

These lecture notes are based on notes originally written by Nikhil Sharma and on the textbook *Artificial Intelligence: A Modern Approach*.

Last updated: November 8, 2022

In this note we will cover how to optimize functions using the gradient descent algorithm. We will also learn about a classification method called logistic regression and how it can be extended to multi-class classification. This will motivate our further discussion on neural networks and backpropagation.

## Optimization

We saw in the previous note in the linear regression method that we can derive a closed form solution for the optimal weights by just differentiating the loss function and setting the gradient equal to zero. In general though, a closed form solution may not exist for a given objective function. In cases like that we have to use **gradient-based methods** to find the optimal weights. The idea behind this is that the gradient points towards the direction of steepest increase of the objective. We maximize a function by moving towards the steepest **ascent**, and we minimize a function by moving towards the steepest **descent** direction.

**Gradient ascent** is used if the objective is a function which we try to maximize.

```
Randomly initialize  $\mathbf{w}$ 
while  $\mathbf{w}$  not converged do
  |  $\mathbf{w} \leftarrow \mathbf{w} + \alpha \nabla_{\mathbf{w}} f(\mathbf{w})$ 
end
```

### Algorithm 1: Gradient ascent

**Gradient descent** is used if the objective is a loss function that we are trying to minimize. Notice that this only differs from gradient ascent in that we follow the opposite direction of the gradient.

```
Randomly initialize  $\mathbf{w}$ 
while  $\mathbf{w}$  not converged do
  |  $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} f(\mathbf{w})$ 
end
```

### Algorithm 2: Gradient descent

At the beginning, we initialize the weights randomly. We denote the learning rate, which captures the size of the steps we make towards the gradient direction, with  $\alpha$ . For most functions in the machine learning world it is hard to come up with an optimal value for the learning rate. In reality, we want a learning rate that is large enough so that we move fast towards the correct direction but at the same time small enough so that the method does not diverge. A typical approach in machine learning literature is to start gradient descent with a relatively large learning rate and reduce the learning rate as the number of iterations increases (**learning rate decay**).

If our dataset has a large number of  $n$  data points then computing the gradient as above in each iteration of the gradient descent algorithm might be too computationally intensive. As such, approaches like stochastic and batch gradient descent have been proposed. In **stochastic gradient descent** at each iteration of the

algorithm we use only one data point to compute the gradient. That one data point is each time randomly sampled from the dataset. Given that we only use one data point to estimate the gradient, stochastic gradient descent can lead to noisy gradients and thus make convergence a bit harder. **Mini-batch gradient descent** is a compromise between stochastic and the ordinary gradient descent algorithm as it uses a batch of size  $m$  of data points each time to compute the gradients. The batch size  $m$  is a user specified parameter.

Let's see an example of gradient descent on a model we've seen before—linear regression. Recall that in linear regression, we defined our loss function as

$$Loss(h_{\mathbf{w}}) = \frac{1}{2} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$$

Linear regression has a celebrated closed form solution  $\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$ , which we saw in the last note. However, we could have also chosen to solve for the optimal weights by running gradient descent. We'd calculate the gradient of our loss function as

$$\nabla_{\mathbf{w}} Loss(h_{\mathbf{w}}) = -\mathbf{X}^T \mathbf{y} + \mathbf{X}^T \mathbf{X} \mathbf{w}$$

Then, we use this gradient to write out the gradient descent algorithm for linear regression:

```
Randomly initialize  $\mathbf{w}$ 
while  $\mathbf{w}$  not converged do
  |  $\mathbf{w} \leftarrow \mathbf{w} - \alpha(-\mathbf{X}^T \mathbf{y} + \mathbf{X}^T \mathbf{X} \mathbf{w})$ 
end
```

**Algorithm 3:** Least squares gradient descent

It is a good exercise to create a linear regression problem and confirm that the closed form solution is the same as the converged solution you obtain from gradient descent.

## Logistic Regression

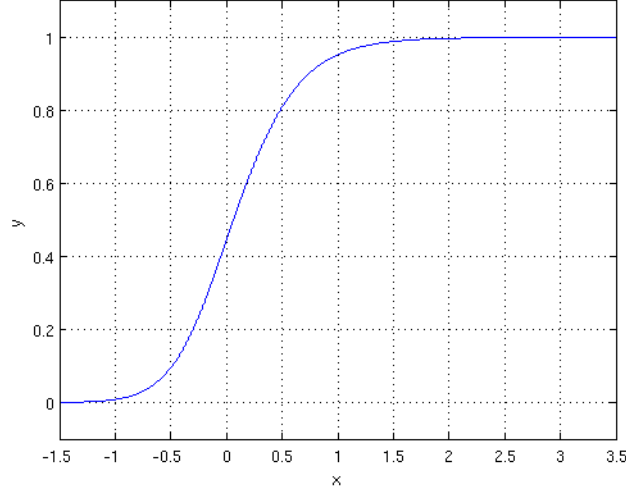
Let's again think about our previous method of linear regression. In it, we assumed that the output is a numeric real-valued number.

What if we instead want to predict a categorical variable? Logistic regression allows us to turn a linear combination of our input features into a probability using the logistic function:

$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}.$$

It is important to note that though logistic regression is named as regression, this is a misnomer. Logistic regression is used to solve classification problems, not regression problems.

The logistic function  $g(z) = \frac{1}{1+e^{-z}}$  is frequently used to model binary outputs. Note that the output of the function is always between 0 and 1, as seen in the following figure:



Intuitively, the logistic function models the probability of a data point belonging in class with label 1. The reason for that is that the output of the logistic function is bounded between 0 and 1, and we want our model to capture the probability of a feature having a specific label. For instance, after we have trained logistic regression, we obtain the output of the logistic function for a new data point. If the value of the output is greater than 0.5 we classify it with label 1 and we classify it with label 0 otherwise. More specifically, we model the probabilities as follows:

$$P(y = +1 | \mathbf{f}(\mathbf{x}); \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{f}(\mathbf{x})}}$$

and

$$P(y = -1 | \mathbf{f}(\mathbf{x}); \mathbf{w}) = 1 - \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{f}(\mathbf{x})}},$$

where we use  $\mathbf{f}(\mathbf{x})$  to denote the function (which often is the identity) of the feature vector  $\mathbf{x}$  and the semi-colon ; denotes that the probability is a function of the parameter weights  $\mathbf{w}$ .

A useful property to note is that the logistic function is that the derivative is  $g'(z) = g(z)(1 - g(z))$ .

How do we train the logistic regression model? The loss function for logistic regression is the  $L2$  loss  $Loss(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - h_{\mathbf{w}}(\mathbf{x}))^2$ . Since a closed form solution is not possible for the logistic regression, we estimate the unknown weights  $\mathbf{w}$  via gradient descent. For that we need to compute the gradient of the loss function using the chain rule of differentiation. The gradient of the loss function with respect to the weight of coordinate  $i$  is given by:

$$\frac{\partial}{\partial w_i} \frac{1}{2} (y - h_{\mathbf{w}}(\mathbf{x}))^2 = (y - h_{\mathbf{w}}(\mathbf{x})) \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(\mathbf{x})) = -(y - h_{\mathbf{w}}(\mathbf{x})) h_{\mathbf{w}}(\mathbf{x}) (1 - h_{\mathbf{w}}(\mathbf{x})) x_i,$$

where we used the fact that the gradient of the logistic function  $g(z) = \frac{1}{1 + e^{-z}}$  satisfies  $g'(z) = g(z)(1 - g(z))$ . We can then estimate the weights using gradient descent and then predict, as detailed above.

# Multi-Class Logistic Regression

In multi-class logistic regression, we want to classify data points into  $K$  distinct categories, rather than just two. Thus, we want to build a model that output estimates of the probabilities for a new data point to belong to each of the  $K$  possible categories. For that reason we use the **softmax function** in place of the logistic function, which models the probability of a new data point with features  $\mathbf{x}$  having label  $i$  as follows:

$$P(y = i | \mathbf{f}(\mathbf{x}); \mathbf{w}) = \frac{e^{\mathbf{w}_i^T \mathbf{f}(\mathbf{x})}}{\sum_{k=1}^K e^{\mathbf{w}_k^T \mathbf{f}(\mathbf{x})}}.$$

Note that these probability estimates add up to 1, so they constitute a valid probability distribution. We estimate the parameters  $\mathbf{w}$  to maximize the likelihood that we observed the data. Assume that we have observed  $n$  labelled data points  $(\mathbf{x}_i, y_i)$ . The likelihood, which is defined as the joint probability distribution of our samples, is denoted with  $\ell(\mathbf{w}_1, \dots, \mathbf{w}_K)$  and is given by:

$$\ell(\mathbf{w}_1, \dots, \mathbf{w}_K) = \prod_{i=1}^n P(y_i | \mathbf{f}(\mathbf{x}_i); \mathbf{w}).$$

To compute the values of the parameters  $\mathbf{w}_i$  that maximize the likelihood, we compute the gradient of the likelihood function with respect to each parameter, set it equal to zero, and solve for the unknown parameters. If a closed form solution is not possible, we compute the gradient of the likelihood and we use gradient ascent to obtain the optimal values.

A common trick we do to simplify these calculations is to first take the logarithm of the likelihood function which will break the product into summations and simplify the gradient calculations. We can do this because the logarithm is a strictly increasing function and the transformation will not affect the maximizers of the function.

For the likelihood function we need a way to express the probabilities  $P(y_i | \mathbf{f}(\mathbf{x}_i); \mathbf{w})$  in which  $y \in \{1, \dots, K\}$ . So for each data point  $i$ , we define  $K$  parameters  $t_{i,k}$ ,  $k = 1, \dots, K$  such that  $t_{i,k} = 1$  if  $y_i = k$  and 0 otherwise. Hence, we can now express the likelihood as follows:

$$\ell(\mathbf{w}_1, \dots, \mathbf{w}_K) = \prod_{i=1}^n \prod_{k=1}^K \left( \frac{e^{\mathbf{w}_k^T \mathbf{f}(\mathbf{x}_i)}}{\sum_{\ell=1}^K e^{\mathbf{w}_\ell^T \mathbf{f}(\mathbf{x}_i)}} \right)^{t_{i,k}}$$

and we also obtain for the log-likelihood that:

$$\log \ell(\mathbf{w}_1, \dots, \mathbf{w}_K) = \sum_{i=1}^n \sum_{k=1}^K t_{i,k} \log \left( \frac{e^{\mathbf{w}_k^T \mathbf{f}(\mathbf{x}_i)}}{\sum_{\ell=1}^K e^{\mathbf{w}_\ell^T \mathbf{f}(\mathbf{x}_i)}} \right)$$

Now that we have an expression for the objective we must estimate the  $\mathbf{w}_i$ s such that they maximize that objective.

In the example of the multi-class logistic regression the gradient with respect to  $\mathbf{w}_j$  is given by:

$$\nabla_{\mathbf{w}_j} \log \ell(\mathbf{w}) = \sum_{i=1}^n \nabla_{\mathbf{w}_j} \sum_{k=1}^K t_{i,k} \log \left( \frac{e^{\mathbf{w}_k^T \mathbf{f}(\mathbf{x}_i)}}{\sum_{\ell=1}^K e^{\mathbf{w}_\ell^T \mathbf{f}(\mathbf{x}_i)}} \right) = \sum_{i=1}^n \left( t_{i,j} - \frac{e^{\mathbf{w}_j^T \mathbf{f}(\mathbf{x}_i)}}{\sum_{\ell=1}^K e^{\mathbf{w}_\ell^T \mathbf{f}(\mathbf{x}_i)}} \right) \mathbf{f}(\mathbf{x}_i),$$

where we used the fact that  $\sum_k t_{i,k} = 1$ .