Logistic Regression
Multi-Class Classification by Softmax Regression

M. R. Hasan
Readings

• Bishop: 4.3.4
• Murphy: 8.3.6, 8.3.7
What We Will Cover

• Multi-class Classification for Binary Classifiers: 2 techniques
• One-versus-All & One-versus-One
• Multi-class Classification by Softmax Regression
• Softmax Regression loss function
• Cross-entropy loss function
• Gradient descent algorithm
• Regularization via Early Stopping
Logistic Regression: Classification

• We will discuss the Logistic Regression model for solving two types of classification problems.
  - Binary
  - Multi-class
Multi-Class Classification

Virginica

Petal

Sepal

Versicolor

Setosa
Multi-Class Classification

• Multi-class classifiers can distinguish between more than two classes.

• Some models are capable of handling multiple classes directly.

• For example: K-Nearest Neighbors, Naïve Bayes, and Random Forest classifiers.
Multi-Class Classification

• However many models are strictly binary classifiers.
• For example: Logistic Regression, Support Vector Machine.

How do we train a binary classifier for multi-class classification?
Multi-Class Classification

- We employ a **simple technique**.
- We build **multiple binary classifiers**!
- We will discuss **two strategies** that we can use to perform multi-class classification using multiple binary classifiers.
  - One-versus-All (OvA) (also called one-versus-the-rest, OvR)
  - One-versus-One (OvO)
- Let’s use an example to describe these two strategies.
Multi-Class Classification

• One-versus-All (OvA) (or one-versus-the-rest, OvR)
• Consider a handwritten digit classification problem.
• We want to create a classifier to identify 10 digits (digits from 0 to 9).
• In OvA, we **train 10 binary classifiers**.
• One classifier for each digit (a 0-detector, a 1-detector, a 2-detector, and so on).
• Then to classify an image, we get the **decision score from each classifier** for that image.
• We select the class whose classifier outputs the **highest score**.
Multi-Class Classification

• **One-versus-One (OvO) strategy**
  
  • We train a binary classifier for *every pair of digits*:
    - one to distinguish 0s and 1s,
    - another to distinguish 0s and 2s,
    - another for 1s and 2s, and so on.

• If there are N classes, we need to train $K \times (K - 1) / 2$ classifiers.
Multi-Class Classification

• For the 10-digit classification problem we have to train 45 binary classifiers!
• To classify an image, we have to run the image through all 45 classifiers and see which class wins the most duels.
• The main advantage of OvO is that we don’t need the entire dataset for training each classifier.
• Each classifier only needs to be trained on the part of the training set for the two classes that it must distinguish.
Multi-Class Classification

• Later we will see that some ML models scale poorly with the size of the training set.
• For example: Support Vector Machine.
• So for these models, OvO is preferred.
• Because using OvO it is faster to train many classifiers on small training sets than training few classifiers on large training sets.

For most binary classification models, OvA is preferred for multi-class classification.
Multi-Class Logistic Regression

• Logistic Regression model is a **binary classifier**.
• Thus, OvA is the general approach for multi-class classification using Logistic Regression.
• But there is a technique to **avoid** training multiple binary classifiers in the OvA.
• Using this technique we can build a **single classifier** for Logistic Regression to support **multiple classes directly**.
• This technique is called **Softmax Regression**.
Multi-Class Logistic Regression

• **Summary**: to perform multi-class classification using Logistic Regression, we may use two approaches:
  
  • **Approach 1**: One-vs-All (train multiple *binary* classifiers)
  
  • **Approach 2**: *Softmax Regression* (train a *single* classifier)

We will present the Softmax Regression technique
Multi-Class Logistic Regression: Softmax Regression
Multi-Class Logistic Regression

• The Softmax regression is also known as Multinomial Logistic Regression or maximum entropy classifier.
Multi-Class Logistic Regression

• The idea of the **Softmax regression** is quite simple.
• Given an **instance** \( \vec{x} \) (d-dimensional feature) as a 1D column vector, the Softmax Regression model first **computes a score** \( s_c(\vec{x}) \) for **each class** \( c \).

\[
s_c(\vec{x}) = \overrightarrow{w}_c^T \vec{x}
\]

\( \overrightarrow{w}_c \) is the weight vector for class “c” as a 1D column vector.

Then **estimates the probability** of each class by applying the **softmax function** to the scores.

\[
\hat{\mu}_c = P(y = c \mid \vec{x}, \overrightarrow{W}) = \frac{\exp(\overrightarrow{w}_c^T \vec{x})}{\sum_{c' = 1}^{C} \exp(\overrightarrow{w}_{c'}^T \vec{x})} = \frac{\exp(s_c)}{\sum_{c' = 1}^{C} \exp(s_{c'})}
\]

The **softmax function** is just the **normalized exponential** of the scores.
Multi-Class Logistic Regression

- The **equation** to compute \( s_c(x) \):

\[
s_c(x) = \overrightarrow{w}_c^T \overrightarrow{x}
\]

This equation is just like the **equation** for Linear Regression prediction.

Note that **each class** has its own **dedicated** parameter vector \( \overrightarrow{w}_c \).

All these vectors are typically **stored as rows** in a parameter **matrix** \( \overrightarrow{W} \).

Dimension of \( \overrightarrow{W} \): \( d \times C \)
Multi-Class Logistic Regression

- All these vectors are typically stored as rows in a parameter matrix $\vec{W}$.

\[
\vec{W} = \begin{bmatrix} w_{d,c} \end{bmatrix}
\]

Dimension of $\vec{W}$: $d \times C$

For example: if $C = 3$ & $d = 4$, then $\vec{W}$ will be a $4 \times 3$ matrix.

Each class “c” has its own dedicated parameter vector $\vec{w}_c$.

<table>
<thead>
<tr>
<th></th>
<th>c = 1</th>
<th>c = 2</th>
<th>c = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>d = 1</td>
<td>$w_{1,1}$</td>
<td>$w_{1,2}$</td>
<td>$w_{1,3}$</td>
</tr>
<tr>
<td>d = 2</td>
<td>$w_{2,1}$</td>
<td>$w_{2,2}$</td>
<td>$w_{2,3}$</td>
</tr>
<tr>
<td>d = 3</td>
<td>$w_{3,1}$</td>
<td>$w_{3,2}$</td>
<td>$w_{3,3}$</td>
</tr>
<tr>
<td>d = 4</td>
<td>$w_{4,1}$</td>
<td>$w_{4,2}$</td>
<td>$w_{4,3}$</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\vec{w}_1 &= [w_{1,1} \ w_{2,1} \ w_{3,1} \ w_{4,1}] \\
\vec{w}_2 &= [w_{1,2} \ w_{2,2} \ w_{3,2} \ w_{4,2}] \\
\vec{w}_3 &= [w_{1,3} \ w_{2,3} \ w_{3,3} \ w_{4,3}]
\end{align*}
\]
Multi-Class Logistic Regression

• The scores should be stored in a score matrix $\hat{S}$.
• Each row of the score matrix corresponds to the scores of a single instance of the data.
• The dimension of the score matrix will be $N \times C$.

\[ s_c(\hat{x}) = \hat{w}_c^T \hat{x} \]
Multi-Class Logistic Regression

- For example, if there are 100 samples \((N = 100)\) and three classes \((C = 3)\), the dimension of the score matrix will be \(100 \times 3\).

- For each sample (a single row), the column \(c\) will store the score of that sample belonging to class \(c\): \(\vec{S}_{ic} = \vec{W}_c^T \cdot \vec{x}_i\)

\[
\begin{bmatrix}
\vec{S}_{1c} = \vec{W}_c^T \cdot \vec{x}_1 \\
\vec{S} = \vec{X} \cdot \vec{W}
\end{bmatrix}
\]

<table>
<thead>
<tr>
<th>(i)</th>
<th>Class 1 ((c = 1))</th>
<th>Class 2 ((c = 2))</th>
<th>Class 3 ((c = 3))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(\vec{s}_{1,1} = \vec{w}_1^T \cdot \vec{x}_1)</td>
<td>(\vec{s}_{1,2} = \vec{w}_2^T \cdot \vec{x}_1)</td>
<td>(\vec{s}_{1,3} = \vec{w}_3^T \cdot \vec{x}_1)</td>
</tr>
<tr>
<td>2</td>
<td>(\vec{s}_{2,1} = \vec{w}_1^T \cdot \vec{x}_2)</td>
<td>(\vec{s}_{2,2} = \vec{w}_2^T \cdot \vec{x}_2)</td>
<td>(\vec{s}_{2,3} = \vec{w}_3^T \cdot \vec{x}_2)</td>
</tr>
<tr>
<td>3</td>
<td>(\vec{s}_{3,1} = \vec{w}_1^T \cdot \vec{x}_3)</td>
<td>(\vec{s}_{3,2} = \vec{w}_2^T \cdot \vec{x}_3)</td>
<td>(\vec{s}_{3,3} = \vec{w}_3^T \cdot \vec{x}_3)</td>
</tr>
<tr>
<td>(\ldots)</td>
<td>(\ldots)</td>
<td>(\ldots)</td>
<td>(\ldots)</td>
</tr>
<tr>
<td>100</td>
<td>(\vec{s}_{100,1} = \vec{w}<em>1^T \cdot \vec{x}</em>{100})</td>
<td>(\vec{s}_{100,2} = \vec{w}<em>2^T \cdot \vec{x}</em>{100})</td>
<td>(\vec{s}_{100,3} = \vec{w}<em>3^T \cdot \vec{x}</em>{100})</td>
</tr>
</tbody>
</table>
Multi-Class Logistic Regression

• Once we have computed the score of every class for the instance \( x \),

  we can estimate the probability \( \mu_c \) that the instance belongs to class \( c \) by running the scores through the softmax function.

• The softmax function computes the exponential of every score, then normalizes them (dividing by the sum of all the exponentials).

\[
\hat{\mu}_c = P(y = c \mid \hat{x}, \overrightarrow{W}) = \frac{\exp(\overrightarrow{w}_c^T \hat{x})}{\sum_{c' = 1}^{C} \exp(\overrightarrow{w}_{c'}^T \hat{x})} = \frac{\exp(s_c)}{\sum_{c' = 1}^{C} \exp(s_{c'})}
\]
Multi-Class Logistic Regression

- It computes the **exponential of every score**, then normalizes them (dividing by the sum of all the exponentials). **Example**:

\[
\mu_{i=1,c=1} = P(y_{i=1} = (c = 1) \mid \vec{x}_1, \vec{W}) = \frac{\exp(\vec{w}_1^T \vec{x}_1)}{\exp(\vec{w}_1^T \vec{x}_1) + \exp(\vec{w}_2^T \vec{x}_1) + \exp(\vec{w}_3^T \vec{x}_1)}
\]

\[
\vec{S}_{ic} = \vec{w}_c^T \cdot \vec{x}_i
\]

Note that we shouldn’t compare the **scores directly** to determine the optimal class.

We need to **calibrate** the scores first via the **softmax** function.
Multi-Class Logistic Regression

• Instead of using softmax for normalization, can we use a \textbf{standard normalizer}?

\[ \hat{\mu}_c = P(y = c \mid \mathbf{x}, \mathbf{W}) = \frac{\mathbf{W}_c^T \mathbf{x}}{\sum_{c' = 1}^{C} \mathbf{W}_{c'}^T \mathbf{x}} = \frac{s_c}{\sum_{c' = 1}^{C} s_{c'}} \]

\text{Standard Normalizer:}

\[ \hat{\mu}_c = P(y = c \mid \mathbf{x}, \mathbf{W}) = \frac{\exp(\mathbf{W}_c^T \mathbf{x})}{\sum_{c' = 1}^{C} \exp(\mathbf{W}_{c'}^T \mathbf{x})} = \frac{\exp(s_c)}{\sum_{c' = 1}^{C} \exp(s_{c'})} \]

\text{Softmax Normalizer:}

We prefer softmax normalizer over the standard normalizer for mainly \textbf{two reasons}. 
Reason 1: For smaller scores, softmax assigns **low probability**, whereas standard normalization only look at the proportion of the scores.

```python
import numpy as np

def softmax(a, b):
    numerator = np.exp(a)
    denominator = np.exp(a) + np.exp(b)
    return numerator / denominator

def standard_normalization(a, b):
    return a / (a + b)

def softmax_normalizer(s):
    return np.exp(s) / np.sum(np.exp(s))

print("Standard Normalization: ", standard_normalization(1, 2))
print("Standard Normalization: ", standard_normalization(10, 20))
print("Standard Normalization: ", standard_normalization(100, 200))

print("Softmax Normalization: ", softmax_normalizer([1, 2]))
print("Softmax Normalization: ", softmax_normalizer([10, 20]))
print("Softmax Normalization: ", softmax_normalizer([100, 200]))
```

**Softmax Normalizer:**

\[
\hat{\mu}_c = \frac{\exp(s_c)}{\sum_{c' = 1}^C \exp(s_{c'})}
\]

**Standard Normalizer:**

\[
\hat{\mu}_c = \frac{s_c}{\sum_{c' = 1}^C s_{c'}}
\]

Example: normalized score of a sample given two sample scores
Multi-Class Logistic Regression

Reason 2: To define the multi-class loss function (cross-entropy, discussed later), we compute the log of the softmax probability. The exponential function in softmax is roughly cancelled out due to the log in the loss function. As a result loss is approximately linear in score $\hat{w}_c^T \hat{x}$, which leads to a roughly constant gradient.

It helps to avoid vanishing gradient for an incorrect saturated softmax (because after cancelling exp()), loss becomes much smaller, which will give smaller gradient.

Softmax Normalizer:

\[ \hat{c} = \arg \max_c \left( \sum_{c=1}^C \frac{\exp(w_c^T \hat{x})}{\sum_{c=1}^C \exp(w_c^T \hat{x})} \right) \]

\[ \hat{y} = \{ y = c \mid \hat{c} \} \]

\[ \hat{\mu} = \hat{y} \frac{\sum_{c=1}^C \exp(w_c^T \hat{x})}{\sum_{c=1}^C \exp(w_c^T \hat{x})} \]

\[ \ell(\hat{y}) = -\sum_{y=1}^N \sum_{c=1}^C \hat{y}_c \log \left( \frac{\exp(w_c^T \hat{x})}{\sum_{c=1}^C \exp(w_c^T \hat{x})} \right) \]

Standard Normalizer:

\[ \hat{w}_c = \frac{\sum_{c=1}^C \exp(s_c) \hat{w}_c}{\sum_{c=1}^C \exp(s_c)} \]

\[ \hat{w}_c = \hat{y} \frac{\sum_{c=1}^C \exp(w_c^T \hat{x})}{\sum_{c=1}^C \exp(w_c^T \hat{x})} \]

\[ \frac{\sum_{c=1}^C \exp(w_c^T \hat{x})}{\sum_{c=1}^C \exp(w_c^T \hat{x})} \]
Multi-Class Logistic Regression

- After computing class probabilities using softmax, we store the values in a **probability matrix** $\text{Prob}$.
- **Each row of the probability matrix** corresponds to the probabilities of a single instance of the data for different classes.
- The **dimension** of the probability matrix will be $N \times C$.

$$
\hat{\mu}_c = P(y = c | \bar{x}, \bar{W}) = \frac{\exp(\vec{w}_c^T \bar{x})}{\sum_{c' = 1}^{C} \exp(\vec{w}_{c'}^T \bar{x})} = \frac{\exp(s_c)}{\sum_{c' = 1}^{C} \exp(s_{c'})}
$$
Multi-Class LR

\[ \hat{\mu}_{ic} = P(y_i = c | \tilde{x}_i, \mathcal{W}) = \frac{\exp(\mathbf{w}_c^T \tilde{x}_i)}{\sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^T \tilde{x}_i)} = \frac{\exp(s_{ic})}{\sum_{c'=1}^{C} \exp(s_{ic'})} \]

- For example, if there are 100 samples (N = 100) and three classes (C = 3), the dimension of the probability matrix will be 100 x 3.

- For each sample (a single row), the column \( c \) stores the probability of that sample belonging to class \( c \):

\[ \overrightarrow{Y(Proba)}_{ic} = [\mu_{ic}] \]

<table>
<thead>
<tr>
<th>( i )</th>
<th>Class 1 (c = 1)</th>
<th>Class 2 (c = 2)</th>
<th>Class 3 (c = 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \mu_{1,1} )</td>
<td>( \mu_{1,2} )</td>
<td>( \mu_{1,3} )</td>
</tr>
<tr>
<td>2</td>
<td>( \mu_{2,1} )</td>
<td>( \mu_{2,2} )</td>
<td>( \mu_{2,3} )</td>
</tr>
<tr>
<td>3</td>
<td>( \mu_{3,1} )</td>
<td>( \mu_{3,2} )</td>
<td>( \mu_{3,3} )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>100</td>
<td>( \mu_{100,1} )</td>
<td>( \mu_{100,2} )</td>
<td>( \mu_{100,3} )</td>
</tr>
</tbody>
</table>
Multi-Class LR

\[ \hat{\mu}_{ic} = P(y_i = c | \vec{x}_i, \vec{W}) = \frac{\exp(\vec{w}_c^T \vec{x}_i)}{\sum_{c' = 1}^{C} \exp(\vec{w}_{c'}^T \vec{x}_i)} = \frac{\exp(s_{ic})}{\sum_{c' = 1}^{C} \exp(s_{ic'})} \]

Each row of the score matrix corresponds to the scores of a single instance of the data.

\[ \vec{S}_{ic} = \vec{w}_c^T \vec{x}_i \]

Example: \( C = 3, N = 100, d = 4 \)

Each row of the probability matrix corresponds to the probabilities of a single instance of the data.

\[ \vec{W} = [w_{d,c}] \]

<table>
<thead>
<tr>
<th></th>
<th>( c = 1 )</th>
<th>( c = 2 )</th>
<th>( c = 3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d = 1 )</td>
<td>( w_{1,1} )</td>
<td>( w_{1,2} )</td>
<td>( w_{1,3} )</td>
</tr>
<tr>
<td>( d = 2 )</td>
<td>( w_{2,1} )</td>
<td>( w_{2,2} )</td>
<td>( w_{2,3} )</td>
</tr>
<tr>
<td>( d = 3 )</td>
<td>( w_{3,1} )</td>
<td>( w_{3,2} )</td>
<td>( w_{3,3} )</td>
</tr>
<tr>
<td>( d = 4 )</td>
<td>( w_{4,1} )</td>
<td>( w_{4,2} )</td>
<td>( w_{4,3} )</td>
</tr>
</tbody>
</table>
Multi-Class LR

\[
\hat{\mu}_{ic} = P(y_i = c \mid \hat{x}_i, \overrightarrow{\mathbf{w}}) = \frac{\exp(\overrightarrow{w}_c^T \hat{x}_i)}{\sum_{c'=1}^{C} \exp(\overrightarrow{w}_{c'}^T \hat{x}_i)} = \frac{\exp(s_{ic})}{\sum_{c'=1}^{C} \exp(s_{ic'})}
\]

- Just like the Logistic Regression classifier, the Softmax Regression classifier predicts the class with the highest estimated probability.

- It is simply the class with the highest score.

\[
\hat{\mu} = \arg\max_c s_c(\hat{x}) = \arg\max_c s_c(\overrightarrow{w}_c^T \hat{x})
\]

Thus, for an instance \textbf{i} (row \(i\)), the predicted class will be the column index \(c\) that has the maximum value in \(Y(Proba)_{ic} = [\mu_{ic}]\)
Multi-Class Logistic Regression

- So far, we learned how the model estimates probabilities and makes predictions.
- To compute these probabilities, we need to find $\bar{W}$.
- Let’s discuss how to train the model.

$$
\hat{\mu}_{ic} = P(y_i = c \mid \bar{x}_i, \bar{W}) = \frac{\exp(\bar{w}_c^T \bar{x}_i)}{\sum_{c' = 1}^{C} \exp(\bar{w}_{c'}^T \bar{x}_i)} = \frac{\exp(s_{ic})}{\sum_{c' = 1}^{C} \exp(s_{ic'})}
$$
Softmax Regression: Algorithm for Learning
Multi-Class Logistic Regression

• We find the model parameters via the Gradient Descent algorithm.

• We initialize $\mathbf{W}$ with zero.

• Then, using Gradient Descent we update $\mathbf{W}$ by minimizing the cost/loss function.
Constructing the Loss Function for Multi-Class Logistic Regression
Multi-Class Logistic Regression

- Gradient Descent updates weights for all classes.
- Then, it measures the **prediction loss for each class**.
- Thus, for the samples we **need a label (0 or 1) for each class**.
- However, the target vector $\tilde{Y}$ does not provide the 0/1 label for each class.
- From $\tilde{Y}$, we need to **create labels for each class**.
- We can do this by **one-hot encoding**.

Example: **3-class** classification problem; Each of the 4 samples has **only one label**: $\tilde{Y} = [1 \ 3 \ 2 \ 1]^T$
Multi-Class Logistic Regression

- Using **one-hot-encoding**, the target will be represented as a \( N \times C \) matrix \( Y(\text{one} - \text{hot}) \).

<table>
<thead>
<tr>
<th>( \bar{Y} )</th>
<th>( Y(\text{one} - \text{hot}) )</th>
<th>Class 1 (c = 1)</th>
<th>Class 2 (c = 2)</th>
<th>Class 3 (c = 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>i = 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>i = 2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>i = 3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>i = 4</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Multi-Class Logistic Regression

- We **construct the loss function** using a similar approach that we applied in case of binary classification (logistic regression).
- It is called the **cross-entropy loss function**.

\[
\mathcal{L}(\vec{W}) = - \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic}
\]

\[
\mathcal{L}(\vec{W}) = - \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \left[ \frac{\exp(\vec{w}_c^T \vec{x}_i)}{\sum_{c'}^{C} \exp(\vec{w}_{c'}^T \vec{x}_i)} \right]
\]
Multi-Class LR

\[ \hat{\mu}_{ic} = p(y_i = c \mid \tilde{x}_i, \vec{w}) = \frac{\exp(\vec{w}_c^T \tilde{x}_i)}{\sum_{c'=1}^{C} \exp(\vec{w}_{c'}^T \tilde{x}_i)} = \frac{\exp(s_{ic})}{\sum_{c'=1}^{C} \exp(s_{ic'})} \]

- **Intuitive justification** for using cross-entropy as the loss function.
- We expect a loss function to return large values (i.e., loss is high) if our models makes **misclassification**.

\[ \mathcal{L} (\vec{w}) = - \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic} \]

![Graph showing Loss = - y_{ic} \log(\mu_{ic})](image)

- Loss increases **rapidly** as \( \mu_{ic} \to 0 \)
- \( y_{ic} = 1 \)
- \( c = 1 \)
Multi-Class LR

\[
\hat{\mu}_{ic} = P(y_i = c | \tilde{x}_i, \overrightarrow{W}) = \frac{\exp(\overrightarrow{w}_c^T \tilde{x}_i)}{\sum_{c'=1}^{C} \exp(\overrightarrow{w}_{c'}^T \tilde{x}_i)} = \frac{\exp(s_{ic})}{\sum_{c'=1}^{C} \exp(s_{ic'})}
\]

• Why is the use of the **logarithm** of class probability density justified?
• Because as the class probability goes to 0, changes in log probability is much larger (**hence, very large loss**) than changes to the probability.

Since we are dealing with probabilities, this is the region we care about.

\[
\mathcal{L}(\overrightarrow{W}) = - \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic}
\]
Multi-Class LR: Loss Function

• The **cross-entropy** loss function.

\[
\mathcal{L}(\vec{W}) = - \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic}
\]

In the Batch Gradient Descent algorithm, we use the **average** of the cross-entropy loss.

It’s the **empirical loss** (risk/error), which we use to estimate the true loss.

\[
\mathcal{L}(\vec{W}) = - \frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic}
\]
Multi-Class LR: Loss Function

- The **cross-entropy** loss function for all samples in matrix notation (used in batch GD).

\[ \mathcal{L}(\vec{W}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic} \]

- Matrix one-hot \( \vec{Y} \): \( N \times C \)
- Matrix \( \vec{\mu} \): \( N \times C \)

Multiply each row of the one-hot matrix \( \vec{Y} \) with the respective row of \( \vec{\mu} \), then take the sum of all products.
Multi-Class LR: Loss Function

- The **cross-entropy** loss function for one sample in vector notation (used in SGD).

\[
\mathcal{L}(\vec{W}) = - \frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic}
\]

- Subscript \( i \) refers to the sample index.

- One-hot vector \( \vec{y}_i : 1 \times C \)

- Vector \( \vec{\mu}_i : 1 \times C \)
Multi-Class LR

- **Batch Gradient Descent** algorithm for multi-class logistic regression:

\[
\hat{\mu}_{ic} = P(y_i = c \mid \vec{x}_i, \vec{W}) = \frac{\exp(\vec{w}_c^T \vec{x}_i)}{\sum_{c'=1}^{C} \exp(\vec{w}_{c'}^T \vec{x}_i)} = \frac{\exp(s_{ic})}{\sum_{c'=1}^{C} \exp(s_{ic'})}
\]

\[
\mathcal{L} (\vec{W}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic}
\]

**Initialize** \( \vec{W}^{(0)} = \vec{0} \)

**Update rule:**

\[
\vec{W}^{(t+1)} = \vec{W}^{(t)} - \eta \nabla_{\vec{W}^{(t)}} \mathcal{L}(\vec{W}^{(t)})
\]
Multi-Class LR

- For the **Batch Gradient Descent update rule**, we need to compute the **gradient** of the loss function $g(\overrightarrow{W})$ or $\nabla_{\overrightarrow{W}} \mathcal{L}(\overrightarrow{W})$.

For each class $c$, the **gradient** of $\mathcal{L}(\overrightarrow{W})$ wrt the weights $\overrightarrow{w}_c$ in the $c$’th column of $\overrightarrow{W}$:

For **all classes**, the **cross-entropy gradient** $g(\overrightarrow{W})$ in matrix format:

$$g(\overrightarrow{W}) = \nabla_{\overrightarrow{W}} \mathcal{L}(\overrightarrow{W}) = \frac{1}{N} \hat{X}^T (\hat{\mu} - \hat{Y})$$

Matrix **one-hot** $\hat{Y} : N \times C$

Matrix $\hat{\mu} : N \times C$

\[
\overrightarrow{W}_{(t+1)} = \overrightarrow{W}(t) - \eta \nabla_{\overrightarrow{W}(t)} \mathcal{L}(\overrightarrow{W}(t))
\]

\[
\overrightarrow{W} = [w_{d,c}]
\]

<table>
<thead>
<tr>
<th></th>
<th>$c = 1$</th>
<th>$c = 2$</th>
<th>$c = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d = 1$</td>
<td>$w_{1,1}$</td>
<td>$w_{1,2}$</td>
<td>$w_{1,3}$</td>
</tr>
<tr>
<td>$d = 2$</td>
<td>$w_{2,1}$</td>
<td>$w_{2,2}$</td>
<td>$w_{2,3}$</td>
</tr>
<tr>
<td>$d = 3$</td>
<td>$w_{3,1}$</td>
<td>$w_{3,2}$</td>
<td>$w_{3,3}$</td>
</tr>
<tr>
<td>$d = 4$</td>
<td>$w_{3,1}$</td>
<td>$w_{4,2}$</td>
<td>$w_{4,3}$</td>
</tr>
</tbody>
</table>

\[
\mathcal{L}(\overrightarrow{W}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic}
\]

\[
\nabla_{\overrightarrow{w}_c} \mathcal{L}(\overrightarrow{W}) = \frac{1}{N} \sum_{i=1}^{N} (\mu_{ic} - y_{ic}) \cdot \hat{x}_i
\]
Multi-Class LR

- Notice that the multi-class loss gradient has the **same form** as in the **binary logistic regression** case:
  - an error term times $x_i$

\[
\nabla_{\vec{w}} L(\vec{W}) = \frac{1}{N} \vec{X}^T (\vec{\mu} - \vec{Y})
\]

- **Multi-class**
  - Matrix **one-hot** $\vec{Y}$: $N \times C$
  - Matrix $\vec{\mu}$: $N \times C$

\[
\nabla_{\vec{w}} L(\vec{w}) = \frac{1}{N} \vec{X}^T [\sigma(\vec{X}.\vec{w}^{(t)}) - \vec{Y}]
\]

- **Binary**
  - Vector $\vec{Y}$: $N \times 1$
  - Vector $\sigma(\vec{X}.\vec{w}^{(t)})$: $N \times 1$
Multi-Class LR

- Using the expression for the loss gradient, the Batch Gradient Descent update rule is given as follows.

\[ \nabla_{\vec{W}} \mathcal{L}(\vec{W}) = \frac{1}{N} \vec{X}^T (\vec{\mu} - \vec{Y}) \]

**Initialize** \( \vec{W}^{(0)} = \vec{0} \)

**Update rule:**

\[ \vec{W}^{(t+1)} = \vec{W}^{(t)} - \eta \nabla_{\vec{W}^{(t)}} \mathcal{L}(\vec{W}^{(t)}) \]

\[ \vec{W}^{(t+1)} = \vec{W}^{(t)} - \frac{\eta}{N} \vec{X}^T (\vec{\mu} - \vec{Y}) \]

Let’s derive the gradient.
Multi-Class LR

• For the **Batch Gradient Descent update rule**, we need to compute the gradient of the loss function.

• For convenience of the gradient computation, we drop the $\frac{1}{N}$ term from $\mathcal{L}(\overrightarrow{W})$.

• After computing the gradient, we will reinstate it.

\[
\mathcal{L}(\overrightarrow{W}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic}
\]
Loss Gradient Computation: A Sketch
For each class \( c \), the gradient of \( \mathcal{L}(\vec{W}) \) wrt the weights \( \vec{w}_c \) in the \( c \)'th column \( \vec{W} \):

\[
\nabla_{\vec{w}_c} \mathcal{L}(\vec{W}) = \frac{d\mathcal{L}(\vec{W})}{d\vec{w}_c}
\]

Variation in \( \mathcal{L} \) for a sample \( \vec{x}_i \) is due to its weight \( \vec{w}_c \) for class \( c \) is propagated through the following terms.

Loosely we can write by using the chain rule of calculus:

\[
\nabla_{\vec{w}_c} \mathcal{L}(\vec{W}) := \frac{d\mathcal{L}(\vec{W})}{d\mu_{ic}} \cdot \frac{d\mu_{ic}}{ds_{ic}} \cdot \frac{ds_{ic}}{d\vec{w}_c}
\]

\[
\mathcal{L}(\vec{W}) = -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic}
\]

\[
\mu_{ic} = \frac{\exp(s_{ic})}{\sum_{c'=1}^{C} \exp(s_{ic'})}
\]

\[
s_{ic}(\vec{x}_i) = \vec{w}_c^T \vec{x}_i
\]

\[
\vec{w}_c
\]

\[
s_{ic}(\vec{x}_i)
\]

\[
\mu_{ic}
\]

\[
\mathcal{L}(\vec{x}_i)
\]

\[
y_{ic}
\]
Using the chain rule of calculus:

\[
\nabla_{w_c} \mathcal{L}(\mathbf{W}) = \frac{d \mathcal{L}(\mathbf{W})}{d w_c} = \frac{d \mathcal{L}(\mathbf{W})}{d s_{ic}} \cdot \frac{d s_{ic}}{d w_c}
\]

Here the sum is on all samples (subscript \(i\))

\[
\nabla_{w_c} \mathcal{L}(\mathbf{W}) = \sum_{i=1}^{N} \frac{d \mathcal{L}(\mathbf{W})}{d s_{ic}} \cdot \frac{d s_{ic}}{d w_c}
\]

Using the chain rule of calculus:

\[
\nabla_{w_c} \mathcal{L}(\mathbf{W}) = \sum_{i=1}^{N} \left[ \sum_{c=1}^{C} \frac{d \mathcal{L}(\mathbf{W})}{d \mu_{ic}} \cdot \frac{d \mu_{ic}}{d s_{ic}} \right] \cdot \frac{d s_{ic}}{d w_c}
\]

Here for a single sample (subscript \(i\)), the sum is on all classes (subscript \(c\)) as the score \(s_{ic}\) for class \(c\) influences probabilities of all classes.
For each class $c$, the gradient of $\mathcal{L}(\vec{W})$ wrt the weights $\vec{w}_c$ in the $c$'th column $\vec{W}$:

$$\nabla_{\vec{w}_c} \mathcal{L}(\vec{W}) = \sum_{i=1}^{N} \left[ \sum_{c=1}^{C} \frac{d\mathcal{L}(\vec{W})}{d\mu_{ic}} \cdot \frac{d\mu_{ic}}{ds_{ic}} \right] \cdot \frac{ds_{ic}}{d\vec{w}_c}$$

$$\mathcal{L}(\vec{W}) = -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic}$$

For all classes, cross-entropy gradient $g(\vec{W})$:

$$\nabla \mathcal{L}(\vec{W}) = \hat{X}^T (\hat{\mu} - \hat{Y})$$

<table>
<thead>
<tr>
<th>$c$</th>
<th>$d$</th>
<th>$w_{1,c}$</th>
<th>$w_{2,c}$</th>
<th>$w_{3,c}$</th>
<th>$w_{4,c}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$w_{1,1}$</td>
<td>$w_{1,2}$</td>
<td>$w_{1,3}$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>$w_{2,1}$</td>
<td>$w_{2,2}$</td>
<td>$w_{2,3}$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>$w_{3,1}$</td>
<td>$w_{3,2}$</td>
<td>$w_{3,3}$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>$w_{4,1}$</td>
<td>$w_{4,2}$</td>
<td>$w_{4,3}$</td>
<td></td>
</tr>
</tbody>
</table>
Loss Gradient Computation: Detail Computation
Multi-Class LR

- For each class \( c \), the gradient of \( \mathcal{L}(\overrightarrow{W}) \) wrt the weights \( \overrightarrow{w_c} \) in the \( c \)'th column \( \overrightarrow{W} \):

\[
\nabla_{\overrightarrow{w_c}} \mathcal{L}(\overrightarrow{W}) = \frac{d\mathcal{L}(\overrightarrow{W})}{d\overrightarrow{w_c}}
\]

Using the chain rule of calculus:

\[
\nabla_{\overrightarrow{w_c}} \mathcal{L}(\overrightarrow{W}) = \sum_{i=1}^{N} \frac{d\mathcal{L}(\overrightarrow{W})}{ds_{ic}} \cdot \frac{ds_{ic}}{d\overrightarrow{w_c}}
\]

Here the sum is on all samples (subscript \( i \))

\[
\overrightarrow{W} = [w_{d,c}]
\]

<table>
<thead>
<tr>
<th></th>
<th>( c = 1 )</th>
<th>( c = 2 )</th>
<th>( c = 3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d = 1 )</td>
<td>( w_{1,1} )</td>
<td>( w_{1,2} )</td>
<td>( w_{1,3} )</td>
</tr>
<tr>
<td>( d = 2 )</td>
<td>( w_{2,1} )</td>
<td>( w_{2,2} )</td>
<td>( w_{2,3} )</td>
</tr>
<tr>
<td>( d = 3 )</td>
<td>( w_{3,1} )</td>
<td>( w_{3,2} )</td>
<td>( w_{3,3} )</td>
</tr>
<tr>
<td>( d = 4 )</td>
<td>( w_{3,1} )</td>
<td>( w_{4,2} )</td>
<td>( w_{4,3} )</td>
</tr>
</tbody>
</table>

\[
\mathcal{L}(\overrightarrow{W}) = - \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic}
\]

\[
s_{ic}(\hat{x}_i) = \overrightarrow{w_c}^T \hat{x}_i
\]
\[ \beta_{ic} = \frac{\exp(s_{ic})}{\sum_{c'}^C \exp(s_{ic'})} \]

\[ s_{ic}\left(\vec{x}_i\right) = \vec{w}_c^T \vec{x}_i \]

\[ \vec{W} = [w_{d,c}] \]

- We will compute the following two terms separately.
  - \[ \frac{d\mathcal{L}(\vec{W})}{ds_{ic}} \]
  - \[ \frac{ds_{ic}}{d\vec{w}_c} \]

First compute \[ \frac{d\mathcal{L}(\vec{W})}{ds_{ic}} \]

Again apply the chain rule of calculus:

\[ \mathcal{L}(\vec{W}) = -\sum_{i=1}^N \sum_{c=1}^C y_{ic} \log \mu_{ic} \]

\[ \nabla_{\vec{w}_c} \mathcal{L}(\vec{W}) = \sum_{i=1}^N \frac{d\mathcal{L}(\vec{W})}{ds_{ic}} \cdot \frac{ds_{ic}}{d\vec{w}_c} \]

\[ \frac{d\mathcal{L}(\vec{W})}{ds_{ic}} = \sum_{c=1}^C \frac{d\mathcal{L}(\vec{W})}{d\mu_{ic}} \cdot \frac{d\mu_{ic}}{ds_{ic}} \]
Multi-Class LR

First compute \[ \frac{d\mathcal{L}(\vec{W})}{ds_{ic}} \]

Here for a single sample (subscript i), the sum is on all classes (subscript c) as the score \( s_{ic} \) for class c influences probabilities of all classes.

Compute \[ \frac{d\mu_{ic}}{ds_{ic}} \]

We will consider two cases:

1. \( c = c' \)
2. \( c \neq c' \)
\[
\frac{d\mu_{ic}}{ds_{ic}'} = \begin{cases} \quad c = c' \\
\neq \quad c \neq c' \end{cases}
\]
Multi-Class LR

\[ \hat{\mu}_{ic} = P(y_i = c \mid \hat{x}_i, \vec{W}) = \frac{\exp(\vec{w}_c^T \hat{x}_i)}{\sum_{c' = 1}^{C} \exp(\vec{w}_{c'}^T \hat{x}_i)} = \frac{\exp(s_{ic})}{\sum_{c' = 1}^{C} \exp(s_{ic'})} \]

\[ \frac{d\mu_{ic}}{ds_{ic}} = \frac{\exp(s_{ic})}{\sum_{c' = 1}^{C} \exp(s_{ic'})} - \left( \frac{\exp(s_{ic})}{\sum_{c' = 1}^{C} \exp(s_{ic'})} \right)^2 \]

Quotient Rule: \( \left(\frac{f}{g}\right)' = \frac{f'g - fg'}{g^2} \)

\[ \frac{d\mu_{ic}}{ds_{ic}} = \frac{\exp(s_{ic})}{\sum_{c' = 1}^{C} \exp(s_{ic'})} \left[ 1 - \frac{\exp(s_{ic})}{\sum_{c' = 1}^{C} \exp(s_{ic'})} \right] \]

\[ \frac{d\mu_{ic}}{ds_{ic}} = \mu_{ic} (1 - \mu_{ic}) \]

\[ \nabla_{\vec{w}} L(\vec{W}) = \sum_{i=1}^{N} \frac{\partial L(\vec{W})}{\partial s_{ic}} \cdot \frac{\partial s_{ic}}{\partial \vec{w}_c} \]

\[ \frac{\partial L(\vec{W})}{\partial s_{ic}} = \sum_{c=1}^{C} \frac{\partial L(\vec{W})}{\partial \mu_{ic}} \cdot \frac{\partial \mu_{ic}}{\partial s_{ic}} \]

\[ s_{ic}(\hat{x}_i) = \vec{w}_c^T \hat{x}_i \]
Multi-Class LR

\[ \hat{\mu}_{ic} = P(y_i = c | \tilde{x}_i, \bar{W}) = \frac{\exp(\bar{w}_c^T \tilde{x}_i)}{\sum_{c'=1}^{C} \exp(\bar{w}_{c'}^T \tilde{x}_i)} = \frac{\exp(s_{ic})}{\sum_{c'=1}^{C} \exp(s_{ic'})} \]

\[ \frac{d\mu_{ic}}{ds_{ic'}} = -\frac{\exp(s_{ic}) \cdot \exp(s_{ic'})}{(\sum_{c'=1}^{C} \exp(s_{ic'}))^2} \]

Product Rule: \((fg)' = f'g + fg'\)

\[ \frac{d\mu_{ic}}{ds_{ic'}} = -\frac{\exp(s_{ic}) \cdot \exp(s_{ic'})}{\sum_{c'=1}^{C} \exp(s_{ic'}) \cdot \sum_{c'=1}^{C} \exp(s_{ic'})} \]

\[ \nabla_{\bar{w}_c} \mathcal{L}(\bar{W}) = \sum_{i=1}^{N} \frac{\partial \mathcal{L}(\bar{W})}{\partial \bar{w}_c} \cdot \frac{\partial s_{ic}}{\partial \bar{w}_c} \]

\[ \nabla_s \mathcal{L}(\bar{W}) = \sum_{c=1}^{C} \frac{\partial \mathcal{L}(\bar{W})}{\partial \mu_{ic}} \cdot \frac{\partial \mu_{ic}}{\partial s_{ic}} \]

\[ \hat{\mu}_{ic} = \frac{\exp(s_{ic})}{\sum_{c'=1}^{C} \exp(s_{ic'})} \]

\[ \frac{d\mu_{ic}}{ds_{ic'}} = -\mu_{ic} \cdot \mu_{ic'} \]

\[ s_{ic}(\tilde{x}_i) = \bar{w}_c^T \tilde{x}_i \]
Multi-Class LR

\[ \hat{\mu}_{ic} = P(y_i = c | \tilde{x}_i, \tilde{W}) = \frac{\exp(\tilde{w}_c^T \tilde{x}_i)}{\sum_{c' = 1}^{C} \exp(\tilde{w}_{c'}^T \tilde{x}_i)} = \frac{\exp(s_{ic})}{\sum_{c' = 1}^{C} \exp(s_{ic'})} \]

\[ \frac{d\mu_{ic}}{ds_{ic'}} = \begin{cases} \mu_{ic} (1 - \mu_{ic}) & c = c' \\ -\mu_{ic} \cdot \mu_{ic'} & c \neq c' \end{cases} \]

Combining these two expressions:

\[ \frac{d\mu_{ic}}{ds_{ic}} = \mu_{ic} (I_{cc'} - \mu_{ic'}) = \begin{cases} \mu_{ic} (1 - \mu_{ic}) & c = c' \\ -\mu_{ic} \cdot \mu_{ic'} & c \neq c' \end{cases} \]

Here \( I_{cc'} \) is an indicator function:

\[ I_{cc'} = \begin{cases} 1 & c = c' \\ 0 & c \neq c' \end{cases} \]
Multi-Class LR

\[
\hat{\mu}_{ic} = P(y_i = c \mid \tilde{x}_i, \mathbf{W}) = \frac{\exp(\mathbf{w}_c^T \tilde{x}_i)}{\sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^T \tilde{x}_i)} = \frac{\exp(s_{ic})}{\sum_{c'=1}^{C} \exp(s_{ic'})}
\]

\[
\mathcal{L}(\mathbf{W}) = -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic}
\]

We are computing \(\frac{d\mathcal{L}(\mathbf{W})}{ds_{ic}}\)

\[
\frac{d\mathcal{L}(\mathbf{W})}{d\mu_{ic}} = -\left(\frac{y_{ic}}{\mu_{ic}}\right)
\]

\[
\frac{d\mu_{ic}}{ds_{ic}} = \mu_{ic}(\mathbb{I}_{cc'} - \mu_{ic'})
\]
Multi-Class LR

\[
\frac{d\mathcal{L}(\mathbf{W})}{ds_{ic}} = \sum_{c=1}^{c} \frac{d\mathcal{L}(\mathbf{W})}{d\mu_{ic}} \cdot \frac{d\mu_{ic}}{ds_{ic}}
\]

\[
\frac{d\mathcal{L}(\mathbf{W})}{d\mu_{ic}} = -\left(\frac{y_{ic}}{\mu_{ic}}\right)
\]

\[
\frac{d\mu_{ic}}{ds_{ic}} = \mu_{ic}(\Pi_{cc'} - \mu_{ic'})
\]

\[
\frac{d\mathcal{L}(\mathbf{W})}{ds_{ic}} = \sum_{c=1}^{c} -\left(\frac{y_{ic}}{\mu_{ic}}\right)\mu_{ic}(\Pi_{cc'} - \mu_{ic'})
\]

\[
\frac{d\mathcal{L}(\mathbf{W})}{ds_{ic}} = -\sum_{c=1}^{c} y_{ic}(\Pi_{cc'} - \mu_{ic'})
\]
Multi-Class LR

\[ \hat{\mu}_{ic} = P(y_i = c | \mathbf{x}_i, \mathbf{W}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x}_i)}{\sum_{c'=1}^C \exp(\mathbf{w}_{c'}^T \mathbf{x}_i)} = \frac{\exp(s_{ic})}{\sum_{c'=1}^C \exp(s_{ic'})} \]

\[ \frac{dL(\mathbf{W})}{ds_{ic}} = - \sum_{c=1}^C y_{ic} (\mathbb{I}_{cc'} - \mu_{ic'}) \]

\[ \frac{dL(\mathbf{W})}{ds_{ic}} = - \sum_{c=1}^C y_{ic} \mathbb{I}_{cc'} + \sum_{c=1}^C y_{ic} \mu_{ic'} \]

\[ \frac{dL(\mathbf{W})}{ds_{ic}} = -y_{ic} + \mu_{ic} \]

\[ \frac{dL(\mathbf{W})}{ds_{ic}} = \mu_{ic} - y_{ic} \]

Using:

\[ \sum_{c=1}^C \mathbb{I}_{cc'} = 1 \]

\[ \sum_{c=1}^C y_{ic} = 1 \]

Observe the class summation is gone!
Multi-Class LR

• The **cross-entropy gradient** for class $c$:

$$\nabla \mathcal{L}(\vec{W}) = \sum_{i=1}^{N} \frac{d\mathcal{L}(\vec{W})}{ds_{ic}} \cdot \frac{ds_{ic}}{d\vec{w}_c} \cdot \vec{x}_i$$

**For each class $c$, the gradient for the weights in the $c$’th column**

$$\nabla \mathcal{L}(\vec{W}) = \sum_{i=1}^{N} (\mu_{ic} - y_{ic}) \cdot \vec{x}_i$$

$$\hat{\mu}_{ic} = P(y_i = c | \vec{x}_i, \vec{W}) = \frac{\exp(\vec{w}_c^T \vec{x}_i)}{\sum_{c'=1}^{C} \exp(\vec{w}_{c'}^T \vec{x}_i)} = \frac{\exp(s_{ic})}{\sum_{c'=1}^{C} \exp(s_{ic'})}$$

$$s_{ic}(\vec{x}_i) = \vec{w}_c^T \vec{x}_i$$
Multi-Class LR

- The cross-entropy gradient for class $c$:

$$\nabla_{\vec{w}_c} \mathcal{L}(\vec{W}) = \sum_{i=1}^{N} (\mu_{ic} - y_{ic}) \cdot \vec{x}_i$$

For all classes, cross-entropy gradient $g(\vec{W})$:

$$g(\vec{W}) = \vec{X}^T \cdot (\vec{\mu} - \vec{Y})$$

Dimension of $\vec{W}$: $d \times C$

Dimension of $g(\vec{W})$: $d \times N$

For each class $c$, the gradient for the weights in the $c$’th column

<table>
<thead>
<tr>
<th>$c$</th>
<th>$w_{1,c}$</th>
<th>$w_{2,c}$</th>
<th>$w_{3,c}$</th>
<th>$w_{4,c}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$w_{1,1}$</td>
<td>$w_{1,2}$</td>
<td>$w_{1,3}$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$w_{2,1}$</td>
<td>$w_{2,2}$</td>
<td>$w_{2,3}$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$w_{3,1}$</td>
<td>$w_{3,2}$</td>
<td>$w_{3,3}$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$w_{4,1}$</td>
<td>$w_{4,2}$</td>
<td>$w_{4,3}$</td>
<td></td>
</tr>
</tbody>
</table>
Multi-Class LR

- **Batch Gradient Descent** algorithm for multi-class logistic regression (*softmax*):

  \[ \mathcal{L}(\vec{W}) = - \frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic} \]

  \[ \nabla_{\vec{W}} \mathcal{L}(\vec{W}) = \frac{1}{N} \vec{X}^T (\vec{\mu} - \vec{Y}) \]

  **Initialize** \( \vec{W}^{(0)} = \vec{0} \)

  **Update rule:**

  \[ \vec{W}^{(t+1)} = \vec{W}^{(t)} - \eta \nabla_{\vec{W}^{(t)}} \mathcal{L}(\vec{W}^{(t)}) \]

  \[ \vec{W}^{(t+1)} = \vec{W}^{(t)} - \frac{\eta}{N} \vec{X}^T (\vec{\mu} - \vec{Y}) \]
Multi-Class vs Binary LR

- Notice that **similarity** of the batch GD update rule between the multi-class and binary logistic regression.

Multiclass

\[
\nabla_{\hat{\mathbf{w}}} \mathcal{L}(\hat{\mathbf{w}}) = \frac{1}{N} \hat{X}^T (\hat{\mu} - \hat{Y})
\]

\[
\hat{\mathbf{w}}^{(t+1)} = \hat{\mathbf{w}}^{(t)} - \eta \hat{X}^T (\hat{\mu} - \hat{Y})
\]

In the multiclass LR BGD: \( \hat{\mathbf{w}} \ (d \times C) \), \( \hat{\mu} \ (N \times C) \), and one-hot \( \hat{Y} \ (N \times C) \) are matrices.

Binary

\[
\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \frac{1}{N} \hat{X}^T \left[ \sigma(\hat{X}.\mathbf{w}^{(t)}) - \hat{Y} \right]
\]

\[
\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \frac{\eta}{N} \hat{X}^T \left[ \sigma(\hat{X}.\mathbf{w}^{(t)}) - \hat{Y} \right]
\]

In the binary LR BGD: \( \mathbf{w} \ (d \times 1) \), \( \mathbf{\sigma} \ (N \times 1) \), and \( \hat{Y} \ (N \times 1) \) are 1D vectors.

---

66
Linearly Non-Separable Data

- Logistic regression provides a linear classifier.
- How do we draw decision boundary for **linearly non-separable data**?
- Create a **new representation of features** by augmenting data with its polynomials, then it will become linearly separable.
• **Polynomial** Logistic Regression for nonlinear data.

Compute the score for an instance first by transforming the features using the **basis (polynomial) function**: 

\[ s_{ic} = \overrightarrow{w}_c^T \phi(\vec{x}_i) \]

\[
\hat{\mu}_{ic} = P(y_i = c \mid \phi(\vec{x}_i), \overrightarrow{W}) = \frac{\exp(\overrightarrow{w}_c^T \phi(\vec{x}_i))}{\sum_{c'=1}^{C} \exp(\overrightarrow{w}_{c'}^T \phi(\vec{x}_i))} = \frac{\exp(s_{ic})}{\sum_{c'=1}^{C} \exp(s_{ic'})}
\]
Multi-Class (Softmax) LR Batch Gradient Descent (no regularization): Implementation

- Implementation in python:
  1. Choose the degree of polynomial (for polynomial regression), then add polynomial & bias terms in X
  2. **Standardize** the features
  3. Choose the hyperparameter: $\eta$
  4. Initialize $\vec{W}$ (all zeros)
  5. Choose a **stopping criterion** by setting a **tolerance parameter** that stops iteration when the difference between the two consecutive $\mathcal{L}$’s goes below the tolerance value
  6. Iteratively apply the update rule until the stopping criterion is met (repeat steps 5 & 6)

$$\mathcal{L}(\vec{W}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic}$$

$$\vec{W}^{(t+1)} = \vec{W}^{(t)} - \frac{\eta}{N} \vec{X}^T (\vec{\mu} - \vec{Y})$$
Multi-class Classification: Batch GD Scikit-Learn

- For an *empirical understanding* of how to use Scikit-Learn’s Logistic Regression **Batch Gradient Descent** algorithm for solving a multi-class classification problem using both **OvA and Softmax regression**, see the following Jupyter notebook:
  - https://github.com/rhasanbd/Logistic-Regression-Comparative-Understanding/blob/master/Logistic%20Regression-3-Multiclass%20Classification-Batch%20Gradient%20Descent.ipynb
Multi-Class (Softmax) Logistic Regression: Regularized Batch GD
Multi-Class (Softmax) LR I2 Regularized Batch GD: Implementation

- Implementation in python:

1. Choose the degree of polynomial (for polynomial regression), then add polynomial & bias terms in \( X \)

2. **Standardize** the features

3. Choose the **hyperparameters**: \( \eta \) & \( \lambda \)

4. Initialize \( \vec{W} \) (all zeros)

5. Set up the regularized cross-entropy loss function:

\[
L(\vec{W}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic} + \frac{\lambda}{2} \sum_{j=1}^{d} \sum_{c=1}^{C} W_{jc}^2
\]

Bias is excluded from regularization
Multi-Class (Softmax) LR I2 Regularized
Batch Gradient Descent: Implementation

• Implementation in python:

6. Choose a **stopping criterion** by setting a **tolerance parameter** that stops iteration when the difference between the two consecutive $\mathcal{L}$’s goes below the tolerance value.

$$\vec{W}(t+1) = \vec{W}(t) - \frac{\eta}{N} [\vec{X}^T (\vec{\mu} - \vec{Y}) + \lambda \vec{W}(t)]$$

7. Iteratively apply the update rule until the stopping criterion is met (repeat steps 5 – 7).

Bias is excluded from regularization.
Multi-Class (Softmax) LR \textbf{l1 Regularized} Batch GD: Implementation

- Implementation in python:

1. Choose the degree of polynomial (for polynomial regression), then add polynomial & bias terms in X

2. \textbf{Standardize} the features

3. Choose the \textbf{hyperparameters}: $\eta$ & $\lambda$

4. Initialize $\vec{W}$ (all zeros)

5. Set up the regularized cross-entropy loss function:

$$\mathcal{L}(\vec{W}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic} + \lambda \sum_{j=1}^{d} \sum_{c=1}^{C} W_{jc}$$

- Bias is excluded from regularization
Multi-Class (Softmax) LR \textbf{I1 Regularized} Batch Gradient Descent: Implementation

- Implementation in python:

\[
\vec{W}^{(t+1)} = \vec{W}^{(t)} - \frac{\eta}{N} \left[ \bar{X}^T (\bar{\mu} - \bar{Y}) + \lambda \text{sign}(\vec{W}^{(t)}) \right]
\]

6. Choose a \textbf{stopping criterion} by setting a \textbf{tolerance parameter} that stops iteration when the difference between the two consecutive \( \mathcal{L} \)'s goes below the tolerance value

7. Iteratively apply the update rule until the stopping criterion is met (repeat steps 5 – 7)
Multi-Class (Softmax) Logistic Regression: SGD
Multi-Class (Softmax) Logistic Regression: Stochastic GD

- At each iteration we choose a random data point and update $\hat{W}$.

\[
\begin{align*}
\text{Initialize } & \hat{W}^{(0)} = \vec{0} \\
\hat{W}^{(t+1)} & = \hat{W}^{(t)} - \eta \langle \hat{x}_i^T, (\hat{\mu}_i - \hat{y}_i) \rangle
\end{align*}
\]

The subscript $i$ refers to the sample index.

\[
\mathcal{L}(\hat{W}) = - \sum_{c=1}^{C} y_{ic} \log \mu_{ic}
\]

One-hot vector $\hat{y}_i : 1 \times C$

Vector $\hat{\mu}_i : 1 \times C$

Initially the learning rate (LR) should be large, then reduce it by implementing a learning schedule.

Python code for various LR schedules:
Multi-class Classification: SGD Scikit-Learn

• For an *empirical understanding* of how to use Scikit-Learn’s Logistic Regression *Stochastic Gradient Descent* algorithm for solving a multi-class classification problem using the OvA approach, see the following Jupyter notebook:

  • [https://github.com/rhasanbd/Logistic-Regression-Comparative-Understanding/blob/master/Logistic%20Regression-4-Multiclass%20Classification-Stochastic%20Gradient%20Descent.ipynb](https://github.com/rhasanbd/Logistic-Regression-Comparative-Understanding/blob/master/Logistic%20Regression-4-Multiclass%20Classification-Stochastic%20Gradient%20Descent.ipynb)
An Interesting & Useful Regularization Technique
Validation of a Model

• Before training a model, we usually take the available data and **partition** it into:
  - A training set
  - A validation set (hold-out set)
  - A test set

• Typical ratio is: 80% - 10% - 10%
Validation of a Model

- Training set: used for two purposes:
  - To train a **range of models** (e.g., Logistic Regression, KNN)
  - To train a **given model** with a range of values for its complexity parameters (e.g., learning step, regularization strength).
Validation of a Model

- Validation set:
- Provide an unbiased **evaluation of a model fit on the training dataset during training.**

The validation set is also used to **optimize the model complexity** (e.g., to choose optimum learning step).
Validation of a Model

- Test set:
- Provides the **gold standard used to evaluate** the model.
- It is only used once a model is completely trained (using the train and validation sets).
A New Regularization Technique

- A very different way to **regularize** batch Gradient Descent is to **stop training** as soon as the validation error reaches a **minimum**.
- This is called **early stopping**.

Figure: a high-degree Polynomial Regression model trained using the Batch Gradient Descent.
Regularization Via Early Stopping

- As the epochs go by, the algorithm learns and its prediction loss (e.g., RMSE in this example) on the training set naturally goes down, and so does its prediction loss on the validation set.
- However, after a while the **validation loss stops decreasing** and actually starts to go back up.
Regularization Via Early Stopping

- This indicates that the **model has started to overfit** the training data.
- With early stopping we just **stop training as soon as the validation loss reaches the minimum**.
Regularization Via Early Stopping

- Because we start with *weights almost zero* and they move away as training continues.
- Stopping early corresponds to a *model with more weights close to zero* and effectively *fewer parameters*.

![Graph showing RMSE over epochs for validation and training sets, with an arrow indicating the best model at a certain epoch.](image-url)
Regularization Via Early Stopping

- With **Stochastic and Mini-batch Gradient Descent**, the error vs iterations (epochs) curves are **not so smooth**.
- It may be **hard to know** whether we have reached the minimum or not.
Regularization Via Early Stopping

- One solution is to stop only after the validation loss has been above the minimum for some epochs (when we are confident that the model will not do any better).

Then **roll back the model parameters** to the point where the **validation loss was at a minimum**.
Regularization Via Early Stopping

- It is such a **simple and efficient regularization** technique that Geoffrey Hinton called it a “**beautiful free lunch**.”
Early Stopping Regularization: SGD Scikit-Learn

• For an *empirical understanding* of how to apply the early stopping regularization with Scikit-Learn’s Logistic Regression **Stochastic Gradient Descent** algorithm, see the following Jupyter notebook:

  • [https://github.com/rhasanbd/Logistic-Regression-Comparative-Understanding/blob/master/Logistic%20Regression-4-Multiclass%20Classification-Stochastic%20Gradient%20Descent.ipynb](https://github.com/rhasanbd/Logistic-Regression-Comparative-Understanding/blob/master/Logistic%20Regression-4-Multiclass%20Classification-Stochastic%20Gradient%20Descent.ipynb)