width=7cm,compat=1.8

IN3050 Mathgroup, Derivatives

Tobias Opsahl

March 15, 2023

UNIVERSITY OF OSLO

Loss Functions
Motivation

A lot of machine-learning models is based on *minimizing a loss-function* (or equivalently, maximizing a reward function).

Loss Functions
Motivation

A lot of machine-learning models is based on *minimizing a loss-function* (or equivalently, maximizing a reward function).

Understanding what the function with are optimizing actually means, and why we are chosing it over other functions, is an important part to understand what we can expect of the model when it is trained.

Loss Functions
Motivation

A lot of machine-learning models is based on *minimizing a loss-function* (or equivalently, maximizing a reward function).

Understanding what the function with are optimizing actually means, and why we are chosing it over other functions, is an important part to understand what we can expect of the model when it is trained.

We will look at *Mean Square Error (MSE)*, *binary cross entropy loss* and *multi-class cross entropy loss*. This also includes looking at the *sigmoid-* and *softmax-function* to make sense of the loss.

Loss Functions 2

What is a loss function?

A loss-function is a measure of *how bad the outputs from a model* are.

Loss Functions 2
What is a loss function?

A loss-function is a measure of *how bad the outputs from a model* are.

- In our context, we have a set of input vectors $\{\mathbf{x_i}\}_{i=1:N}$ and corresponding true labels $\{t_i\}_{i=1:N}$, and a model that for every input $\mathbf{x_i}$ gives an output $\mathbf{\hat{y}_i}$. The model usually have some trainable weights W (this is the supervised setting).

Loss Functions 2

What is a loss function?

A loss-function is a measure of *how bad the outputs from a model* are.

- In our context, we have a set of input vectors $\{\mathbf{x_i}\}_{i=1:N}$ and corresponding true labels $\{t_i\}_{i=1:N}$, and a model that for every input $\mathbf{x_i}$ gives an output $\mathbf{\hat{y}_i}$. The model usually have some trainable weights W (this is the supervised setting).
- At any point during or after training, we can meassure the models performance by putting the *outputs* (predicted values) and the true labels into a loss function.

Loss Functions 2

What is a loss function?

A loss-function is a measure of *how bad the outputs from a model* are.

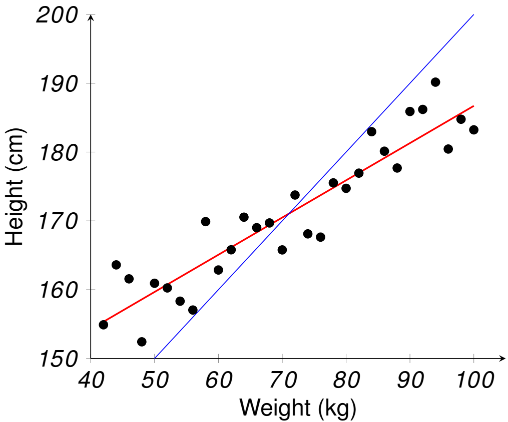
- In our context, we have a set of input vectors $\{\mathbf{x_i}\}_{i=1:N}$ and corresponding true labels $\{t_i\}_{i=1:N}$, and a model that for every input $\mathbf{x_i}$ gives an output $\mathbf{\hat{y}_i}$. The model usually have some trainable weights W (this is the supervised setting).
- At any point during or after training, we can meassure the models performance by putting the *outputs* (predicted values) and the true labels into a loss function.
- The higher the loss function, the worse our model fits adapts to our data, so we want to minimize the loss-function.

A loss-function is a measure of *how bad the outputs from a model* are.

- In our context, we have a set of input vectors $\{\mathbf{x_i}\}_{i=1:N}$ and corresponding true labels $\{t_i\}_{i=1:N}$, and a model that for every input $\mathbf{x_i}$ gives an output $\hat{\mathbf{y}}_\mathbf{i}$. The model usually have some trainable weights W (this is the supervised setting).
- At any point during or after training, we can meassure the models performance by putting the *outputs* (predicted values) and the true labels into a loss function.
- The higher the loss function, the worse our model fits adapts to our data, so we want to minimize the loss-function.
- Interpreting the loss from *training* and *testing* can be very different, but for now, let us focus on the function itself.

Arriving at our first loss function

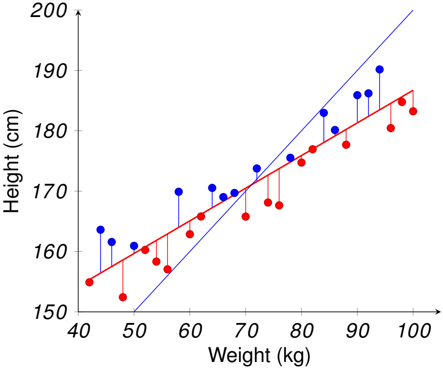
We want to determine how good a linear model fits our data.



Which line fits the data the best, and how do we measure it?

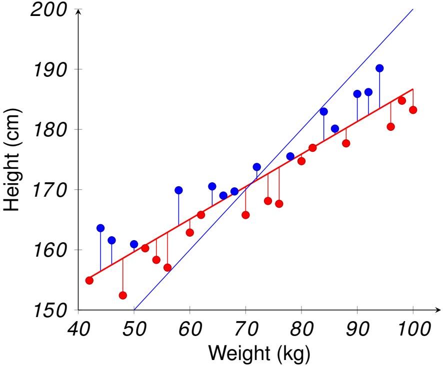
Arriving at our first loss function2

We can sum the distance from the points onto the line.



Arriving at our first loss function2

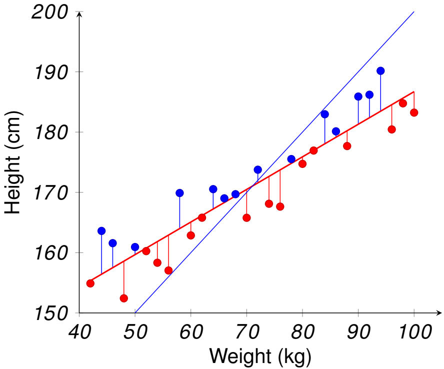
We can sum the distance from the points onto the line.



One important point is that we do not want *negative* and *positive* values to cancel each other.

Arriving at our first loss function2

We can sum the distance from the points onto the line.



One important point is that we do not want *negative* and *positive* values to cancel each other.
We can therefore *square* the distance, to make everything positive. We then take the mean over all the errors (residuals).

Mean Squared Error

Definition and Breakdown

One of the most widely used loss functions is the *Mean Squared Error* (MSE).
It is defined as

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t_i - \hat{y}_i)^2$$

.

.

Mean Squared Error

Definition and Breakdown

One of the most widely used loss functions is the *Mean Squared Error* (MSE).
It is defined as

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t_i - \hat{y}_i)^2$$

.
Let's break this down.

Mean Squared Error

Definition and Breakdown

One of the most widely used loss functions is the *Mean Squared Error* (MSE).
It is defined as

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t_i - \hat{y}_i)^2$$

.
Let's break this down.

- One term of the sum becomes $(t_i - \hat{y}_i)^2$. This is the difference between the true label t_i and the predicted value $\hat{y}$, squared. We squared it to make it positive (so positive and negative values do not cancel each other).
- We then sum over the N amount of inputs, $\sum_{i=1}^{N}(t_i - \hat{y}_i)^2 = (t_1 - \hat{y}_1)^2 + (t_2 - \hat{y}_2)^2 + \ldots + (t_N - \hat{y}_N)^2$

Mean Squared Error

Definition and Breakdown

One of the most widely used loss functions is the *Mean Squared Error* (MSE).
It is defined as

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t_i - \hat{y}_i)^2$$

.

Let's break this down.

- One term of the sum becomes $(t_i - \hat{y}_i)^2$. This is the difference between the true label t_i and the predicted value $\hat{y}$, squared. We squared it to make it positive (so positive and negative values do not cancel each other).
- We then sum over the N amount of inputs, $\sum_{i=1}^{N}(t_i - \hat{y}_i)^2 = (t_1 - \hat{y}_1)^2 + (t_2 - \hat{y}_2)^2 + \ldots + (t_N - \hat{y}_N)^2$
- Lastly, we divide by N, to average over the amount of inputs. If we did not do this, then smaller amount of training data would give smaller error.

Mean Squared Error2

Summary

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t_i - \hat{y}_i)^2$$

Properties:

Mean Squared Error2

Summary

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t_i - \hat{y}_i)^2$$

Properties:

- This can be used for *regression*, predicting *real-values*, or in other words, predicting values on the number-line (prices, height, age, etc). It can be used for *classification* (dog / cat, fail/ success, etc), but is not very well suited for that task.

Mean Squared Error2

Summary

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t_i - \hat{y}_i)^2$$

Properties:

- This can be used for *regression*, predicting *real-values*, or in other words, predicting values on the number-line (prices, height, age, etc). It can be used for *classification* (dog / cat, fail/ success, etc), but is not very well suited for that task.
- It is easy to differentiate, which is very important when optimizing it.

Mean Squared Error2

Summary

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t_i - \hat{y}_i)^2$$

Properties:

- This can be used for *regression*, predicting *real-values*, or in other words, predicting values on the number-line (prices, height, age, etc). It can be used for *classification* (dog / cat, fail/ success, etc), but is not very well suited for that task.
- It is easy to differentiate, which is very important when optimizing it.
- Some models can be optimized *analytically*, without the use of iterativly updating the weights.

Classification

What about classification? We can use MSE, but it does not make so much intuitively sense, and performs poorly mathematically.

Let us first begin with the *binary classification* task. This means we are predicting one of two values, wich we can denote by 0 and 1.

Classification

What about classification? We can use MSE, but it does not make so much intuitively sense, and performs poorly mathematically.

Let us first begin with the *binary classification* task. This means we are predicting one of two values, wich we can denote by 0 and 1.

- We can make our model output only 0 or 1 (as with Perceptron), but this gives us little information about how sure it was of each prediction, and makes gradient descent perform poorly (or not at all).

Classification

What about classification? We can use MSE, but it does not make so much intuitively sense, and performs poorly mathematically.

Let us first begin with the *binary classification* task. This means we are predicting one of two values, wich we can denote by 0 and 1.

- We can make our model output only 0 or 1 (as with Perceptron), but this gives us little information about how sure it was of each prediction, and makes gradient descent perform poorly (or not at all).
- Another alternative is to force our model to output values only on the interval between $(0, 1)$. This means that a the close we are to 0, the more the model is sure that it should predict 0.

Classification

What about classification? We can use MSE, but it does not make so much intuitively sense, and performs poorly mathematically.

Let us first begin with the *binary classification* task. This means we are predicting one of two values, wich we can denote by 0 and 1.

- We can make our model output only 0 or 1 (as with Perceptron), but this gives us little information about how sure it was of each prediction, and makes gradient descent perform poorly (or not at all).
- Another alternative is to force our model to output values only on the interval between $(0, 1)$. This means that a the close we are to 0, the more the model is sure that it should predict 0.
- We can then make a loss function that penalize values that are far from the label t_i. This means that if $t_i = 1$, then $\hat{y}_i = 0.63$ should give a higher loss than if $\hat{y}_i = 0.94$.

Classification

What about classification? We can use MSE, but it does not make so much intuitively sense, and performs poorly mathematically.

Let us first begin with the *binary classification* task. This means we are predicting one of two values, wich we can denote by 0 and 1.

- We can make our model output only 0 or 1 (as with Perceptron), but this gives us little information about how sure it was of each prediction, and makes gradient descent perform poorly (or not at all).
- Another alternative is to force our model to output values only on the interval between $(0, 1)$. This means that a the close we are to 0, the more the model is sure that it should predict 0.
- We can then make a loss function that penalize values that are far from the label t_i. This means that if $t_i = 1$, then $\hat{y}_i = 0.63$ should give a higher loss than if $\hat{y}_i = 0.94$.
- We can interpret $\hat{y}_i = 0.8$ as the model guessing that observation $\mathbf{x}_i$ has *probability* 0.8 to be 1, and probability 0.2 to be 0.

Sigmoid function

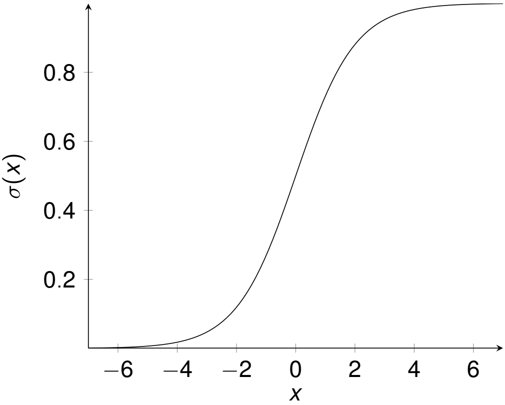
Before we look at the loss, we have to find out how to force our predicted values between 0 and 1.

Sigmoid function

Before we look at the loss, we have to find out how to force our predicted values between 0 and 1.
The sigmoid function, among other, do this. It takes values from $(-\inf, \inf)$ onto $(0, 1)$. It is given by $\sigma(x) = \frac{1}{1+e^{-x}}$.

Sigmoid function

Before we look at the loss, we have to find out how to force our predicted values between 0 and 1.
The sigmoid function, among other, do this. It takes values from $(-\inf, \inf)$ onto $(0, 1)$. It is given by $\sigma(x) = \frac{1}{1+e^{-x}}$.
It looks like this:



Sigmoid function 2

We can now put the values we get from our model, say h, into the sigmoid function, to convert it to probabilities.

This is what we do with *logistic regression*. We have linear weights, as with linear regression: $h = \beta_0 + \beta_1 x_1 + \ldots + \beta_m x_m$, and then get our probability with $\hat{y} = \sigma(h)$.

Sigmoid function 2

We can now put the values we get from our model, say h, into the sigmoid function, to convert it to probabilities.

This is what we do with *logistic regression*. We have linear weights, as with linear regression: $h = \beta_0 + \beta_1 x_1 + \ldots + \beta_m x_m$, and then get our probability with $\hat{y} = \sigma(h)$.

With linear regression we use the h value in the MSE to calculate the loss, but with logistic regression we but the value $\hat{y}_i = \sigma(h_i)$ into the *binary cross entropy loss*.

Sigmoid function 2

We can now put the values we get from our model, say h, into the sigmoid function, to convert it to probabilities.

This is what we do with *logistic regression*. We have linear weights, as with linear regression: $h = \beta_0 + \beta_1 x_1 + \ldots + \beta_m x_m$, and then get our probability with $\hat{y} = \sigma(h)$.

With linear regression we use the h value in the MSE to calculate the loss, but with logistic regression we but the value $\hat{y}_i = \sigma(h_i)$ into the *binary cross entropy loss*.

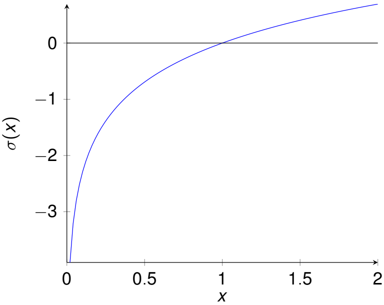
Note that the sigmoid function can be used with other models as well, for example some kinds of neural networks.

Binary Cross Entropy Loss

Let us now slowly get to the loss function. Let us first look at when the true label is 1, $t_i = 1$. We want to have a low loss for when $\hat{y}_i$ is close to 1, and a high loss when it is close to 0.

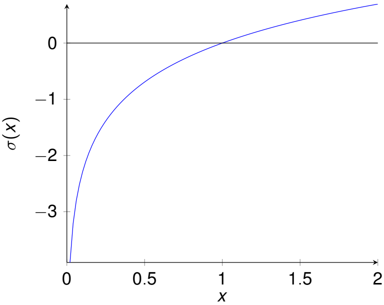
Binary Cross Entropy Loss

Let us now slowly get to the loss function. Let us first look at when the true label is 1, $t_i = 1$. We want to have a low loss for when $\hat{y}_i$ is close to 1, and a high loss when it is close to 0.
We can get help from the logarithmic function, which look like this:



Binary Cross Entropy Loss

Let us now slowly get to the loss function. Let us first look at when the true label is 1, $t_i = 1$. We want to have a low loss for when $\hat{y}_i$ is close to 1, and a high loss when it is close to 0.
We can get help from the logarithmic function, which look like this:



Note that we will only be using x-values between 0 and 1. We get that $\log(x)$ is close to 0 when x is close to 1, and $\log(x)$ goes to $-\inf$ when x goes to 0.

Binary Cross Entropy Loss 2

Repeat: We get that $\log(x)$ is close to 0 when x is close to 1, and $\log(x)$ goes to $-\inf$ when x goes to 0.

Therefore, when t_i is 1, we can add the loss $-\log(\hat{y})$. This gets close to 0 when $\hat{y}$ is close to 1, and close to infinity when $\hat{y}$ is close to 0.

Binary Cross Entropy Loss 2

Repeat: We get that $\log(x)$ is close to 0 when x is close to 1, and $\log(x)$ goes to $-\inf$ when x goes to 0.

Therefore, when t_i is 1, we can add the loss $-\log(\hat{y})$. This gets close to 0 when $\hat{y}$ is close to 1, and close to infinity when $\hat{y}$ is close to 0.

If $t_i = 0$, we can use the loss $-\log(1 - \hat{y})$. This gets close to 0 when $\hat{y}$ is close to 0, and close to infinity when $\hat{y}$ is close to 0.

Binary Cross Entropy Loss 2

Repeat: We get that $\log(x)$ is close to 0 when x is close to 1, and $\log(x)$ goes to $-\inf$ when x goes to 0.

Therefore, when t_i is 1, we can add the loss $-\log(\hat{y})$. This gets close to 0 when $\hat{y}$ is close to 1, and close to infinity when $\hat{y}$ is close to 0.

If $t_i = 0$, we can use the loss $-\log(1 - \hat{y})$. This gets close to 0 when $\hat{y}$ is close to 0, and close to infinity when $\hat{y}$ is close to 0.

This is almost all the intuition behind the loss function, There is a couple of ways to write this in one line.

Binary Cross Entropy Loss 2

The complete formula can be written ass

$$L(\hat{\mathbf{y}}, \mathbf{t}) = -\frac{1}{N}\sum_{i=1}^{N}(t_i \log(\hat{y}_i) + (1 - t_i)\log(1 - \hat{y}_i))$$

- Notice that only one of the term is non-zero! Since every t_i is either 0 or 1, either t_i or $(1 - t_i)$ will be 0, and cancel out the term.

Binary Cross Entropy Loss 2

The complete formula can be written ass

$$L(\hat{\mathbf{y}}, \mathbf{t}) = -\frac{1}{N}\sum_{i=1}^{N}(t_i \log(\hat{y}_i) + (1 - t_i)\log(1 - \hat{y}_i))$$

- Notice that only one of the term is non-zero! Since every t_i is either 0 or 1, either t_i or $(1 - t_i)$ will be 0, and cancel out the term.
- Therefore, we are left with the log-term from the previous slide that we wanted.

Binary Cross Entropy Loss 2

The complete formula can be written ass

$$L(\hat{\mathbf{y}}, \mathbf{t}) = -\frac{1}{N}\sum_{i=1}^{N}(t_i \log(\hat{y}_i) + (1 - t_i)\log(1 - \hat{y}_i))$$

- Notice that only one of the term is non-zero! Since every t_i is either 0 or 1, either t_i or $(1 - t_i)$ will be 0, and cancel out the term.
- Therefore, we are left with the log-term from the previous slide that we wanted.
- We then sum over all the predictions and true labels and divide by N, the amount of observations. Notice that we have also moved the minus-sign outside the sum.

Binary Cross Entropy Loss 2

The complete formula can be written ass

$$L(\hat{\mathbf{y}}, \mathbf{t}) = -\frac{1}{N} \sum_{i=1}^{N} (t_i \log(\hat{y}_i) + (1 - t_i) \log(1 - \hat{y}_i))$$

- Notice that only one of the term is non-zero! Since every t_i is either 0 or 1, either t_i or $(1 - t_i)$ will be 0, and cancel out the term.
- Therefore, we are left with the log-term from the previous slide that we wanted.
- We then sum over all the predictions and true labels and divide by N, the amount of observations. Notice that we have also moved the minus-sign outside the sum.
- The terms can be equivalently written as $\log(\hat{y}_i)^{t_i} \cdot \log(1 - \hat{y}_i)^{1-t_i}$ (look at what happens when t_i equals 0 and 1).

Binary cross Entropy Loss 3

A few things to remember:

Binary cross Entropy Loss 3

A few things to remember:

- The cross entropy only really makes sense when our predictions are between 0 and 1, not any real value.

Binary cross Entropy Loss 3

A few things to remember:

- The cross entropy only really makes sense when our predictions are between 0 and 1, not any real value.
- The predictions can also not be exactly 0 or 1, since then a wrong guess will give loss equal infinity.

Binary cross Entropy Loss 3

A few things to remember:

- The cross entropy only really makes sense when our predictions are between 0 and 1, not any real value.
- The predictions can also not be exactly 0 or 1, since then a wrong guess will give loss equal infinity.
- This is used when we have exactly *two* classes. There are also other losses than can be used for binary classifications, but this is probably the most used one.

Binary cross Entropy Loss 3

A few things to remember:

- The cross entropy only really makes sense when our predictions are between 0 and 1, not any real value.
- The predictions can also not be exactly 0 or 1, since then a wrong guess will give loss equal infinity.
- This is used when we have exactly *two* classes. There are also other losses than can be used for binary classifications, but this is probably the most used one.
- This is often call just ”Cross Entropy Loss” or ”Log Loss”. The name comes from *information theory*.

Binary cross Entropy Loss 3

A few things to remember:

- The cross entropy only really makes sense when our predictions are between 0 and 1, not any real value.
- The predictions can also not be exactly 0 or 1, since then a wrong guess will give loss equal infinity.
- This is used when we have exactly *two* classes. There are also other losses than can be used for binary classifications, but this is probably the most used one.
- This is often call just ”Cross Entropy Loss” or ”Log Loss”. The name comes from *information theory*.
- Two reason that it works well in practice is that it 1) makes a convex function for logistic regression and 2) the derivative is very easy to compute.

Multiclass Classification

Now we will look at the *multiclass classification* situation. This means that we will have C different classes to predict. We assume that only one class is the true value.

Multiclass Classification

Now we will look at the *multiclass classification* situation. This means that we will have C different classes to predict. We assume that only one class is the true value.

We often do this with one-hot-encoded vectors, so that $\hat{\mathbf{y}}_i$ and $\mathbf{t}_i$ now are *C-dimensional vectors*. We make every input in $\mathbf{t_i}$ be 0 except one, which has value 1, and corresponds to the correct class.

Multiclass Classification

Now we will look at the *multiclass classification* situation. This means that we will have C different classes to predict. We assume that only one class is the true value.

We often do this with one-hot-encoded vectors, so that $\hat{\mathbf{y}}_i$ and $\mathbf{t}_i$ now are C*-dimensional vectors*. We make every input in $\mathbf{t_i}$ be 0 except one, which has value 1, and corresponds to the correct class.
For example, if we have 5 classes and the ith input's true value is class 4, we have $\mathbf{t_i} = (0, 0, 0, 1, 0)$

Multiclass Classification

Now we will look at the *multiclass classification* situation. This means that we will have C different classes to predict. We assume that only one class is the true value.

We often do this with one-hot-encoded vectors, so that $\hat{\mathbf{y}}_i$ and $\mathbf{t}_i$ now are C*-dimensional vectors*. We make every input in $\mathbf{t_i}$ be 0 except one, which has value 1, and corresponds to the correct class.
For example, if we have 5 classes and the ith input's true value is class 4, we have $\mathbf{t_i} = (0, 0, 0, 1, 0)$

In the binary situation, we made $\hat{y}_i$ be between 0 and 1. **Question:** What should we do we the C-dimensional vector $\hat{\mathbf{y}}_i$?

Multiclass Classification

Now we will look at the *multiclass classification* situation. This means that we will have C different classes to predict. We assume that only one class is the true value.

We often do this with one-hot-encoded vectors, so that $\hat{\mathbf{y}}_i$ and $\mathbf{t}_i$ now are *C-dimensional vectors*. We make every input in $\mathbf{t_i}$ be 0 except one, which has value 1, and corresponds to the correct class.
For example, if we have 5 classes and the ith input's true value is class 4, we have $\mathbf{t_i} = (0, 0, 0, 1, 0)$

In the binary situation, we made $\hat{y}_i$ be between 0 and 1. **Question:** What should we do we the C-dimensional vector $\hat{\mathbf{y}}_i$?
Answer: We can let the inputs in $\hat{\mathbf{y}}_\mathbf{i}$ *sum* to 1. One way of doing this is with the *softmax*-function

Softmax Function

We define the softmax function as

$$s(h_i) = \frac{e^{h_i}}{\sum_{j=1}^{C} e^{h_j}}$$

What happens here?

Softmax Function

We define the softmax function as

$$s(h_i) = \frac{e^{h_i}}{\sum_{j=1}^{C} e^{h_j}}$$

What happens here?

- The exponents makes all of the input in the sum *positive*, thus all of the outputs will be positive.

Softmax Function

We define the softmax function as

$$s(h_i) = \frac{e^{h_i}}{\sum_{j=1}^{C} e^{h_j}}$$

What happens here?

- The exponents makes all of the input in the sum *positive*, thus all of the outputs will be positive.
- The denominator is the sum of the numerators, making the sum of the $s(h_i)$'s sum to 1, $\sum_{i=1}^{C} s(h_i) = 1$.

Softmax Function

We define the softmax function as

$$s(h_i) = \frac{e^{h_i}}{\sum_{j=1}^{C} e^{h_j}}$$

What happens here?

- The exponents makes all of the input in the sum *positive*, thus all of the outputs will be positive.
- The denominator is the sum of the numerators, making the sum of the $s(h_i)$'s sum to 1, $\sum_{i=1}^{C} s(h_i) = 1$.
- Due to the positive and summming to 1 properties, we can interpret the outputs of the softmax for class i as the *probability of the input to belong in class i*.

Multiclass Cross Entropy Loss

We are now ready to define Multiclass Cross Entropy Loss, which may be the most used loss function for multiclass classification. For one input, is defined as:

$$-\sum_{j=1}^{C} t_j \log(s(h_j)) = -\sum_{i=j}^{C} t_j \log(\hat{y}_j)$$

Lets break it down.

Multiclass Cross Entropy Loss

We are now ready to define Multiclass Cross Entropy Loss, which may be the most used loss function for multiclass classification. For one input, is defined as:

$$-\sum_{j=1}^{C} t_j \log(s(h_j)) = -\sum_{i=j}^{C} t_j \log(\hat{y}_j)$$

Lets break it down.

- Remember that $\mathbf{t}$ is one-hot-encoded, so only one of the t_i will be non-zero. This makes all of the terms in the sum 0, except the one representing the true class.

Multiclass Cross Entropy Loss

We are now ready to define Multiclass Cross Entropy Loss, which may be the most used loss function for multiclass classification. For one input, is defined as:

$$-\sum_{j=1}^{C} t_j \log(s(h_j)) = -\sum_{i=j}^{C} t_j \log(\hat{y}_j)$$

Lets break it down.

- Remember that $\mathbf{t}$ is one-hot-encoded, so only one of the t_i will be non-zero. This makes all of the terms in the sum 0, except the one representing the true class.
- For the input k that represent the true class (assume $t_k = 1$), the loss will be determined by $-\log \hat{y}_k$. This works, since if $\hat{y}_k$ is close to 1, we get close to 0 loss, and if it is close to 0, the loss goes to infinity.

Multiclass Cross Entropy Loss 2

$$-\sum_{j=1}^{C} t_j \log(s(h_j)) = -\sum_{i=j}^{C} t_j \log(\hat{y}_j)$$

It is important that we use something like the softmax function to make the outputs sum to 1.

Question: What happens if we would not use the softmax function, but just use for example a sigmoid activation function in each output node?

Multiclass Cross Entropy Loss 2

$$-\sum_{j=1}^{C} t_j \log(s(h_j)) = -\sum_{i=j}^{C} t_j \log(\hat{y}_j)$$

It is important that we use something like the softmax function to make the outputs sum to 1.

Question: What happens if we would not use the softmax function, but just use for example a sigmoid activation function in each output node?

Answer: We might get every node to output 1 and get 0 loss for every input.

Multiclass Cross Entropy Loss 3

Note that this was only for *one* ovservation, so to get the full loss we have to sum over all of the observations.

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} t_{i,j}\log(\hat{y}_{i,j})$$

where $t_{i,j}$ marks the j'th input to the label of the i'th observation.

Multiclass Cross Entropy Loss 3

Note that this was only for *one* ovservation, so to get the full loss we have to sum over all of the observations.

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} t_{i,j}\log(\hat{y}_{i,j})$$

where $t_{i,j}$ marks the j'th input to the label of the i'th observation.

- We can also use other loss functions.

Multiclass Cross Entropy Loss 3

Note that this was only for *one* ovservation, so to get the full loss we have to sum over all of the observations.

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} t_{i,j}\log(\hat{y}_{i,j})$$

where $t_{i,j}$ marks the j'th input to the label of the i'th observation.

- We can also use other loss functions.
- One reason this works well is that the derivative is very quick to compute.

Multiclass Cross Entropy Loss 3

Note that this was only for *one* ovservation, so to get the full loss we have to sum over all of the observations.

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} t_{i,j}\log(\hat{y}_{i,j})$$

where $t_{i,j}$ marks the j'th input to the label of the i'th observation.

- We can also use other loss functions.
- One reason this works well is that the derivative is very quick to compute.
- The binary cross entropy loss can also be written at this form, but then we would have to one-hot-encode the labels and outputs, instead of just using 0 and 1.

Summary

We have (hopefully) looked at:

- What loss functions are and represent.
- Mean Squared Error for regression.
- Sigmoid function and binary cross entropy loss.
- Softmax function and multiclass cross entropy loss.