Parameter Estimation in Machine Learning

Yuetian Li
li.yuet@northeastern.edu

1 Introduction

Parameter Estimation is perhaps the single most important thing in machine learning. In general, what a machine learning algorithm is doing is all about *estimating* the *parameters* in a function that can describe a phenomenon.

From traditional algorithms like Linear Regression and Naive Bayes to modern algorithms like Deep Neural Networks, their essence of trying to do parameter estimation remain unchanged.

2 Intuition

To address again, in machine learning, we hypothesize arbitrarily that there must be a function that well represents how the features of an observation are related to its label.

However we do not know the parameters that defines the function, thus, we need to estimate them.

We may take a function as a example:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

And it may be generalized into matrix form when we have multiple x_1 to x_n as:

$$\mathbf{y} = \mathbf{X} \cdot \mathbf{w}$$

Where:

$$\mathbf{X} = \begin{bmatrix} 1 & \mathbf{x}_1 \\ 1 & \mathbf{x}_2 \\ \vdots & \vdots \\ 1 & \mathbf{x}_m \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1n} \\ 1 & x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

- $\mathbf{X}$ is the *feature*.
  - Each line in the matrix is an *observation* in the data set with n features.
  - There are m lines in the matrix, which indicates there are m observations in the data set.
  - For example, x_{11} to x_{1n} is the n features for observation x_1.
- $\mathbf{y}$ is the *label*.
  - For example, y_1 is the label for observation $\mathbf{x}_1$ with feature x_{11} to x_{1n}.
  - There are m lines in the matrix, which also indicates that there are m observations in the data set.
- $\mathbf{w}$ is the *parameters* that we want to estimate.
  - Where $\mathbf{w}$ stands for *weight*.
  - The w_0 here is also called *bias*, which sometimes denoted as b.

In real world pratices, features and labels are usually given together, so we may also indicate the data set $\mathcal{D}$ as:

$$\mathcal{D} = \{\mathbf{X}; \mathbf{y}\} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1n} & y_1 \\ x_{21} & x_{22} & x_{23} & \cdots & x_{2n} & y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{m1} & x_{m2} & x_{m3} & \cdots & x_{mn} & y_m \end{bmatrix}$$

And each line is an observation in the data set.

In this case, the goal is to accurately estimate the *parameter* $\mathbf{w}$ that when we calculate the *prediction* $\hat{\mathbf{y}}$ with $\mathbf{X}$ and $\mathbf{w}$ by

$$\hat{\mathbf{y}} = \mathbf{X} \cdot \mathbf{w}$$

And the resulting $\hat{\mathbf{y}}$ and the true label $\mathbf{y}$ could be close enough. If they are close enough, we can say we are good with the estimation of the *parameters* $\mathbf{w}$.

3 Estimation Methods

So how should we estimate the parameters?

In general, our strategy is to choose the parameter that maximize the likelihood between the prediction and the true label, or minimize the loss between the prediction and the label. We may follow a procedure as:

1. Define a function that represents the likelihood or loss. It is usually called loss function or objective function.
2. Calculate partial differentiation of the parameter on the loss function to find the stationary point of the function, which gives us the optimal value for the parameters.

The key is to define the loss function. Once you have the loss function on hand, all you need to do is to optimize the loss function with the parameters.

There are two ways to think about defining the loss function, Maximum Likelihood Estimation (MLE) and Maximum A-Posteriori Estimation (MAP).

Estimating the paramter with MLE is also referred as *Frequentist interpretation*, and MAP as *Bayesian interpretation*.

3.1 Least Squares Estimation (LSE)

We may start with Least Squares Estimation (LSE), which is a special form of Maximum Likelihood Estimation (MLE). We may define the loss function as:

$$\begin{aligned}\hat{\mathbf{w}}_{LSE} &= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 \\ &= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{N} (x_i w_i - y_i)^2 \\ &= \underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{X} \cdot \mathbf{w} - \mathbf{y})^2\end{aligned}$$

It means choosing the parameter $\mathbf{w}$ that minimize $(\mathbf{X} \cdot \mathbf{w} - \mathbf{y})^2$, or say minimize $\sum_{i=1}^{N} (x_i w_i - y_i)^2$, which is denoted by $\hat{\mathbf{w}}$.

Exercise 3.1.1

Given the function from the *Intuition* section:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Derive least square estimation for parameter $\mathbf{w} = (w_0, w_1, \cdots, w_n)$.

Solution 3.1.1

We may define a loss function $\mathcal{L}$ to represent the loss between $\hat{\mathbf{y}}$ and $\mathbf{y}$:

$$\begin{aligned}\mathcal{L}(\mathbf{w}) &= \frac{1}{2}\sum_{i=1}^{n}(x_i w_i - y_i)^2 \\ &= \frac{1}{2}(\mathbf{X}\cdot\mathbf{w} - \mathbf{y})^2\end{aligned}$$

Calculate partial differentiation on $\mathbf{w}$*:*

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}}\mathcal{L} &= \frac{\partial}{\partial \mathbf{w}}\frac{1}{2}(\mathbf{X}\cdot\mathbf{w} - \mathbf{y})^2 \\ &= (\mathbf{X}^T - 0)\cdot(\mathbf{X}\cdot\mathbf{w} - \mathbf{y}) \\ &= \mathbf{X}^T\cdot\mathbf{X}\cdot\mathbf{w} - \mathbf{X}^T\cdot\mathbf{y} \\ &= 0\end{aligned}$$

So at this time:

$$\begin{aligned}&\mathbf{X}^T\cdot\mathbf{X}\cdot\mathbf{w} - \mathbf{X}^T\cdot\mathbf{y} = 0 \\ &\mathbf{X}^T\cdot\mathbf{X}\cdot\mathbf{w} = \mathbf{X}^T\cdot\mathbf{y} \\ &\mathbf{w} = (\mathbf{X}^T\cdot\mathbf{X})^{-1}\cdot\mathbf{X}^T\cdot\mathbf{y}\end{aligned}$$

Thus $\hat{\mathbf{w}} = (\mathbf{X}^T\cdot\mathbf{X})^{-1}\cdot\mathbf{X}^T\cdot\mathbf{y}$ *gives the least square estimation for our parameter* $\mathbf{w}$*.*

3.2 Maximum Likelihood Estimation (MLE)

In a more general definition, Maximum Likelihood Estimation (MLE) is to find the parameters that maximum the likelihood between $\hat{\mathbf{y}}$ and $\mathbf{y}$. We may define the loss function of Maximum Likelihood Estimation (MLE) as:

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}}\ p(\mathcal{D} \mid \theta)$$

Where:

- θ represent the parameter, that is the same thing as $\mathbf{w}$ indicate above.
- $\mathcal{D}$ represents the data set.
- $p(\mathcal{D} \mid \theta)$ represents the probability on the data set $\mathcal{D}$ calculated by the parameter θ. More specifically, it can be interpreted as $p(\hat{\mathbf{y}} \mid \mathbf{y})$ where $\mathbf{y}$ is from $\mathcal{D}$ and $\hat{\mathbf{y}}$ is calculated by $\hat{\mathbf{y}} = \mathbf{X}\cdot\mathbf{w}$ where $\mathbf{X}$ is from $\mathcal{D}$ and $\mathbf{w}$ is the parameter θ.
- $\hat{\theta}_{MLE}$ is the parameter that we find in this way.

It means finding the argument θ that maximize $p(\mathcal{D} \mid \theta)$.

Note:

You may actually derive MLE into LSE form, as LSE is a special form of MLE.

Exercise 3.2.1 Normal Distribution

Suppose the likelihood between $\mathbf{X}$ and $\mathbf{y}$ in the data set obeys Normal Distribution given as:

$$p(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(x_i-\mu)^2}$$

Derive maximum likelihood estimation for the parameter μ.

Solution 3.2.1

We may take logarithm of the original function as:

$$\begin{aligned}
\ln p(x \mid \mu, \sigma) &= \ln \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(x_i-\mu)^2} \\
&= \sum_{i=1}^{n} \ln(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(x_i-\mu)^2}) \\
&= \sum_{i=1}^{n} (\ln e^{-\frac{1}{2\sigma^2}(x_i-\mu)^2} - \ln\sqrt{2\pi}\sigma) \\
&= \sum_{i=1}^{n} -\frac{1}{2\sigma^2}(x_i-\mu)^2 - n\ln\sqrt{2\pi} - n\ln\sigma
\end{aligned}$$

Calculate partial differentiation on μ:

$$\begin{aligned}
\frac{\partial}{\partial\mu} \sum_{i=1}^{n} -\frac{1}{2\sigma^2}(x_i-\mu)^2 - n\ln\sqrt{2\pi} - n\ln\sigma = 0 \\
-\frac{\partial}{\partial\mu} \sum_{i=1}^{n} \frac{1}{2\sigma^2}(x_i-\mu)^2 = 0 \\
\sum_{i=1}^{n} (x_i-\mu)^2 = 0 \\
\sum_{i=1}^{n} x_i = n\mu \\
\mu_{MLE} = \frac{1}{n}\sum_{i=1}^{n} x_i
\end{aligned}$$

Thus $\mu_{MLE} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the maximum likelihood estimation for parameter μ.

3.3 Maximum A-Posteriori Estimation (MAP)

Maximum A-Posteriori Estimation (MAP) is another way to look at the problem. We may define the loss function as:

$$\hat{\theta}_{MAP} = \underset{\theta}{\text{argmax}}\ p(\theta \mid \mathcal{D}) = \underset{\theta}{\text{argmax}}\ \frac{p(\mathcal{D} \mid \theta)p(\theta)}{p(\mathcal{D})} = \underset{\theta}{\text{argmax}}\ p(\mathcal{D} \mid \theta)p(\theta)$$

Where:

- $p(\mathcal{D} \mid \theta)$ is the likelihood of paramter θ on data set $\mathcal{D}$, just like what we have in MLE.
- $p(\theta)$ is the conjugate prior probability of paramter θ.
- $\hat{\theta}_{MAP}$ is the paramter that we want to estimate.

It means finding the argument θ that maximize $p(\theta \mid \mathcal{D})$ or finding the argument θ that maximize $p(\mathcal{D} \mid \theta)p(\theta)$

In MAP Estimation, we multiply a prior probability of the parameter θ on the likelihood of $\mathcal{D}$. This is because we are thinking one step further than MLE: that we are taking the probability of the parameter θ into the consideration.

You may notice that if $p(\theta) = 1$, which means we are not considering the probability of the paramters θ, MAP becomes identical with MLE.

Exercise 3.3.1 Bernoulli Distribution

Suppose the likelihood between $\mathbf{X}$ and $\mathbf{y}$ in the data set obeys Bernoulli Distribution given as:

$$p_{Bernoulli}(x \mid \theta) = \theta^x(1-\theta)^x$$

And given Beta Distribution as its conjugate prior:

$$p_{Beta}(\theta \mid \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}$$

Where $\Gamma(n) = (n-1)!$ and α and β are constants.

Derive maximum a-posteriori estimation for the parameter θ.

Solution 3.3.1

Multiply Bernoulli function $p(x \mid \theta)$ and its conjugate prior Beta function $p(\theta \mid \alpha, \beta)$

$$\begin{aligned}\mathcal{L}(\theta \mid X) &\propto p(X \mid \theta) \cdot p(\theta \mid \alpha, \beta) \\ &= (\prod_{i=1}^{N} p_{Bernoulli}(x_i \mid \theta)) \cdot p_{Beta}(\theta \mid \alpha, \beta)\end{aligned}$$

Calculate log-likelihood for this:

$$\begin{aligned}\mathcal{LL}(\theta \mid X) &= \log \mathcal{L}(\theta \mid X) \\ &= \log((\prod_{i=1}^{N} p_{Bernoulli}(x_i \mid \theta)) \cdot p_{Beta}(\theta \mid \alpha, \beta)) \\ &= \sum_{i=1}^{N} \log p_{Bernoulli}(x_i \mid \theta) + \log p_{Beta}(\theta \mid \alpha, \beta)\end{aligned}$$

Calculate partial derivative to find the maximum:

$$\begin{aligned}\frac{\partial}{\partial\theta}\mathcal{LL}(\theta \mid X) &= \frac{\partial}{\partial\theta}\sum_{i=1}^{N} \log p_{Bernoulli}(x_i \mid \theta) + \frac{\partial}{\partial\theta}\log p_{Beta}(\theta \mid \alpha, \beta) \\ &= \sum_{i=1}^{N} \frac{\partial}{\partial\theta}\log \theta^{x_i}(1-\theta)^{1-x_i} + \frac{\partial}{\partial\theta}\log(\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\cdot\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}) \\ &= \frac{1}{\theta}\sum_{i=1}^{N} x_i - \frac{1}{1-\theta}\sum_{i=1}^{N}(1-x_i) + \frac{\partial}{\partial\theta}\log\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\cdot\Gamma(\beta)} + \frac{\partial}{\partial\theta}\log\theta^{\alpha-1} + \frac{\partial}{\partial\theta}\log(1-\theta)^{\beta-1} \\ &= \frac{1}{\theta}\sum_{i=1}^{N} x_i - \frac{1}{1-\theta}\sum_{i=1}^{N}(1-x_i) + 0 + (\alpha-1)\frac{1}{\theta} + (\beta-1)\frac{1}{1-\theta}\end{aligned}$$

Let $\frac{\partial}{\partial\theta}\mathcal{LL}(\theta \mid X) = 0$*:*

$$\begin{aligned}&\frac{1}{\theta}\sum_{i=1}^{N} x_i - \frac{1}{1-\theta}\sum_{i=1}^{N}(1-x_i) + 0 + (\alpha-1)\frac{1}{\theta} + (\beta-1)\frac{1}{1-\theta} = 0 \\ &\theta(\sum_{i=1}^{N}(1-x_i) + (\beta-1)) = (1-\theta)(\sum_{i=1}^{N} x_i + (\alpha-1)) \\ &\theta(\sum_{i=1}^{N}(1-x_i) + \sum_{i=1}^{N}(x_i) + \alpha + \beta - 2) = \sum_{i=1}^{N} x_i + \alpha - 1 \\ &(N + \alpha + \beta - 2) = \sum_{i=1}^{N} x_i + \alpha - 1 \\ &\theta = \frac{\sum_{i=1}^{N} x_i + \alpha - 1}{N + \alpha + \beta - 2} \\ &\theta = \frac{N_{positive} + \alpha - 1}{N + \alpha + \beta - 2}\end{aligned}$$

Thus we know:

$$\hat{\theta}_{MAP} = \frac{N_{positive} + \alpha - 1}{N + \alpha + \beta - 2}$$

gives us the MAP estimation on data set $\mathcal{D}$, where N is the total number of observation in the data set, and $N_{positive}$ is the total number of observation with positive feature ($x = 1$) in the data set.

Exercise 3.3.2 Multinomial Distribution

Suppose the likelihood between $\mathbf{X}$ and $\mathbf{y}$ in the data set obeys Multinomial Distribution given as:

$$p_{Multinomial}(x_1, x_2, \cdots, x_N \mid \theta_1, \theta_2, \cdots, \theta_N) = (\prod_{i=1}^{N} \theta_i^{x_i})$$

And given Dirichlet Distribution as its prior:

$$p_{Dirichlet}(\theta_1, \theta_2, \cdots, \theta_N \mid \alpha_1, \alpha_2, \cdots, \alpha_N) = \frac{\Gamma(\sum_{i=1}^{N} \alpha_i)}{\prod_{i=1}^{N} \Gamma(\alpha_i)} \prod_{j=1}^{N} \theta_j^{\alpha_j - 1}$$

Where each α_i is a constant.

Derive maximum a-posteriori estimation for each parameter θ_i.

Solution 3.3.2

Multiply likelihood function $\mathcal{L}$ and prior function.

$$\begin{aligned}
\mathcal{L}(\theta \mid X) &\propto p_{Multinomial}(x_1, x_2, \cdots, x_N \mid \theta_1, \theta_2, \cdots, \theta_N) \cdot p_{Dirichlet}(\theta_1, \theta_2, \cdots, \theta_N \mid \alpha_1, \alpha_2, \cdots, \alpha_N) \\
&= (\prod_{i=1}^{N} \theta_i^{x_i}) \cdot \frac{\Gamma(\sum_{j=1}^{N} \alpha_j)}{\prod_{j=1}^{N} \Gamma(\alpha_j)} \prod_{i=1}^{N} \theta_i^{\alpha_i - 1} \\
&= C_{const} \cdot \prod_{i=1}^{N} \theta_i^{x_i} \cdot \theta_i^{\alpha_i - 1} \\
&= C_{const} \cdot \prod_{i=1}^{N} \theta_i^{x_i + \alpha_i - 1}
\end{aligned}$$

Calculate the log-likelihood:

$$\begin{aligned}
\mathcal{LL}(\theta \mid X) &= \log(C_{const} \cdot \prod_{i=1}^{N} \theta_i^{x_i + \alpha_i - 1}) \\
&= C_{const} \sum_{i=1}^{N} \log \theta_i^{x_i + \alpha_i - 1} \\
&= C_{const} \sum_{i=1}^{N} (x_i + \alpha_i - 1) \log \theta_i
\end{aligned}$$

Where we have a constraint of

$$\sum_{i=1}^{N} \theta_i = 1$$

We can see the constant coefficient C_{const} does not influence the value of the log-likelihood function, so we may wipe it out.

Then, we will want to add a Lagrange multiplier λ to impose the constraint on the log-likelihood:

$$\mathcal{LL}(\theta \mid X) = \sum_{i=1}^{N} (x_i + \alpha_i - 1) \log \theta_i - \lambda (\sum_{i=1}^{N} \theta_i - 1)$$

Calculate partial derivative to find maximum:

$$\begin{aligned} \frac{\partial}{\partial \theta_i} \mathcal{LL}(\theta \mid X) &= \frac{\partial}{\partial \theta_i} (\sum_{i=1}^{N} (x_i + \alpha_i - 1) \log \theta_i) - \frac{\partial}{\partial \theta_i} (\lambda (\sum_{i=1}^{N} \theta_i - 1)) \\ &= \frac{x_i + \alpha_i - 1}{\theta_i} - \lambda \end{aligned}$$

Let $\frac{\partial}{\partial \theta_i} \mathcal{LL}(\theta \mid X) = 0$:

$$\begin{aligned} \frac{\partial}{\partial \theta_i} \mathcal{LL} &= \frac{x_i + \alpha_i - 1}{\theta_i} - \lambda = 0 \\ \theta_i &= \frac{x_i + \alpha_i - 1}{\lambda} \end{aligned}$$

We then may calculate λ with the above constraint

$$\begin{aligned} \sum_{i=1}^{N} \theta_i &= 1 \\ \sum_{i=1}^{N} (\frac{x_i + \alpha_i - 1}{\lambda}) &= 1 \\ \frac{\sum_{i=1}^{N} (x_i + \alpha_i - 1)}{\lambda} &= 1 \\ \lambda &= \sum_{i=1}^{N} (x_i + \alpha_i - 1) \end{aligned}$$

Substitute λ, we can get

$$\hat{\theta}_i = \frac{x_i + \alpha_i - 1}{\sum_{i=1}^{N}(x_i + \alpha_i - 1)}$$

And it is the MAP estimation for each parameter θ_i.

4 Optimizer

With the above estimation method, we may solve the optimization problem, and directly calculate the optimal parameters.

However, in many cases of machine learning, we are not directly calculating the optimal parameters. Instead, we are using some algorithm to approximate the optimal parameters, instead of calculating them directly. The algorithms we use here are called Optimizer.

Some of reasons why we are doing this are:

- It costs too much computational resources (including time and space) to find and calculate the optimal parameters directly.
- It is hard to directly find the optimal parameters out of their expression by mathematical derivations for some algorithm, like Deep Neural Network.
- More discussions: `https://stats.stackexchange.com/questions/278755/why-use-gradient-descent-for-linear-regression-when-a-closed-form-math-solution`

While using an algorithm to approximate the optimal parameters is simple, usually approximates well and applies to most machine learning algorithms that needs to do parameter estimation.

The common algorithms that we use to do the approximation to the optimal parameters including Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam).

4.1 Stochastic Gradient Descent (SGD)

For Stochastic Gradient Descent (SGD), we start form some imperfect parameters w and update them by:

$$w = w - \eta \nabla \mathcal{L}(w) = w - \frac{\eta}{n} \sum_{i=1}^{n} \nabla \mathcal{L}_i(w)$$

Where:

- w is each of our weights in $\mathbf{w}$.

- $\mathcal{L}(w)$ is our loss function. $\nabla\mathcal{L}(w)$ is the gradient between our current prediction and true label.
- η is the step size we go in each iteration. It is also called learning rate.
- n is the number of observation we consider to optimize the parameters in each iteration.

We may infer that in each iteration, it is trying to approximate a little bit to the optimal value by shrinking the loss between current prediction and the true label, and after a number of iterations, we may get to a place where we are very close to the optimal parameter $\mathbf{w}^*$

The algorithm of SGD is basically running a for-loop with the above update operation until the loss between our prediction and the true label could small enough.

A later proposed version of SGD is SGD with momentum (SGDM), which the update becomes:

$$\Delta w = \alpha\Delta w - \eta\nabla\mathcal{L}(w)$$
$$w = w + \Delta w$$

Where:

- α is a forgetting factor between 0 and 1.
- $\nabla\mathcal{L}(w)$ is the gradient of w on the loss function.
- η is the step size as we have in original SGD.

By doing this, the optimizer tends to remember the mainstream of the previous updating directions, and thus less affected by the random special cases in the data set.

You may see SGD with Momentum is SGD with the influence of the previous gradients.

4.2 Adaptive Moment Estimation (Adam)

When you are using SGD, you may find that the learning rate η is a constant. That looks not so smart, since in the beginning of the optimization, you could be far away from the optimal point and approximating the optimal point with tiny steps could take too much time.

Thus, optimizer with adaptive learning rate are proposed, in order to accelerate the process of approximating the optimal parameters. One of the representative adaptive optimizers is Adaptive Moment Estimation (Adam). It modifies the learning rate for each of the parameters in each step of the optimization process.

The updates in each iteration becomes:

$$w^{(t+1)} = w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon}$$

Where:

- $w^{(t+1)}$ is the parameter in step $t+1$, $w^{(t)}$ is the parameter in step t.
- η is a constant learning rate that we still keep as we do in SGD.
- ϵ is a tiny factor preventing division by 0.
- $\hat{m}_w$ and $\hat{v}_w$ are:

$$\hat{m}_w = \frac{m_w^{(t+1)}}{1 - \beta_1^{t+1}}$$
$$\hat{v}_w = \frac{v_w^{(t+1)}}{1 - \beta_2^{t+1}}$$

- β_1 is the forgetting factor for gradients between 0 and 1. When iteration t goes big, the update in each iteration $\hat{m}_w$ becomes small.
- β_2 is the forgetting factor for second moments of gradients, or say square of the gradients between 0 and 1. Similar to β_1, when iteration t goes big, the update in each iteration $\hat{v}_w$ becomes small.
- m_w and v_w is updated by:

$$m_w^{(t+1)} = \beta_1 m_w^{(t)} + (1 - \beta_1) \nabla_w \mathcal{L}^{(t)}$$

$$v_w^{(t+1)} = \beta_2 v_w^{(t)} + (1 - \beta_2) (\nabla_w \mathcal{L}^{(t)})^2$$

You may notice Adam is a combination of SGD with Momentum which is SGD with the influence of the previous gradients, and SGD with the influence of the second moment (square) of the previous gradients.

Adam is controversy. It looks smarter than doing the SGD and in many cases accelerate the process of the optimization. However, some researchers claim it is not helping us on finding the best parameters at all. You may read more on this topic in the paper *The Marginal Value of Adaptive Gradient Methods in Machine Learning* (`https://arxiv.org/abs/1705.08292`).