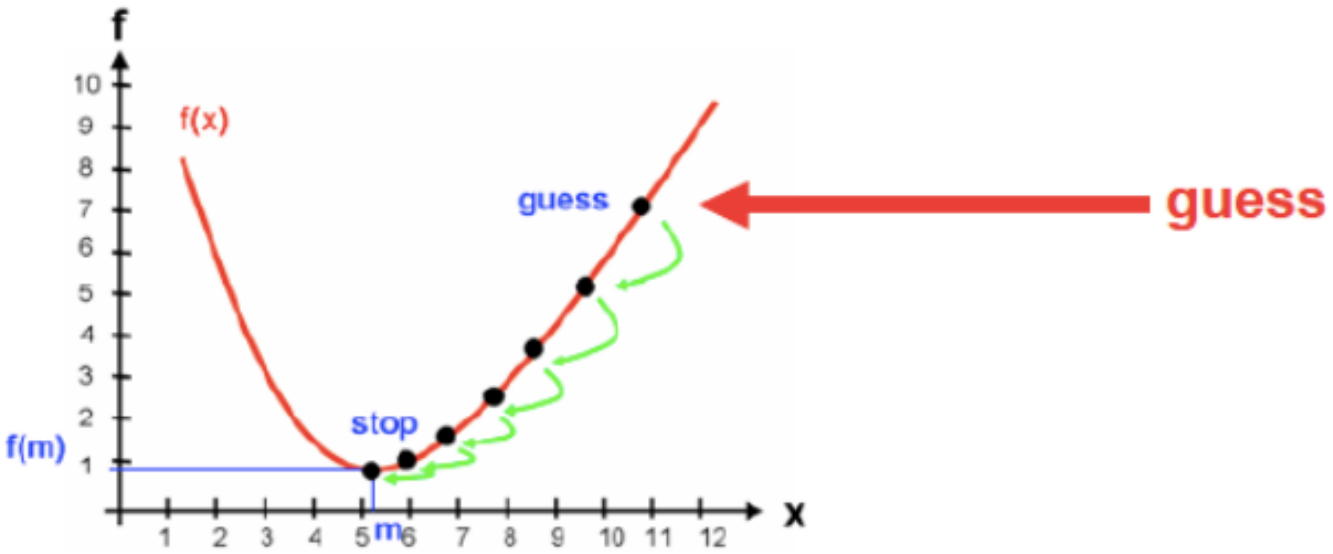
Gradient descent: algorithm

Start with a point (guess)

Repeat

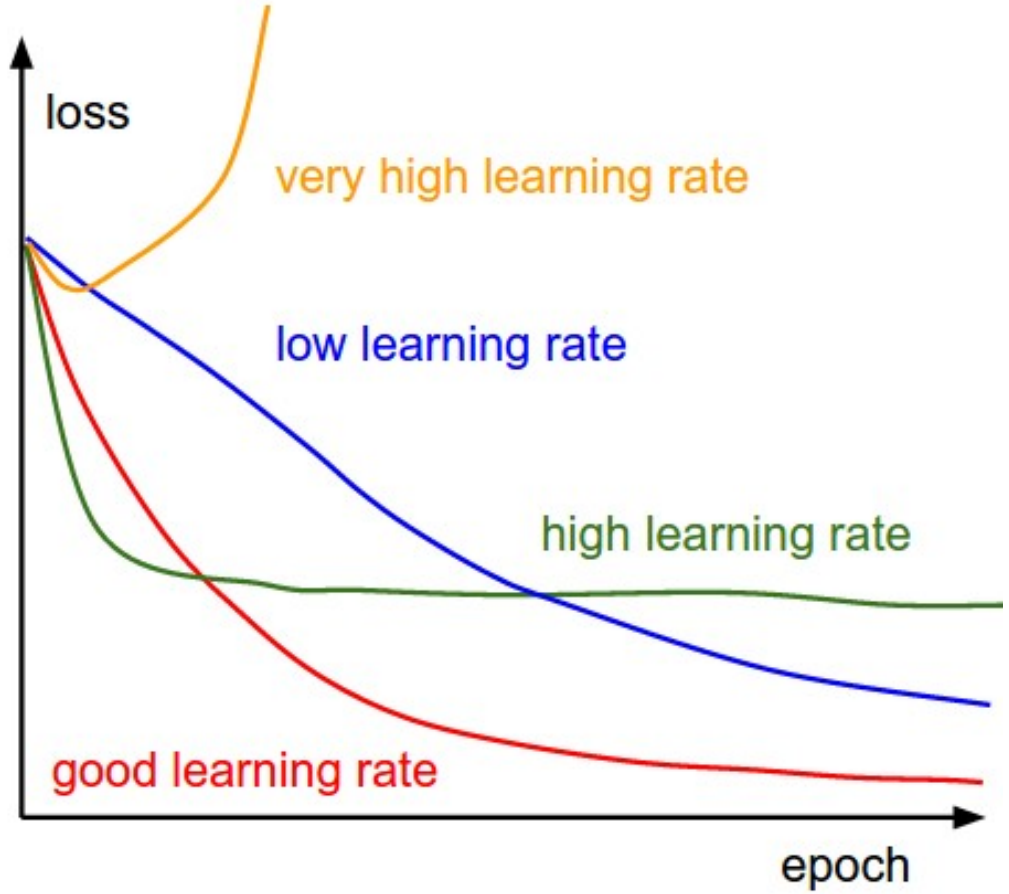
- Determine a descent direction
- Choose a step
- Update

Until stopping criterion is satisfied



Methods for multi-variate functions

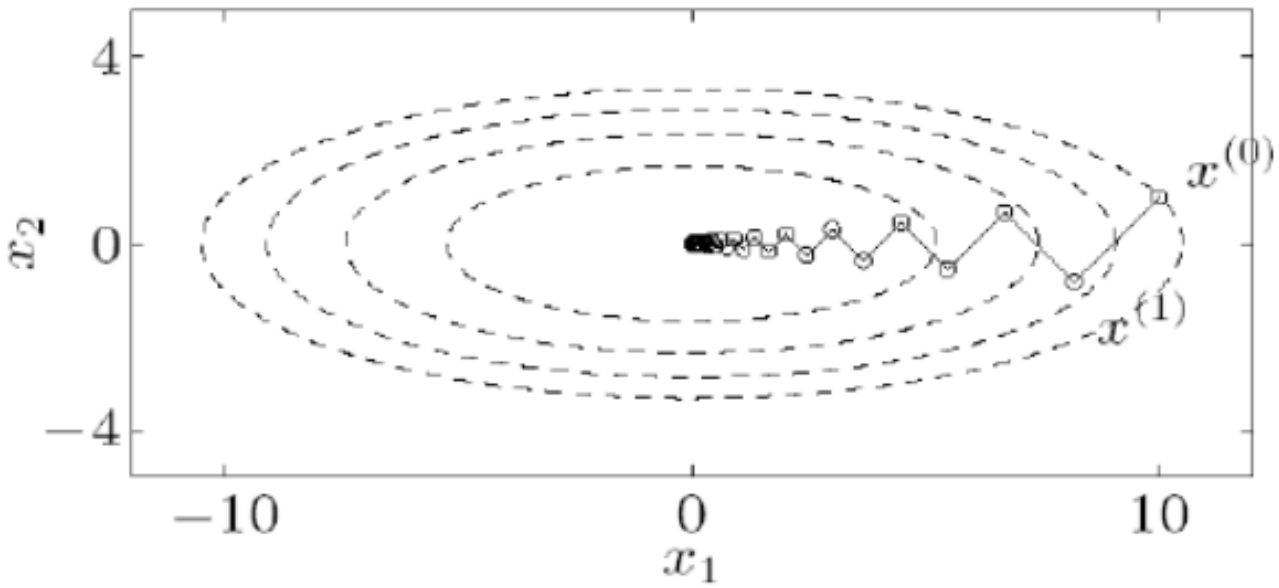
Steepest Descent



Methods for multi-variate functions

Problems with Steepest Descent

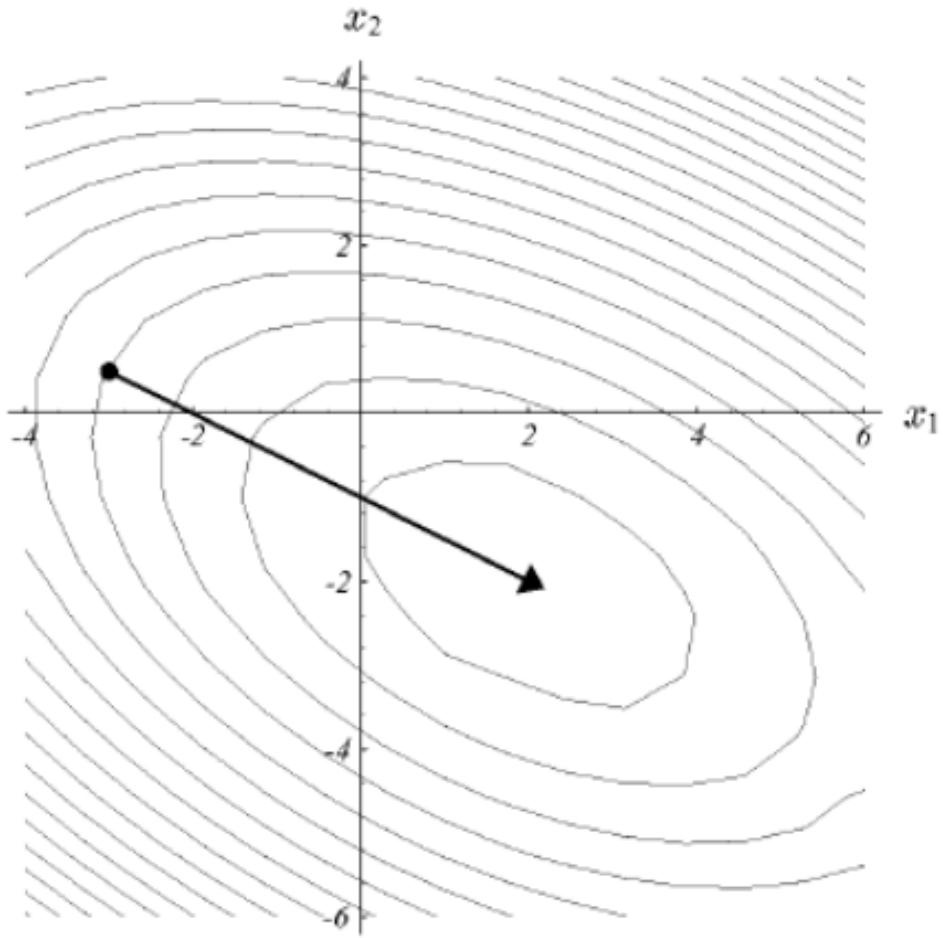
$$f(x) = (1/2)(x_1^2 + \gamma x_2^2) \qquad (\gamma > 0)$$



[S. Boyd, L. Vandenberghe, Convex Convex Optimization lect. Notes, Stanford Univ. 2004]

Methods for multi-variate functions

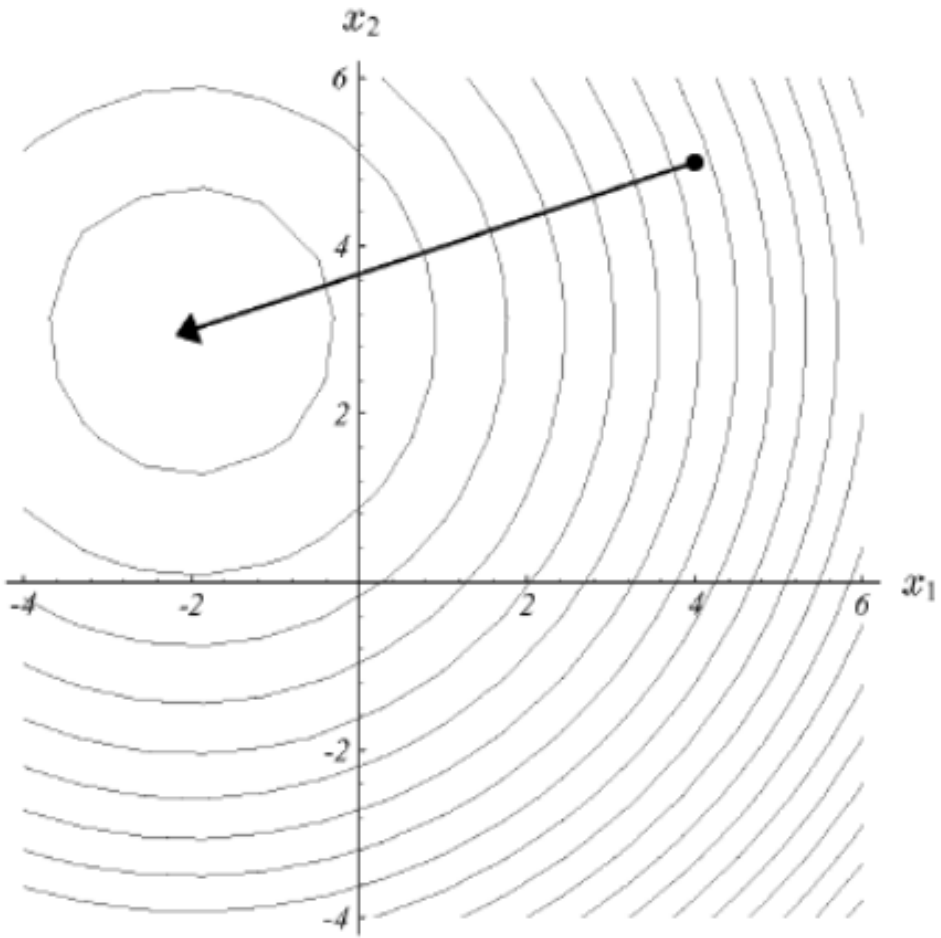
Problems with Steepest Descent



Steepest Descent converges to the exact solution on the first iteration if the error term is an eigenvector.

Methods for multi-variate functions

Problems with Steepest Descent



Steepest Descent converges to the exact solution on the first iteration if the eigenvalues are all equal.

Conjugate Gradient Algorithm

- Guess $\mathbf{w}_0$, set $\mathbf{r}_0 \leftarrow \mathbf{b}$; $\mathbf{p}_0 \leftarrow \mathbf{r}_0$;n $\leftarrow 1$
- do
- $\alpha_n \leftarrow \frac{\mathbf{r}_{n-1}^T \mathbf{r}_{n-1}}{\mathbf{p}_{n-1}^T \mathbf{A}\mathbf{p}_{n-1}}$ step length
- $\mathbf{w}_n \leftarrow \mathbf{w}_{n-1} + \alpha_n \mathbf{p}_{n-1}$ model update
- $\mathbf{r}_n \leftarrow \mathbf{r}_{n-1} - \alpha_n \mathbf{A}\mathbf{p}_{n-1}$ update residual
- $\beta_n \leftarrow \frac{\mathbf{r}_n^T \mathbf{r}_n}{\mathbf{r}_{n-1}^T \mathbf{r}_{n-1}}$ gradient correction factor
- $\mathbf{p}_n \leftarrow \mathbf{r}_n + \beta_n \mathbf{p}_{n-1}$ New search direction
- $n \leftarrow n + 1$
- enddo
- return $\mathbf{w}_n$

Non-linear CG: CG can be used not only to find the minimum point of a quadratic form, but to minimize any continuous function $L(\mathbf{w})$ for which the gradient $L'(\mathbf{w})$ can be computed.
Here we set the residual to the negation of the gradient; i.e.,

$$\mathbf{r}_n \leftarrow F'(\mathbf{w}_n).$$

Our objective is to find the minimum of our function $L(\mathbf{w})$. At $\mathbf{w} = \tilde{\mathbf{w}}$, we approximate $F(\mathbf{w})$ as follows:

$$L(\mathbf{w}) \approx h(\mathbf{w}) := L(\tilde{\mathbf{w}}) + (\nabla L(\tilde{\mathbf{w}})^T(\mathbf{w} - \tilde{\mathbf{w}}) + \frac{1}{2}(\mathbf{w} - \tilde{\mathbf{w}})^T H(\tilde{\mathbf{w}})(\mathbf{w} - \tilde{\mathbf{w}}),$$

which is the quadratic Taylor expansion of $L(\mathbf{w})$ at $\mathbf{w} = \tilde{\mathbf{w}}$. ∇L and H are the gradient and Hessian respectively of $L(\mathbf{w})$.
We minimize $h(\mathbf{w})$ by setting $\nabla h(\mathbf{w}) = 0$. That is

$$\nabla h(\mathbf{w}) = \nabla L(\tilde{\mathbf{w}}) + H(\tilde{\mathbf{w}})(\mathbf{w} - \tilde{\mathbf{w}}) = 0,$$

which yields

$$\mathbf{w} - \tilde{\mathbf{w}} = -H^{-1}(\tilde{\mathbf{w}})\nabla L(\tilde{\mathbf{w}}). \tag{1}$$

The direction $-H^{-1}(\tilde{\mathbf{w}})\nabla L(\tilde{\mathbf{w}})$ is called the *Newton direction* or *Newton Step* at $\mathbf{w} = \tilde{\mathbf{w}}$.

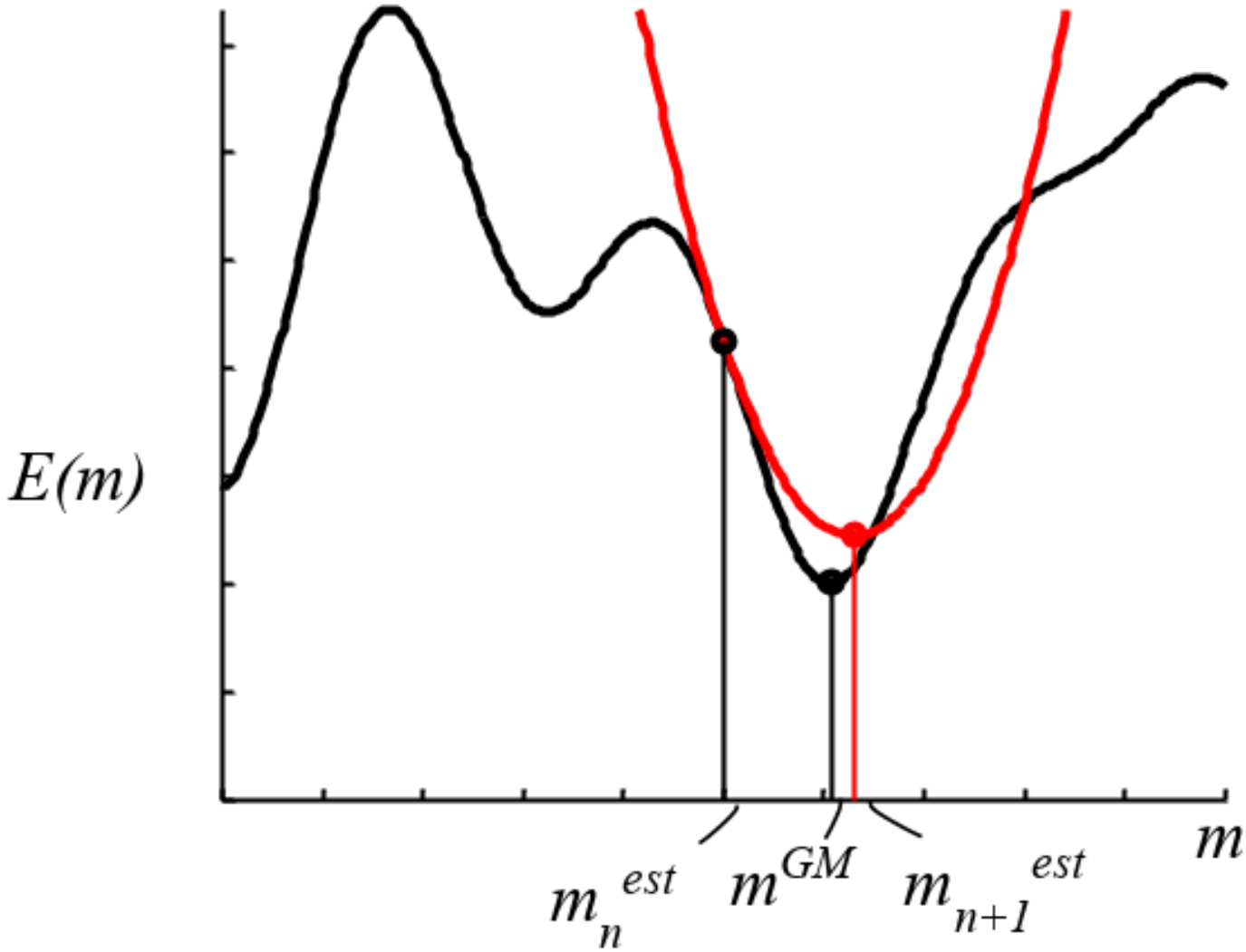
Newton's Algorithm

- Guess $\mathbf{w}_0$, set $k \leftarrow 0$
- while $||\nabla L(\mathbf{w})|| \geq \epsilon$ do
- $\quad \mathbf{d}_k \leftarrow -H^{-1}(\mathbf{w}_k)\nabla L(\mathbf{w_k})$
- $\quad$ Choose α_k
- $\quad \mathbf{w_{k+1}} \leftarrow \mathbf{w_k} + \alpha_k \mathbf{d}_k$
- $\quad k \leftarrow k + 1$
- enddo
- return $\mathbf{w}_k$

compute α by 1D minimization.

Methods for multi-variate functions

Newton's Method



Methods for multi-variate functions

Gauss-Newton

Let us consider our standard L_2 norm objective function, given by

$$L(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - f(\mathbf{w}, \mathbf{x}))^T(\mathbf{y} - f(\mathbf{w}, \mathbf{x})),$$

$$\nabla L(\mathbf{w}_k) = -\mathbf{J}(\mathbf{w}_k)^T(\mathbf{y} - f(\mathbf{w}_k)),$$

$$H(\mathbf{w}_k) = -\frac{\partial \mathbf{J}(\mathbf{w}_k)}{\partial \mathbf{w}} + \mathbf{J}^T(\mathbf{w}_k)\mathbf{J}(\mathbf{w}_k),$$

where $\mathbf{J}(\mathbf{w}_k) = \frac{\partial f(\mathbf{w}_k)}{\partial \mathbf{w}}$ is called the *Jacobian Matrix*. Note that if our problem was linear, $\mathbf{J}$ would be a constant and its derivative $\frac{\partial \mathbf{J}(\mathbf{w}_k)}{\partial \mathbf{w}}$ would be zero. Assuming that our problem is close to linear, we ignore the first term in the right hand side of Eq. (26), which gives

$$H(\mathbf{w}_k) \approx 2\mathbf{J}^T(\mathbf{w}_k)\mathbf{J}(\mathbf{w}_k),$$

Substituting Eq. (15) and (27)

$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + [\mathbf{J}^T(\mathbf{w}_k)\mathbf{J}(\mathbf{w}_k)]^{-1}\mathbf{J}^T(\mathbf{w}_k)[\mathbf{y} - f(\mathbf{w}))].$$

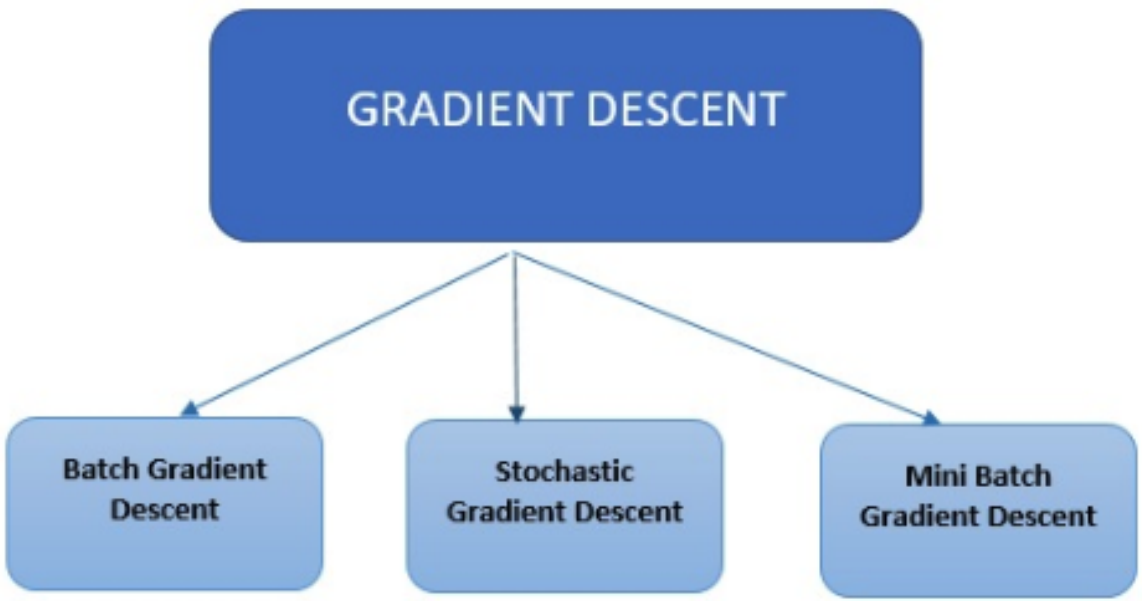
This is the very reminiscent of the Least Squares!
In the **Lavenberg-Marquardt (LM) method**, the GN method is modified as follows:

$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + [\mathbf{J}^T(\mathbf{w}_k)\mathbf{J}(\mathbf{w}_k) + \lambda\mathbf{I}]^{-1}\mathbf{J}^T(\mathbf{w}_k)[\mathbf{y} - f(\mathbf{w}))], \tag{2}$$

which reminds us of *damped least squares*.

Methods for multi-variate functions

Gradient Descent



Methods for multi-variate functions

Stochastic Gradient Descent

In the standard gradient descent approach, we compute the gradient of the cost function with respect to the parameters **w**. Such a method would be called **Batch Gradient Descent**.

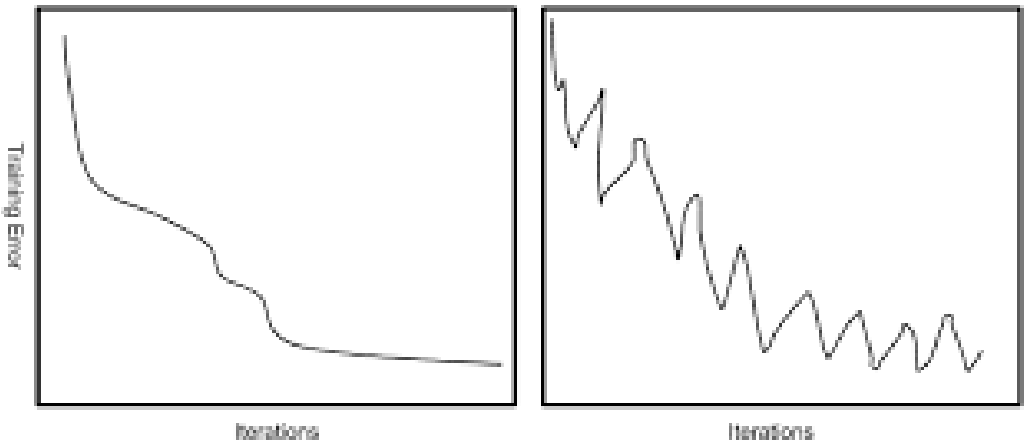
- **Stochastic Gradient Descent** in contrast, performs a parameter update for each training example x^i, and label y^i:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}; x^i; y^i)$$

- Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online.
- SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily as in the image below.

Methods for multi-variate functions

SGD Method



Methods for multi-variate functions

Mini-batch SGD Method

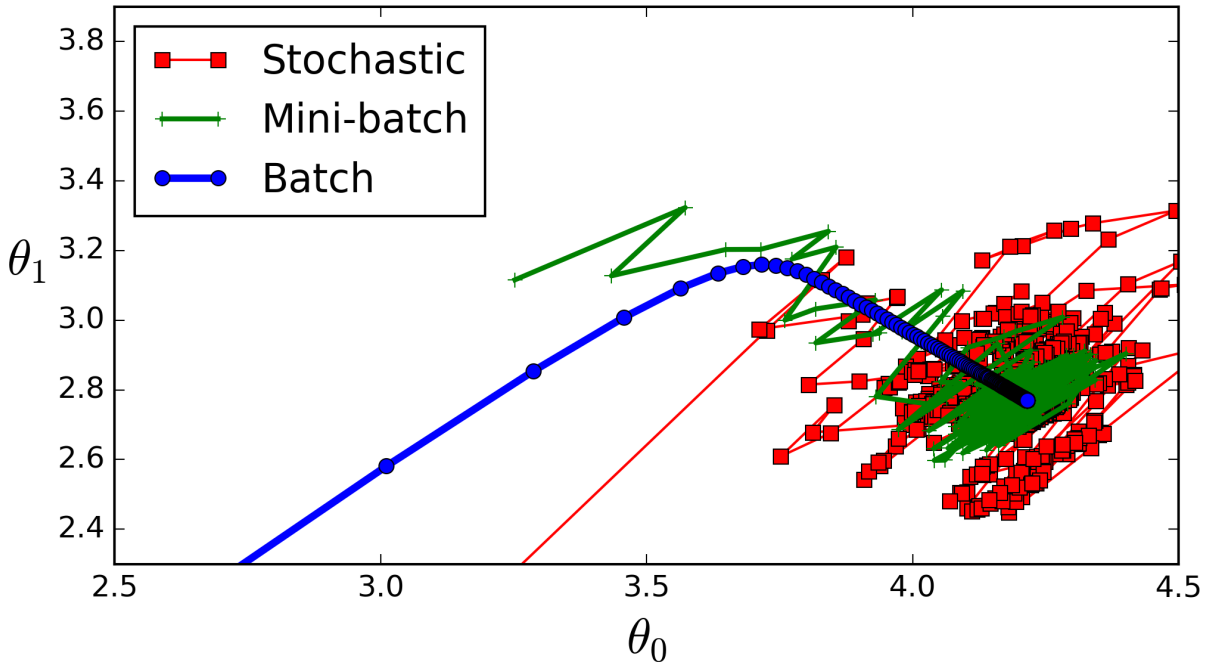
- Mini-batch gradient descent finally takes the best of both worlds and performs an update for every mini-batch of n training examples:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}; x^{(i:i+n)}; y^{(i:i+n)}).$$

- This way, it a) reduces the variance of the parameter updates, which can lead to more stable convergence; and b) can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient w.r.t. a mini-batch very efficient. Common mini-batch sizes range between 50 and 256, but can vary for different applications. Mini-batch gradient descent is typically the algorithm of choice when training a neural network and the term SGD usually is employed also when mini-batches are used.

Methods for multi-variate functions

Mini-batch SGD Method



Batch Gradient Descent

Advantages of Batch Gradient Descent

- Less oscillations and noisy steps taken towards the global minima of the loss function due to updating the parameters by computing the average of all the training samples rather than the value of a single sample.
- It can benefit from the vectorization which increases the speed of processing all training samples together.
- It produces a more stable gradient descent convergence and stable error gradient than stochastic gradient descent.
- It is computationally efficient as all computer resources are not being used to process a single sample rather are being used for all training samples.

Disadvantages of Batch Gradient Descent

- Sometimes a stable error gradient can lead to a local minima and unlike stochastic gradient descent no noisy steps are there to help get out of the local minima.
- The entire training set can be too large to process in the memory due to which additional memory might be needed.
- Depending on computer resources it can take too long for processing all the training samples as a batch.

https://medium.com/@divakar_239/stochastic-vs-batch-gradient-descent-8820568eada1

Stochastic Gradient Descent

Advantages of Stochastic Gradient Descent

- It is easier to fit into memory due to a single training sample being processed by the network.
- It is computationally fast as only one sample is processed at a time.
- For larger datasets it can converge faster as it causes updates to the parameters more frequently.
- Due to frequent updates the steps taken towards the minima of the loss function have oscillations which can help getting out of local minimums of the loss function (in case the computed position turns out to be the local minimum.)

Disadvantages of Stochastic Gradient Descent

- Due to frequent updates the steps taken towards the minima are very noisy. This can often lead the gradient descent into other directions.
- Also, due to noisy steps it may take longer to achieve convergence to the minima of the loss function.
- Frequent updates are computationally expensive due to using all resources for processing one training sample at a time.
- It loses the advantage of vectorized operations as it deals with only a single example at a time.

`https://medium.com/@divakar_239/stochastic-vs-batch-gradient-descent-8820568eada1`

Mini-batch Gradient Descent

Advantages of Mini-batch Gradient Descent

- This method ensures the following advantages of both stochastic and batch gradient descent are used due to which Mini Batch Gradient Descent is most commonly used in practice.
  1. Easily fits in the memory.
  2. It is computationally efficient.
  3. Benefit from vectorization.
  4. If stuck in local minimums, some noisy steps can lead the way out of them.
  5. Average of the training samples produces stable error gradients and convergence.

https://medium.com/@divakar_239/stochastic-vs-batch-gradient-descent-8820568eada1

Different Gradient Descent Algorithms

Optimiser	Year	Learning Rate	Gradient
Momentum	1964		✓
AdaGrad	2011	✓	
RMSprop	2012	✓	
Adadelta	2012	✓	
Nesterov	2013		✓
Adam	2014	✓	✓
AdaMax	2015	✓	✓
Nadam	2015	✓	✓
AMSGrad	2018	✓	✓

https://towardsdatascience.com/10-gradient-descent-optimisation-algorithms-86989510b5e9

Different Gradient Descent Algorithms

- **SGD**

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha \nabla_{\mathbf{w}_k} L.$$

- **Momentum**: Instead of depending only on the current gradient to update the weight, gradient descent with momentum (Polyak, 1964) replaces the current gradient with p "momentum", which is an **aggregate of gradients**. This aggregate is the exponential moving average of current and past gradients (i.e. up to iteration k).

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha \mathbf{p}_k,$$

$$\mathbf{p}_k = \beta \mathbf{p}_{k-1} - (1-\beta) \nabla_{\mathbf{w}_k} L.$$

common default value $\beta = 0.9$, and $\mathbf{p}_0 = 0$.

https://towardsdatascience.com/10-gradient-descent-optimisation-algorithms-86989510b5e9

Different Gradient Descent Algorithms

- **ADAGRAD**: Adaptive gradient, or AdaGrad (Duchi et al., 2011), adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features. For this reason, it is well-suited for dealing with sparse data. All methods we described so far perform an update for all parameters w at once as every parameter w_i use the same learning rate α. Adagrad uses a different learning rate for every parameter w_i at each iteration k.

$$\mathbf{g}_{k,i} = \nabla_{\mathbf{w}_k} L(\mathbf{w}_k, i).$$

$$\mathbf{w}_{k+1,i} = \mathbf{w}_{k,i} - \frac{\alpha}{\sqrt{G_{k,ii}}} \mathbf{g}_{k,i}.$$

$\mathbf{G}_k$ here is a diagonal matrix where each diagonal element is (i, i) .

ADAM Optimization

ADAM: Adaptive Moment Estimation

"We introduce Adam, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal re-scaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters." [Kingma and Ba 2017]

What is Moment in the context of SGD?

Recall that the simple SD update rule is given by:

$$\Delta \mathbf{w}_k = \eta \nabla_{\mathbf{w}} L(\mathbf{w}_k),$$

where k is the epoch, η is the learning rate.

ADAM Optimization

The inclusion of a momentum term[1] has been found to increase the rate of convergence dramatically (Rumelhart, Hinton and Williams, 1986). With the momentum term, we have

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mathbf{p}_k, \tag{3}$$

where

$$\mathbf{p}_k = \beta \mathbf{p}_{k-1} + \alpha \nabla_{\mathbf{w}} L(\mathbf{w}_k), \tag{4}$$

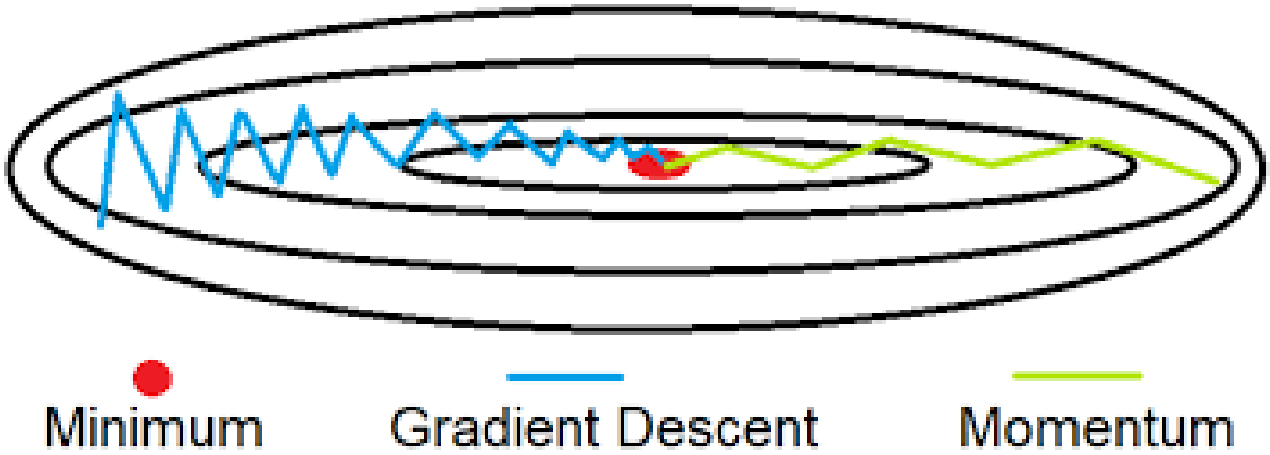
where $\mathbf{p}_k$, and $\mathbf{p}_{k-1}$ are the **moments** at epochs k and $(k-1)$ respectively, is the learning rate and β is the desired faction of the previous update, called the **momentum parameter**. That is, **the update of the weight vector at the current time step depends on both the current gradient and the weight change of the previous step**.

Intuitively, the rational for the use of the momentum term is that the steepest descent is particularly slow when there is a long and narrow valley in the error function surface. In this situation, the direction of the gradient is almost perpendicular to the long axis of the valley. The system thus oscillates back and forth in the direction of the short axis, and only moves

ADAM Optimization

- $\alpha \leftarrow 0.01$, $\epsilon \leftarrow 10^{-8}$ learning rate and tolerance
- $\beta_1 \leftarrow 0.9$, $\beta_2 \leftarrow 0.9999$ moment parameters (default)
- $\mathbf{p}_0 \leftarrow 0$, $\mathbf{q}_0 \leftarrow 0$ Momemntum vectors
- θ_0 Initial model vector
- $k \leftarrow 0$, Initialize iteration number
- while not converged do
- $k \leftarrow k + 1$
- $\mathbf{g}_k \leftarrow \nabla_{\mathbf{w}} L(\mathbf{w}_{k-1})$
- $\mathbf{p}_k \leftarrow \beta_1 \mathbf{p}_{k-1} + (1 - \beta_1)\mathbf{g}_k$
- $\mathbf{q}_k \leftarrow \beta_2 \mathbf{q}_{k-1} + (1 - \beta_2)(\mathbf{g}_k . \mathbf{g}_k)$ element-wise
- $\hat{\mathbf{p}}_k \leftarrow \mathbf{p}_k / (1 - (\beta_1)^k)$ Bias-corrected first moment
- $\hat{\mathbf{q}}_k \leftarrow \mathbf{q}_k / (1 - (\beta_2)^k)$ Bias-corrected second moment
- $\mathbf{w}_k \leftarrow \mathbf{w}_{k-1} - \alpha \mathbf{p}_k / (\sqrt{\hat{\mathbf{q}}_k} + \epsilon)$ Model Update
- enddo
- return $\mathbf{w}_k$

ADAM



LASSO

LASSO: Least Absolute Shrinkage and Selection Operator

Loss Function:

$$L(\mathbf{w}, \mathbf{y}) = ||\mathbf{y} - \hat{\mathbf{y}}||_2 + \alpha||\mathbf{w}||_1.$$