2a: Cross Entropy, Softmax, Weight Decay and Momentum

Week 2: Overview

In this lesson, we will discuss cross entropy and softmax, which are used for classification tasks as alternatives to the sum squared error loss function. Finally, we will describe weight decay, momentum, and adaptive moment estimation (Adam). Along the way, we will need to introduce the mathematical concepts of Maximum Likelihood Estimation (MLE) and Maximum A Posteriori (MAP) Estimation.

Then we will look at the basic structure and components of a typical PyTorch program, and run some simple examples. We will also learn how to analyze the hidden unit dynamics of neural networks.

Weekly learning outcomes

By the end of this week, you will be able to:

- explain and compute cross entropy, softmax
- explain weight decay, momentum
- code simple PyTorch operations
- analyze the geometry of hidden unit activations in neural networks

Cross Entropy and Softmax

Loss Functions

In Week 1 we introduced the **sum squared error** (**SSE**) loss function, which is suitable for function approximation tasks.

$$E = \frac{1}{2}\sum_i (t_i - z_i)^2$$

However, for **binary classification** tasks, where the target output is either zero or one, it may be more logical to use the **cross entropy** error:

$$E = \sum_i (-t_i \log(z_i) - (1 - t_i) \log(1 - z_i))$$

In order to explain the motivation for both of these loss functions, we need to introduce the mathematical concept of Maximum Likelihood.

Maximum Likelihood

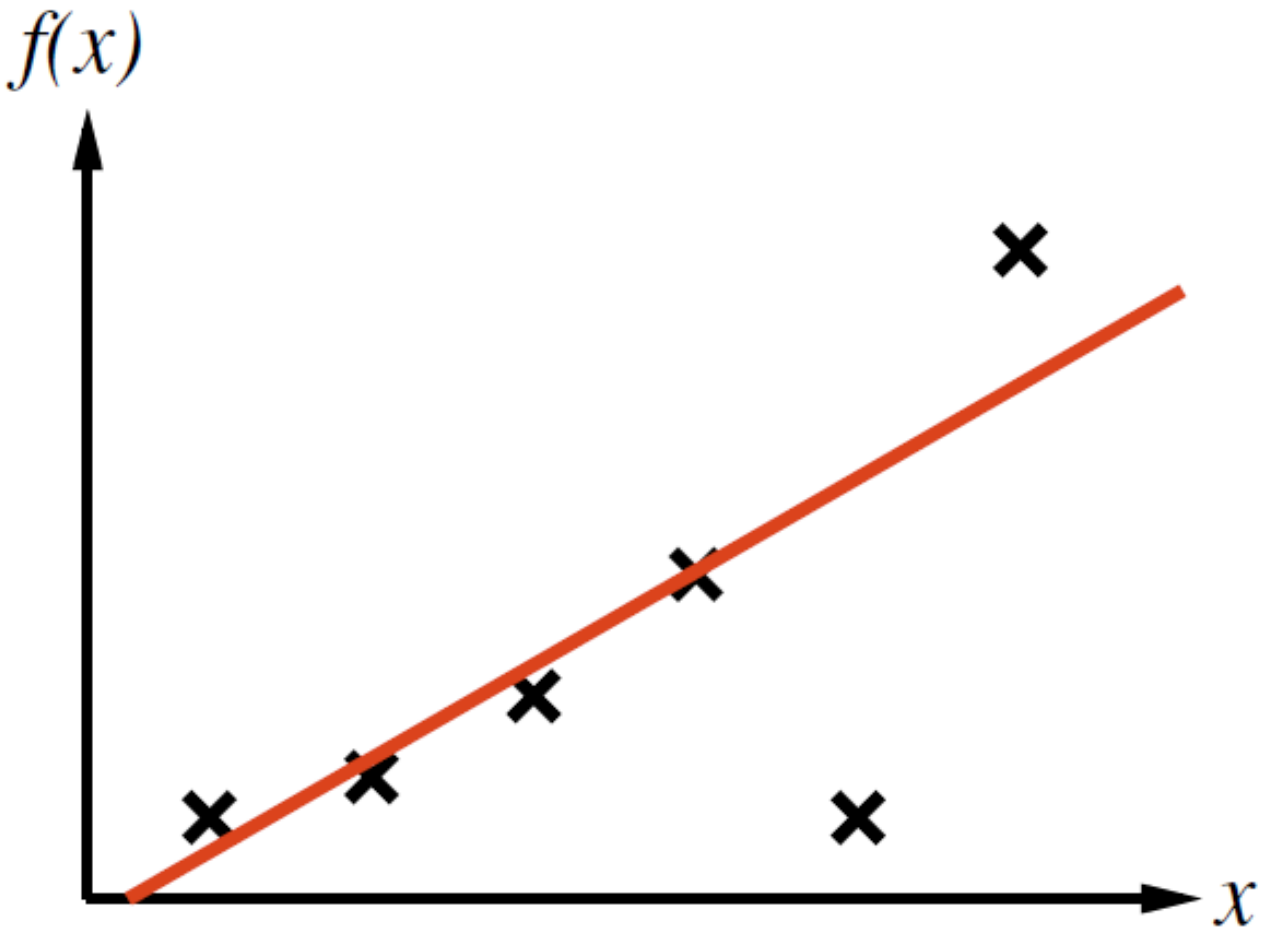
Let H be a class of **hypotheses** for predicting certain **data** D.

Let $\mathrm{Prob}(\mathrm{D}|h)$ be the probability of data D being generated under hypothesis $h \in \mathrm{H}$.

The logarithm of this probability, $\log \mathrm{Prob}(\mathrm{D}|h)$ is called the **likelihood**.

The **Maximum Likelihood Principle** states that we should choose $h \in \mathrm{H}$ which maximizes this likelihood, i.e. maximizes $\mathrm{Prob}(\mathrm{D}|h)$ or, equivalently, maximizes $\log \mathrm{Prob}(\mathrm{D}|h)$.

In our case, the data D consist of a target value t_i for each set of input features x_i in a Supervised Learning task, and we can think of each hypothesis h as a function $f()$ determined by a neural network with specified weights or, to give a simpler example, $f()$ could be a straight line with a specified slope and y-intercept.



Derivation of Least Squares

As previously mentioned, noise in the data is often caused by an accumulation of small errors due to factors which are not captured by the model. The Central Limit Theorem tells us that when a large number of independent random variables are added together, the combined error is well approximated by a Gaussian distribution.

In order to accommodate this kind of noise, let's suppose that our data $\mathrm{D} = \{t_i\}$ are generated by a linear function $f()$ plus noise generated from a Gaussian distribution with mean zero and standard deviation σ. Then

$$\mathrm{Prob}(\,\mathrm{D} \mid h\,) = \mathrm{Prob}(\{t_i\} \mid f\,) = \prod_i \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(t_i - f(x_i))^2}$$

$$\log \mathrm{Prob}(\{t_i\} \mid f\,) = \sum_i -\frac{1}{2\sigma^2}\left(t_i - f\,(x_i)\right)^2 - \log(\sigma) - \frac{1}{2}\log(2\pi)$$

$$\begin{aligned} f_{ML} &= \mathrm{argmax}_{f \in \mathrm{H}} \log \mathrm{Prob}(t \mid f) \\ &= \mathrm{argmin}_{f \in \mathrm{H}} \frac{1}{2} \sum_i (t_i - f(x_i))^2 \end{aligned}$$

It is interesting to note that we do not need to know the value of σ.

Derivation of Cross Entropy

For binary classification tasks, the target value t_i is either 0 or 1. It makes sense to interpret the output $f(x_i)$ of the neural network as the **probability** of the true value being 1, i.e.

$$\begin{aligned}\text{Prob}(\,1 \mid f(x_i)) &= f(x_i)\\ \text{Prob}(\,0 \mid f(x_i)) &= 1 - f(x_i)\\ \text{Prob}(\,t_i \mid f(x_i)) &= f(x_i)^{t_i}(1 - f(x_i)^{1-t_i}\end{aligned}$$

Then

$$\log \text{Prob}(\{t_i\} \mid f) = \sum_i (\,t_i \log f(x_i) + (1 - t_i) \log(1 - f(x_i)))$$

So, according to the Maximum Likelihood principle, we need to maximize the expression on the right hand side (or minimize its negative).

Cross Entropy loss is often used in combination with sigmoid activation at the output node, which guarantees that the output is strictly between 0 and 1, and also makes the backprop computations a bit simpler, as follows:

$$E = \sum_i (-t_i \log(z_i) - (1 - t_i) \log(1 - z_i))$$

$$\frac{\partial E}{\partial z_i} = -\frac{t_i}{z_i} + \frac{1 - t_i}{1 - z_i} = \frac{z_i - t_i}{z_i(1 - z_i)}$$

$$\text{If} \quad z_i = \frac{1}{1 + e^{-s_i}}, \qquad \frac{\partial E}{\partial s_i} = \frac{\partial E}{\partial z_i}\frac{\partial z_i}{\partial s_i} = z_i - t_i$$

SSE and cross entropy behave a bit differently when it comes to outliers. SSE is more likely to allow outliers in the training set to be misclassified, because the contribution to the loss function for each item is bounded between 0 and 1. Cross Entropy is more likely to keep outliers correctly classified, because the loss function grows logarithmically as the distance between the target and actual value approaches 1. For this reason, Cross Entropy works particularly well for classification tasks that are **unbalanced** in the sense of negative items vastly outnumbering positive ones (or vice versa).

Log Softmax

Some Supervised Learning tasks require data to be classified into more than two classes. If the number of classes is N and we have a neural network with N outputs $z_i, \ldots z_N$, we can make the assumption that the network's estimate for the probability of class j is proportional to $\exp(z_j)$.

Because the probabilities must add up to 1, we need to normalize by dividing by their sum:

$$\mathrm{Prob}(i) = \frac{\exp\left(z_i\right)}{\sum_{j=1}^{N} \exp\left(z_j\right)}$$

$$\log \mathrm{Prob}(i) = z_i - \log \sum_{j=1}^{N} \exp\left(z_j\right)$$

If the correct class is k, we can treat $-\log \mathrm{Prob}(k)$ as our loss function, and the gradient is

$$\frac{d}{dz_i} \log \mathrm{Prob}(k) = \delta_{ik} - \frac{\exp(z_i)}{\sum_{j=1}^{N} \exp\left(z_j\right)} = \delta_{ik} - \mathrm{Prob}(i)$$

where δ_{ik} is the Kronecker delta. The first term pushes up the correct class $i = k$ in proportion to the distance of its assigned probability from 1, while the second term pushes down the incorrect classes $i \neq k$ in proportion to the probabilities assigned to them.

Softmax compared to Boltzmann Distribution and Sigmoid

If you have studied mathematics or physics, you may be interested to know that Softmax is related to the Boltzmann Distribution, with the negative of output z_i playing the role of the "energy" for "state" i. The familiar Sigmoid function can also be seen as a special case of Softmax, with two classes and one output, as follows: Consider a simplified case where there is a choice between two classes, Class 0 and Class 1. We consider the output z of the network to be associated with Class 1, and we imagine a fixed "output" for Class 0 which is always equal to zero. In this case, the softmax becomes:

$$\text{Prob}(1) = \frac{e^z}{e^z + e^0} = \frac{1}{1 + e^{-z}}$$

Optional video

Further Reading

Textbook Deep Learning (Goodfellow, Bengio, Courville, 2016):

- Maximum Likelihood (5.5)
- Cross Entropy (3.13)
- Softmax (6.2.2)
- Derivative rules

Exercise: Softmax and Backpropagation

Recall that the formula for Softmax is:

$$\text{Prob}(i) = \frac{\exp(z_i)}{\sum_{j=1}^{N} \exp(z_j)}$$

$$\log \text{Prob}(i) = z_i - \log \sum_{j=1}^{N} \exp(z_j)$$

Consider a classification task with three classes 1, 2, 3. Suppose a particular input is presented, producing outputs:

$$z_1 = 1, \quad z_2 = 2, \quad z_3 = 3$$

and that the correct class is 2.

Question 1 *Submitted Sep 7th 2022 at 9:18:24 pm*

Compute each of the following, to two decimal places:

$$\text{Prob}(1)$$
$$\text{Prob}(2)$$
$$\text{Prob}(3)$$

Solution

$$\text{Prob}(1) = 0.09$$
$$\text{Prob}(2) = 0.24$$
$$\text{Prob}(3) = 0.67$$

Explanation

$$\text{Prob}(1) = e^1/(e^1 + e^2 + e^3) = \ \ 2.718/30.193 = 0.09$$
$$\text{Prob}(2) = e^2/(e^1 + e^2 + e^3) = \ \ 7.389/30.193 = 0.24$$
$$\text{Prob}(3) = e^3/(e^1 + e^2 + e^3) = 20.086/30.193 = 0.67$$

Question 2 *Submitted Sep 7th 2022 at 9:19:47 pm*

Compute each of the following, to two decimal places:

$$\mathrm{d}(\log \mathrm{Prob}(2))/\mathrm{d}z_1$$
$$\mathrm{d}(\log \mathrm{Prob}(2))/\mathrm{d}z_2$$
$$\mathrm{d}(\log \mathrm{Prob}(2))/\mathrm{d}z_3$$

Solution

$$\mathrm{d}(\log \mathrm{Prob}(2))/\mathrm{d}z_1 = -0.09$$
$$\mathrm{d}(\log \mathrm{Prob}(2))/\mathrm{d}z_2 = \ \ 0.76$$
$$\mathrm{d}(\log \mathrm{Prob}(2))/\mathrm{d}z_3 = -0.67$$

Note how the correct class (2) is pushed up, while the incorrect class with the highest activation (3) is pushed down the most.

Explanation

$$\mathrm{d}(\log \mathrm{Prob}(2))/\mathrm{d}z_1 = \mathrm{d}(z_2 - \log \sum_j \exp(z_j))/\mathrm{d}z_1 \ = \ -\exp(z_1)/\sum_j \exp(z_j) = -0.09$$
$$\mathrm{d}(\log \mathrm{Prob}(2))/\mathrm{d}z_2 = \mathrm{d}(z_2 - \log \sum_j \exp(z_j))/\mathrm{d}z_2 \ = 1 - \exp(z_2)/\sum_j \exp(z_j) = \ \ 0.76$$
$$\mathrm{d}(\log \mathrm{Prob}(2))/\mathrm{d}z_3 = \mathrm{d}(z_2 - \log \sum_j \exp(z_j))/\mathrm{d}z_3 \ = \ -\exp(z_3)/\sum_j \exp(z_j) = -0.67$$

Question 3 *Submitted Sep 7th 2022 at 9:20:48 pm*

Consider a degenerate case of supervised learning where the training set consists of just a single input, repeated 100 times. In 80 of the 100 cases, the target output value is 1; in the other 20, it is 0. What will a back-propagation neural network predict for this example, assuming that it has been trained and reaches a global minimum? Does it make a difference whether the loss function is sum squared error or cross entropy? (Hint: to find the global minimum, differentiate the loss function and set the derivative to zero.)

Solution

The output will converge to 0.8, for both SSE and cross entropy loss.

Explanation

When sum squared error is minimized, we have

$$
\begin{aligned}
E &= 80 \times \frac{1}{2}(z-1)^2 + 20 \times \frac{1}{2}(z-0)^2 \\
\frac{dE}{dz} &= 80(z-1) + 20(z-0) \\
&= 100\,z - 80 \\
&= 0 \;\text{ when }\; z = 0.8
\end{aligned}
$$

When cross entropy is minimized, we have

$$
\begin{aligned}
E &= -80 \times \log(z) - 20 \times \log(1-z) \\
\frac{dE}{dz} &= -\frac{80}{z} + \frac{20}{1-z} \\
&= \frac{-80(1-z) + 20\,z}{z(1-z)} \\
&= \frac{100\,z - 80}{z(1-z)} \\
&= 0 \;\text{ when }\; z = 0.8, \;\text{ as before.}
\end{aligned}
$$

Weight Decay and Momentum

Weight Decay

Sometimes a penalty term is added to the loss function which encourages the neural network weights w_j to remain small:

$$E = \frac{1}{2}\sum_i (z_i - t_i)^2 + \frac{\lambda}{2}\sum_j w_j^2$$

(Note: the sum squared error term may be replaced with cross entropy or softmax).

This additional loss term prevents the weights from "saturating" to very high values. It is sometimes referred to as "elastic weights" because it simulates a force on each weight as if there were a spring pulling it back towards the origin according to Hooke's Law. The scaling factor λ needs to be determined from experience, or empirically.

In order to explain the theoretical justification for Weight Decay, we need to introduce Baysian Inference and Maximum A Posteriori (MAP) estimation.

Bayesian Inference and MAP estimation

Recall the Maximum Likelihood principle which selects hypothesis $h \in \mathrm{H}$ for which $\mathrm{Prob}(\mathrm{D} \mid h)$ is maximal.

Bayesian Inference instead seeks to maximize $\mathrm{Prob}(h \mid \mathrm{D})$ i.e. the probability that hypothesis h is correct, given that data D have been observed.

According to Bayes' Theorem:

$$P(h \mid \mathrm{D})P(\mathrm{D}) = P(D \mid h)P(h)$$
$$P(h \mid D) = \frac{P(\mathrm{D} \mid h)P(h)}{P(D)}$$

We do **not** need to know $\mathrm{Prob}(\mathrm{D})$ in order to maximize this expression over h. However, we **do** need to be able to estimate $\mathrm{Prob}(h)$ which is called the **prior** probability of h (i.e. our estimate of the probability **before** the data have been observed). $\mathrm{Prob}(h \mid \mathrm{D})$ is called the **posterior** probability because it is our estimate **after** the data have been observed. For this reason, maximizing $\mathrm{Prob}(h \mid \mathrm{D})$ in this context is sometimes called Maximum A Posteriori or **MAP** estimation.

Note also that $\mathrm{Prob}(h)$ must be a well-defined probability in the sense that its integral over H must be finite and not infinite.

Weight Decay as MAP Estimation

We assume a Gaussian prior distribution for the weights, i.e.

$$P(w) = \prod_j \frac{1}{\sqrt{2\pi}\sigma_0} e^{-w_j^2/2\sigma_0^2}$$

$$P(w \mid t) = \frac{P(t \mid w)P(w)}{P(t)} = \frac{1}{P(t)} \prod_i \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(z_i - t_i)^2} \prod_j \frac{1}{\sqrt{2\pi}\sigma_0} e^{-w_j^2/2\sigma_0^2}$$

$$\log P(w \mid t) = -\frac{1}{2\sigma^2} \sum_i (z_i - t_i)^2 - \frac{1}{2\sigma_0^2} \sum_j w_j^2 + \text{ constant}$$

$$\begin{aligned} w_{\text{MAP}} &= \text{argmax}_{w \in H} \log P(w \mid t) \\ &= \text{argmin}_{w \in H} \left(\frac{1}{2} \sum_i (z_i - t_i)^2 + \frac{\lambda}{2} \sum_j w_j^2 \right) \end{aligned}$$

where $\lambda = \sigma^2 / \sigma_0^2$

Optional video

Momentum

It is often helpful to maintain a running average of the gradient for each weight, and use this running

average to update that weight:

$$\delta w \leftarrow \alpha \delta w - \eta \frac{\partial E}{\partial w}$$
$$w \leftarrow w + \delta w$$

The parameter $\alpha \in [0, 1)$ is called the **Momentum**.

This is helpful in two situations. Firstly, if the weights travel through a flat region in the landscape, momentum will help to speed up the learning in this region. Secondly, if the landscape is shaped like a "rain gutter", weights will tend to oscillate without much improvement. Momentum will theoretically dampen the sideways oscillations but amplify the downhill motion by a factor of $\frac{1}{1-\alpha}$.

When momentum is introduced, we generally reduce the learning rate at the same time, in order to compensate for this implicit factor of $\frac{1}{1-\alpha}$.

Adaptive Moment Estimation (Adam)

Adaptive Moment Estimation or **Adam** maintains a running average of the gradients (m_t) and the squared gradients (v_t) for each weight in the network (Kingma & Ba, 2015).

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

To speed up the training in the early stages, compensating for the fact that m_t, g_t are initialized to zero, we rescale as follows:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Finally, each parameter is adjusted according to:

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t$$

Adam has been found to work well across a wide variety of task domains.

As with momentum, when switching from SGD to Adam we often need to reduce the learning rate η in order to compensate for the factor of $\frac{1}{\sqrt{\hat{v}_t} + \varepsilon}$.

Second Order Methods

For smaller networks, optimisation methods such as Conjugate Gradients or Natural Gradients are

sometimes used, which compute the second derivative of the loss function with respect to every pair of weights in the network. However, these methods are generally not practical in the context of deep learning, because the computation time is proportional to the square of the number of weights, which may be in the millions or even billions.

Adam is seen as a very cost effective method which, in practice, provides a similar benefit to second order methods, but with far less computation.

Optional video

References

Kingma, D.P., & Ba, J., 2015. Adam: A method for stochastic optimization, *International Conference on Learning Representations* (poster).

Further Reading

https://ruder.io/optimizing-gradient-descent/index.html

Textbook Deep Learning (Goodfellow, Bengio, Courville, 2016):

- Weight Decay (5.2.2) and MAP Estimation (5.6.1)
- Momentum (8.3) and Adam (8.5)

Revision 3: Backprop Variations

This is a revision quiz to test your understanding of the material from Week 2 on Backprop Variations.

You must attempt to answer each question yourself, before looking at the sample answer.

Question 1 *Submitted Sep 7th 2022 at 9:25:02 pm*

Explain the difference between the following paradigms, in terms of what is presented to the system, and what it aims to achieve:

- Supervised Learning
- Reinforcement Learning
- Unsupervised Learning

Solution

Supervised Learning: The system is presented with training items consisting of an input and a target output. The aim is to predict the output, given the input (for the training set as well as an unseen test set).

Reinforcement Learning: The system chooses actions in a simulated environment, observing its state and receiving rewards along the way. The aim is to maximize the cumulative reward.

Unsupervised Learning: The system is presented with training items consisting of only an input (no target value). The aim is to extract hidden features or other structure from these data.

Question 2 *Submitted Sep 7th 2022 at 9:26:21 pm*

Explain what is meant by Overfitting in neural networks, and list four different methods for avoiding it.

Solution

Overfitting is where the training set error continues to reduce, but the test set error stalls or increases. This can be avoided by

- limiting the number of neurons or connections in the network
- early stopping, with a validation set
- dropout
- weight decay (this can avoid overfitting by limiting the size of the weights)

Question 3 *Submitted Sep 7th 2022 at 9:27:24 pm*

Explain how Dropout is used for neural networks, in both the training and testing phase.

Solution

During each minibatch of training, a fixed percentage (usually, one half) of nodes are chosen to be inactive. In the testing phase, all nodes are active but the activation of each node is multiplied by the same percentage that was used in training.

Question 4 *Submitted Sep 7th 2022 at 9:28:32 pm*

Write the formulas for these Loss functions: Squared Error, Cross Entropy, Softmax, Weight Decay (remember to define any variables you use)

Solution

Assume z_i is the actual output, t_i is the target output and w_j are the weights.

Squared Error: $E = \frac{1}{2}\sum_i (z_i - t_i)^2$

Cross Entropy: $E = \sum_i (-t_i \log z_i - (1 - t_i)\log(1 - z_i))$

Softmax: $E = -(z_i - \log \sum_j \exp(z_j))$, where i is the correct class.

Weight Decay: $E = \frac{1}{2}\sum_j w_j^2$

Question 5 *Submitted Sep 7th 2022 at 9:29:45 pm*

In the context of Supervised Learning, explain the difference between Maximum Likelihood estimation and Bayesian Inference.

In Maximum Likelihood estimation, the hypothesis $h \in H$ is chosen which maximises the conditional probability $P(D \mid h)$ of the observed data D, conditioned on h.

In Bayesian Inference, the hypothesis $h \in H$ is chosen which maximizes $P(D \mid h)P(h)$, where $P(h)$ is the prior probability of h.

Question 6 *Submitted Sep 7th 2022 at 9:30:47 pm*

Briefly explain the concept of Momentum, as an enhancement for Gradient Descent.

Solution

A running average of the differentials for each weight is maintained and used to update the weights as follows:

$$\delta w \ = \ \alpha \delta w \ - \eta \frac{dE}{dw}$$

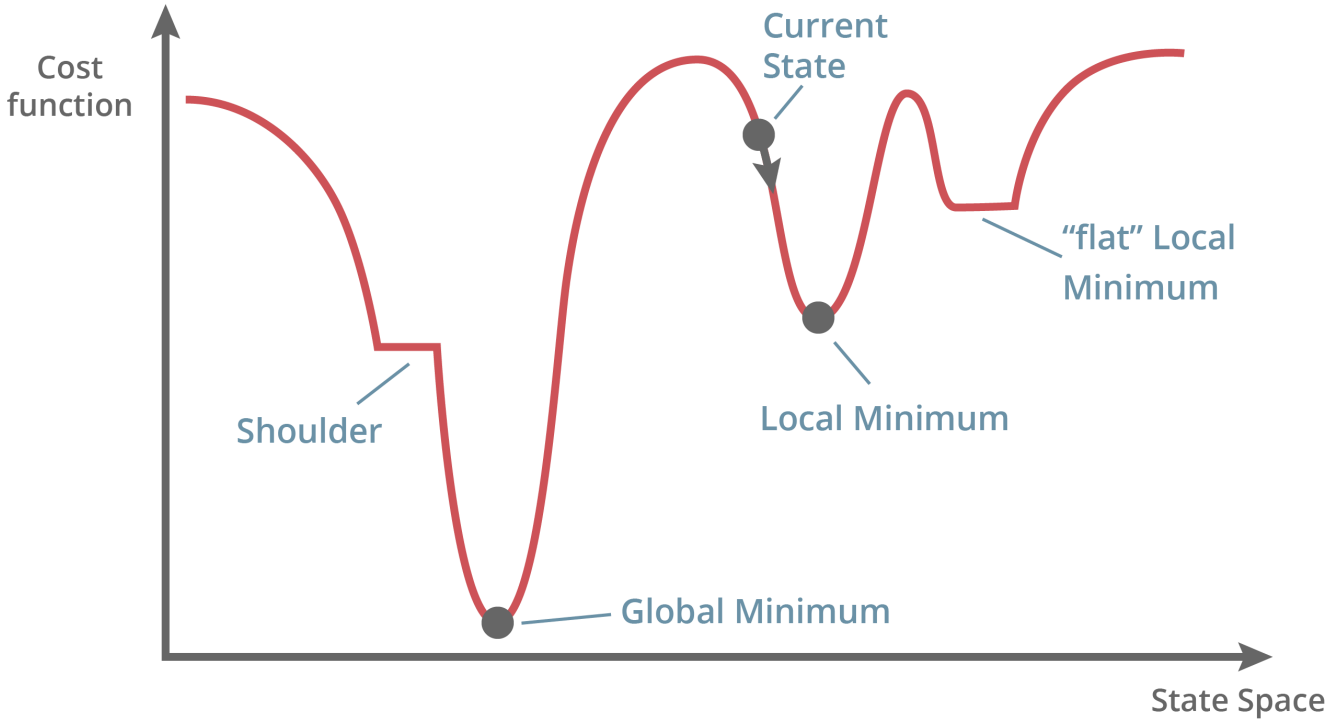
$$w \ = \ w \ + \ \delta w$$

The constant α with $0 \leq \alpha < 1$ is called the momentum.

Coding Exercise - Gradient Descent with NumPy

Objective

In this exercise, you will learn how to use gradient descent to solve a linear regression problem.



Instructions

Complete the parts of the code marked as "**TODO".**

To run the cell you can press Ctrl-Enter or hit the "Play" button at the top.