Philipp Lintl
12152498
lintl.philipp@gmail.com

Homework Assignment 3

Machine Learning 1, 18/19

2018-11-29

1 Naive Bayes Text Classification

Consider a document classification task, that classifies your documents into K classes $\mathcal{C}_{\|}$. To do this. a Bag-of-Words representation of the training documents is chosen. The BOW is a vector, x_n of Dimensions D for each document indicating whether each word in the vocabulary appears in a document (**only presence matters, not frequency count**). So, $x_{ni} = 1$ indicates that word i is present withn document n ($x_{ni} = 0$: word i not present in document n). D is thought of as the Vocabulary size of the training data. Data situation emerges as:

- training set $\mathbf{X} \in \mathbb{R}^{N \times D}$ of word counts
- target matrix $\mathbf{T}$ consisting of row vectors $\mathbf{t}_n^T = (t_{n1}, \dots, 1, \dots t_{nk})$ and $\mathbf{t}_n^T = (0, \dots, 1, \dots 0)$
- if $n \in \mathcal{C}_i$, then the $\mathbf{t_n}^T$ vector has a 1 at position i
- we also know $p(\mathcal{C}_i) = \pi_i$ with constraint $\sum_{i=1}^K \pi_i = 1$

In this task, we model the word counts using Bernoulli distribution, yielding a conditional density of:

$$p(\mathbf{x}|\mathcal{C}_k, \theta_{1k}, \dots, \theta_{Dk}) = \prod_{d=1}^{D} \theta_{dk}^{x_d} (1 - \theta_{dk})^{1-x_d}$$

So each word is distributed according to the Bernoulli distribution with parameter θ_{dk} when conditioned on class $\mathcal{C}_k$.

1.1

Write down the data likelihood, $p(\mathbf{T}, \mathbf{X}|\theta)$ **without** independence assumption. Now, derive the data likelihood for the general k classes naive Bayes classifier, stating where you make use of the product rula and the naive Bayes assumption. You should wute the likelihood in terms of $p(x_d|\mathcal{C}_k)$, meaning you should not assume the explicit Bernoulli distribution.

The data likelihood without indepence assumption is as follows:

$$p(\mathbf{T}, \mathbf{X}|\theta) \overset{\text{i.i.d. observations } \mathbf{x}}{=} \prod_{n=1}^{N} p(\mathbf{x}_n, \mathbf{t}_n)|\theta) \overset{\text{product rule}}{=} \prod_{n=1}^{N} p(\mathbf{t}_n|\boldsymbol{\theta}) \cdot p(\mathbf{x}_n|\mathbf{t}_n, \boldsymbol{\theta})$$

For the general K NB classifier, with C_k denoting the class observation $\mathbf{x}_n$ is assigned to, this leads to:

$$p(\mathbf{x}, C_k|\boldsymbol{\theta}_k) \overset{\text{product rule}}{=} p(\mathbf{x}|C_k, \boldsymbol{\theta}_k) \cdot p(C_k|\boldsymbol{\theta}_k),$$

whereas $p(C_k|\boldsymbol{\theta}_k) = \pi_k$ and the independence assumption of features $\mathbf{x}$ leads to:

$$p(\mathbf{x}, C_k|\boldsymbol{\theta}_k) = \pi_k \cdot \prod_{d=1}^{D} p(\mathbf{x_d}|\theta_{dk}, C_k)$$

Plugging this into the overall data likelihood, we arrive at:

$$\begin{aligned}
p(\mathbf{T}, \mathbf{X}|\theta) &= \prod_{n=1}^{N} p(\mathbf{t}_n|\boldsymbol{\theta}) \cdot p(\mathbf{x}_n|\mathbf{t}_n, \boldsymbol{\theta}) \\
&= \prod_{n=1}^{N} \Big(p(t_{n1} = 1|\boldsymbol{\theta}) \cdot p(\mathbf{x}_n|t_{n1} = 1, \boldsymbol{\theta})^{\mathbb{1}(t_{n1}=1)} \cdot \ldots \cdot p(t_{nk} = 1|\boldsymbol{\theta}) \cdot p(\mathbf{x}_n|t_{nk} = 1, \theta)^{\mathbb{1}(t_{nk}=1)} \Big) \\
&= \prod_{n=1}^{N} \prod_{k=1}^{K} \Big(p(t_{nk} = 1|\boldsymbol{\theta}) \cdot p(\mathbf{x}_n|t_{nk} = 1, \boldsymbol{\theta})^{\mathbb{1}(t_{nk}=1)} \Big)
\end{aligned}$$

Applying the Naive Bayes assumption, yields:

$$p(\mathbf{T}, \mathbf{X}|\theta) = \prod_{n=1}^{N} \prod_{k=1}^{K} \left(\pi_k \cdot \prod_{d=1}^{D} p(x_{nd}|\theta_{dk}, t_{nk} = 1)^{\mathbb{1}(t_{nk}=1)} \right)$$

1.2

How does the number of parameters change if you make use of the naive Bayes assumption? Why is the assumption called *naive* and can you think of an example in which this assumption does not hold?

Referring to Bishop page 202, the non naive way resembles the case of discrete features and thus implies the parameter number of 2^D (one for each possible combination of D values t_{nk} and a class) for each class and as we are having K classes, $K \cdot 2^D$ parameters. The naive assumption means that feature values conditioned on a class k are assumed independent from each other. Thus, it is only necessary to estimate one parameter for each word-class combination, leaving $K \cdot D$ parameters to be estimated.
The assumption is called naive, as it completely ignores any correlation among words/features. This is a very simplifying assumption, as it is much more reasonable to assume correlated features given the class. Just assume a corpus of newspaper articles with the topics (sports, economy, politics, ...) as labels and each article as an observation with the appearing words as features. It is very likely, that words coming from a sports article are (cor)related, as e.g. 'soccer', 'goal', 'football', 'referee'. So they are not independent among each other given the class.

1.3

Write down the data log-likelihood $p(\mathbf{T}, \mathbf{X}|\boldsymbol{\theta})$ for the Bernoulli model.

At first, the log is applied to the likelihood from above:

$$\begin{aligned}
ln\left(p(\mathbf{T}, \mathbf{X}|\theta)\right) &= \sum_{n=1}^{N}\sum_{k=1}^{K}\left(\mathbb{1}(t_{nk}=1)\cdot\left(ln(\pi_k) + \sum_{d=1}^{D} ln\left(p(x_{nd}|\theta_{dk}, t_{nk}=1)\right)\right)\right) \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K}\left(\mathbb{1}(t_{nk}=1)\cdot\left(ln(\pi_k) + \sum_{d=1}^{D} x_{nd}\cdot ln\left(\theta_{dk}\right) + (1-x_{nd})\cdot ln\left(1-\theta_{dk}\right)\right)\right) \\
&= \sum_{k=1}^{K}\sum_{n\in C_k}^{N}\left(ln(\pi_k) + \sum_{d=1}^{D} x_{nd}\cdot ln\left(\theta_{dk}\right) + (1-x_{nd})\cdot ln\left(1-\theta_{dk}\right)\right)
\end{aligned}$$

whereas $p(x_{nd}|\theta_{dk}, t_{nk}=1) = \theta_{dk}^{x_{nd}}(1-\theta_{dk})^{1-x_{nd}}$

1.4

Solve for the MLE estimators for θ_{dk}. Express in your own words, how the result can be interpreted.

MLE requires deriving the log-likelihood for θ_{dk} and setting it zero to arrive at $\theta_{dk}^{ML} = \underset{\theta_{dk}}{argmax}\ ln\left(p(\mathbf{T}, \mathbf{X}|\theta)\right)$:

$$\begin{aligned}
\frac{\partial ln\left(p(\mathbf{T}, \mathbf{X}|\theta)\right)}{\partial\theta_{dk}} &= \sum_{n\in C_k}^{N}\frac{x_{nd}}{\theta_{dk}} - \frac{1-x_{nd}}{1-\theta_{dk}} \overset{!}{=} 0 \\
\longleftrightarrow & \sum_{n\in C_k}^{N}\frac{x_{nd}\cdot(1-\theta_{dk}) - \theta_{dk}\cdot(1-x_{nd})}{\theta_{dk}\cdot(1-\theta_{dk})} = 0 \\
\longleftrightarrow & \sum_{n\in C_k}^{N} x_{nd} - x_{nd}\theta_{dk} - \theta_{dk} + x_{nd}\theta_{dk} = 0 \\
\longleftrightarrow & \sum_{n\in C_k}^{N} x_{nd} - \theta_{dk} = 0 \\
\longleftrightarrow & \ \theta_{dk}^{ML} = \frac{1}{N_k}\sum_{n\in C_k}^{N} x_{nd}
\end{aligned}$$

In other words, the MLE estimate for parameter θ_{dk} is the fraction of observations/text documents which contain feature/word d, as x_{nd} only indicates the presence of a word within a document and not the frequency.

1.5

Write $p(C_1|\mathbf{x})$ for the general k naive Bayes classifier.

$$\begin{aligned} p(C_1|\mathbf{x}, \boldsymbol{\theta}_1) &= \frac{p(\mathbf{x}|C_1, \boldsymbol{\theta}_1) \cdot p(C_1|\boldsymbol{\theta}_1)}{\sum_{k=1}^K p(\mathbf{x}|C_k, \boldsymbol{\theta}_k) p(C_k|\boldsymbol{\theta}_k)} \\ &= \frac{\pi_1 \cdot \prod_{d=1}^D p(x_{nd}|C_1, \theta_{d1})}{\sum_{k=1}^K \pi_k \cdot \prod_{d=1}^D p(x_{nd}|C_k, \theta_{dk})} \end{aligned}$$

1.6

Write $p(C_1|\mathbf{x})$ for the Bernoulli model.
Plugging the Bernoulli model into the afore-derived $p(C_1|\mathbf{x})$, yields:

$$\begin{aligned} p(C_1|\mathbf{x}, \boldsymbol{\theta}_1) &= \frac{\pi_1 \cdot \prod_{d=1}^D p(x_{nd}|C_1, \theta_{d1})}{\sum_{k=1}^K \pi_k \cdot \prod_{d=1}^D p(x_{nd}|C_k, \theta_{dk})} \\ &= \frac{\pi_1 \cdot \prod_{d=1}^D \theta_{d1}^{x_{nd}} \cdot (1-\theta_{d1})^{1-x_{nd}}}{\sum_{k=1}^K \pi_k \cdot \prod_{d=1}^D \theta_{dk}^{x_{nd}} \cdot (1-\theta_{dk})^{1-x_{nd}}} \end{aligned}$$

1.7

For the Bernoulli model, express the conditions (inequalities) of the region where x is predicted to be C_1.
Provide linear inequalities on the form $\mathbf{x}^T\mathbf{a} > c$.

Compared to the practical, there are now K classes, instead of 3, which complicates the procedure a little.

$$p(C_1|\mathbf{x}, \boldsymbol{\theta}_1) > p(C_i|\mathbf{x}, \boldsymbol{\theta}_i) \quad \forall i \neq 1$$

$$\text{(fill in 1.6)} \quad \frac{\pi_1 \cdot \prod_{d=1}^D \theta_{d1}^{x_{nd}} \cdot (1-\theta_{d1})^{1-x_{nd}}}{\sum_{k=1}^K \pi_k \cdot \prod_{d=1}^D \theta_{dk}^{x_{nd}} \cdot (1-\theta_{dk})^{1-x_{nd}}} > \frac{\pi_i \cdot \prod_{d=1}^D \theta_{di}^{x_{nd}} \cdot (1-\theta_{di})^{1-x_{nd}}}{\sum_{k=1}^K \pi_k \cdot \prod_{d=1}^D \theta_{dk}^{x_{nd}} \cdot (1-\theta_{dk})^{1-x_{nd}}} \quad \forall i \neq 1$$

$$\text{(same denominator)} \quad \pi_1 \cdot \prod_{d=1}^D \theta_{d1}^{x_{nd}} \cdot (1-\theta_{d1})^{1-x_{nd}} > \pi_i \cdot \prod_{d=1}^D \theta_{di}^{x_{nd}} \cdot (1-\theta_{di})^{1-x_{nd}} \quad \forall i \neq 1$$

$$\text{(rearranging (no change of sign))} \quad \frac{\pi_1}{\pi_i} > \prod_{d=1}^D \frac{\theta_{di}^{x_{nd}} \cdot (1-\theta_{di})^{1-x_{nd}}}{\theta_{d1}^{x_{nd}} \cdot (1-\theta_{d1})^{1-x_{nd}}} \quad \forall k \neq i$$

$$\begin{aligned} \text{(ln monot.} \rightarrow\neq \text{signchange)} \quad ln\,\frac{\pi_1}{\pi_i} > \sum_{d=1}^D & x_{nd} \cdot ln\,(\theta_{di}) + (1-x_{nd}) ln\,(1-\theta_{di}) \\ & - x_{nd} \cdot ln\,(\theta_{d1}) - (1-x_{nd}) ln\,(1-\theta_{d1}) \end{aligned}$$

$$\text{(rearranging)} \quad ln\left(\frac{\pi_1}{\pi_i}\right) > \sum_{d=1}^D x_{nd} \cdot ln\left(\frac{\theta_{di}}{\theta_{d1}}\right) + (1-x_{nd}) \cdot ln\left(\frac{1-\theta_{di}}{1-\theta_{d1}}\right)$$

$$\text{(rearranging)} \quad ln\left(\frac{\pi_1}{\pi_i}\right) - \sum_{d=1}^D ln\left(\frac{1-\theta_{di}}{1-\theta_{d1}}\right) > \sum_{d=1}^D x_{nd} \cdot \left(ln\left(\frac{\theta_{di}}{\theta_{d1}}\right) - ln\left(\frac{1-\theta_{di}}{1-\theta_{d1}}\right)\right)$$

$$\text{(rearranging)} \quad ln\left(\frac{\pi_1}{\pi_i}\right) - \sum_{d=1}^D ln\left(\frac{1-\theta_{di}}{1-\theta_{d1}}\right) > \sum_{d=1}^D x_{nd} \cdot ln\left(\frac{\theta_{di}(1-\theta_{d1})}{\theta_{d1}(1-\theta_{di})}\right)$$

$$\text{(due to form } \mathbf{x}^T a > c\text{)} \quad ln\left(\frac{\pi_i}{\pi_1}\right) - \sum_{d=1}^D ln\left(\frac{1-\theta_{d1}}{1-\theta_{di}}\right) < \sum_{d=1}^D x_{nd} \cdot ln\left(\frac{\theta_{d1}(1-\theta_{di})}{\theta_{di}(1-\theta_{d1})}\right)$$

Similar to the practical, the lefthandside shall be denoted by

$$\mathbf{c}_{1i} = ln\left(\frac{\pi_i}{\pi_1}\right) - \sum_{d=1}^{D} ln\left(\frac{1-\theta_{d1}}{1-\theta_{di}}\right)$$

and the term multiplied to x by

$$\mathbf{a}_{1i} = \left[ln\left(\frac{\theta_{11}(1-\theta_{1i})}{\theta_{1i}(1-\theta_{11})}\right) \quad \cdots \quad ln\left(\frac{\theta_{D1}(1-\theta_{Di})}{\theta_{Di}(1-\theta_{D1})}\right)\right]^T.$$

This leads to the final form of:

$$\mathbf{x}_n^T \mathbf{a}_{1i} > \mathbf{c}_{1i}$$

1.8

So far we only considered feature vectors with discrete values, is it possible to have continous features? Argue why yes/no and if yes, what would be an ecample?

At the moment, we encountered integer and boolean features for textclassification naive Bayes. In my opinion it is. The only change would be a continous class conditional density $p(\mathbf{x}|\boldsymbol{\theta})$, such as the Gaussian. The remaining naive Bayes setup stays the same. An example within text classification would be to either transform the feature space with basis functions or for instance word-embeddings like word2vec. That way, we'd have a continous feature space (drastically lower dimensionality), which would be fed into the NB formulas.

2 Multi-class Logistic Regression and multilayer Perceptron

2.1

In this question we will be deriving the matrix form of the log-likelihood $\nabla_{\mathbf{w}_j} p(\mathbf{T}|\mathbf{\Phi}, \boldsymbol{w_1}, \dots, \boldsymbol{w_K})$. The following is stated:

$$y_k(\phi) = p(C_k|\phi) = \frac{exp(a_k)}{\sum_i exp(a_i)}$$

The steps to follow in order are:

- Write down the likelihood $p(\mathbf{T}|\mathbf{\Phi}, \boldsymbol{w_1}, \dots, \boldsymbol{w_K})$. Use the entries of **T** as selectors of the correct class.

$$\begin{aligned} p(\mathbf{T}|\mathbf{\Phi}, \boldsymbol{w_1}, \dots, \boldsymbol{w_K}) &= \prod_{n=1}^{N}\prod_{k=1}^{K} p(C_k|\phi_n)^{t_{nk}} \\ &= \prod_{n=1}^{N}\prod_{k=1}^{K} y_k(\phi_n)^{t_{nk}} \end{aligned}$$

- Write down the log-likelihood $ln\ p(\mathbf{T}|\mathbf{\Phi}, \boldsymbol{w_1}, \dots, \boldsymbol{w_K})$.

$$ln\ p(\mathbf{T}|\mathbf{\Phi}, \boldsymbol{w_1}, \dots, \boldsymbol{w_K}) = \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \cdot ln\ y_k(\phi_n)$$

- Compute the derivative $\frac{\partial y_k}{\partial \mathbf{w}_j}$.

$$\begin{aligned} \frac{\partial y_k}{\partial \mathbf{w}_j} &= \frac{\partial \frac{exp(a_k)}{\sum_i exp(a_i)}}{\partial \mathbf{w}_j} = \frac{\partial \frac{exp(\mathbf{w}_k^T\phi)}{\sum_i exp(\mathbf{w}_i^T)\phi}}{\partial \mathbf{w}_j} \\ (quotientrule) \quad &= \frac{exp(\mathbf{w}_k^T\phi)\cdot\frac{\partial \mathbf{w_k^T}\phi}{\partial \mathbf{w}_j}\cdot\sum_i exp(\mathbf{w_i^T}\phi)}{\left(\sum_i exp(\mathbf{w}_i^T\phi)\right)^2} - \frac{exp(\mathbf{w}_k^T\phi)\cdot exp(\mathbf{w}_j^T\phi)\cdot\frac{\partial \mathbf{w_j^T}\phi}{\partial \mathbf{w}_j}}{\left(\sum_i exp(\mathbf{w}_i^T\phi)\right)^2} \\ &= \underbrace{\frac{exp(\mathbf{w}_k^T\phi)\cdot\phi^T\cdot\mathbb{1}(k=j)}{\sum_i exp(\mathbf{w}_i^T\phi)}}_{y_k(\phi)\phi^T\mathbb{I}_{kj}} - \underbrace{\frac{exp(\mathbf{w}_k^T\phi)}{\sum_i exp(\mathbf{w}_i^T\phi)}\cdot\frac{exp(\mathbf{w}_j^T\phi)\cdot\phi^T}{\sum_i exp(\mathbf{w}_i^T\phi)}}_{y_k(\phi)\cdot y_j(\phi)\cdot\phi^T} \\ &= y_k(\phi)\phi^T\mathbb{I}_{kj} - y_k(\phi)\cdot y_j(\phi)\cdot\phi^T, \end{aligned}$$

with notation from above.

- Use the chain rule and the derivatives $\frac{\partial y_k}{\partial \mathbf{w}_j}$ to compute $\nabla_{\mathbf{w}_j} ln\ p(\mathbf{T}|\mathbf{\Phi}, \boldsymbol{w_1}, \dots, \boldsymbol{w_K})$.

$$\begin{aligned}
\nabla_{\mathbf{w}_j} ln\ p(\mathbf{T}|\mathbf{\Phi}, \boldsymbol{w_1}, \dots, \boldsymbol{w_K}) &= \frac{\partial}{\partial \mathbf{w}_j} \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \cdot ln\ y_k(\phi_n) \\
\text{(chain rule)} \quad &= \sum_{n=1}^{N} \sum_{k=1}^{K} \frac{t_{nk}}{y_k(\phi_n)} \cdot \frac{\partial}{\partial \mathbf{w}_j} y_k(\phi_n) \\
\text{(filling in derivative of before)} \quad &= \sum_{n=1}^{N} \sum_{k=1}^{K} \frac{t_{nk}}{y_k(\phi_n)} \cdot \left(y_k(\phi)\phi_n^T \mathbb{I}_{kj} - y_k(\phi_n) \cdot y_j(\phi_n) \cdot \phi_n^T \right) \\
\text{(cancelling out)} \quad &= \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \cdot (\phi_n^T \mathbb{1}_{kj} - y_j \phi_n^T) \\
\text{(rearranging)} \quad &= \sum_{n=1}^{N} \phi_n^T \underbrace{\sum_{k=1}^{K} t_{nk} \mathbb{1}_{kj}}_{t_{nj}} - \sum_{n=1}^{N} y_j(\phi_n)\phi_n^T \cdot \underbrace{\sum_{k=1}^{K} t_{nk}}_{=1} \\
\text{(identity and one-hot properties)} \quad &= \sum_{n=1}^{N} (t_{nj} - y_j(\phi_n))\phi_n^T
\end{aligned}$$

- Your gradient is now a sum, can you rewrite it as a matrix multiplication? Why is this useful?

  For algorithm formulations, as in 2.3 it can be useful to have matrix formulation insted of possibly tedious sum formulations. Also from a computational standpoint, it is useful, as matrix multiplications are much more efficient than looping over operations for each of the variables.

$$\nabla_{\mathbf{w}_j} ln\ p(\mathbf{T}|\mathbf{\Phi}, \boldsymbol{w_1}, \dots, \boldsymbol{w_K}) = \sum_{n=1}^{N} (t_{nj} - y_j(\phi_n))\phi_n^T = (\mathbf{T}_j - \mathbf{Y}_j)^T \cdot \mathbf{\Phi}$$

  whereas $\mathbf{\Phi} \in \mathbb{R}^{N \times M}$ with definition as always upto this point, $\mathbf{T} \in \mathbb{R}^{N \times K}$, specifically $\mathbf{T}_j$ denotes the j-th column of the matrix which incorporated one-hot encoding in each row for each data point. $\mathbf{Y}_j$ represents the j-th column of a matrix **Y**, which has for instance row i: $\mathbf{Y}_{i,:} = (y_1(\phi_i), \dots, y_K(\phi_i))$ and column j: $\mathbf{Y}_{:,j} = (y_j(\phi_1), \dots, y_j(\phi_n))^T$. That way, multiplying a $\mathbb{R}^{1 \times N}$ vecor ($(\mathbf{T}_j - \mathbf{Y}_j)^T$) with a $\mathbb{R}^{N \times M}$ matrix $\mathbf{\Phi}$, yields a $\mathbf{R}^{1 \times M}$ gradient, which is exactly what we were looking for.

2.2

Write down the negative log-likelihood, this will be our objective function. Can you recognize the function you obtain? What is the relationship between the two (the function you obtain and the original log-likelihood)

$$-ln\ p(\mathbf{T}|\mathbf{\Phi}, \boldsymbol{w_1}, \dots, \boldsymbol{w_K}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \cdot ln\ y_k(\phi_n)$$

This objective function corresponds to the cross entropy error $E(\mathbf{w}_1, \dots, \mathbf{w}_K)$. The cross entropy error function simply is the negative log-likelihood. For optimization, this means, that for the cross entropy error, it is to be **minimized**, whereas the log-likelihood itself is to be **maximized**. The gradient subject to optimization is except for the sign change the same.

2.3

Write a mini-batch gradient algorithm for logistic regression using this objective function. Let B be the batch size n_B number of Batches and $E_n = -\sum_{k=1}^{K} t_{nk} ln\ y_k(\phi_n)$ the objective evaluated on data point n. Make sure to include indices for time, to define the learning rate and check what dimension the weights you are updating have, transposing the gradient if necessary. Name one advantage of mini-batch gradient descent over stochastic gradient descent with single data points. How dies it compare to full bath gradient descent?

```
1) initialization of $\mathbf{w}_1, \dots, \mathbf{w}_K$
2) initialization of $\eta$
3) initializaiton of $n_B$
for $\tau = 1$ to $n_B$ do
    choose $B$ data points randomly such that B rows from $\mathbf{\Phi}_B \subset \mathbf{\Phi}$, as well as according $\mathbf{T}_B \subset \mathbf{T}$ ;
    calculate $\mathbf{Y}_B$;
    $\mathbf{w}_j^{(\tau+1)} = \mathbf{w}_j^{(\tau)} - \eta(\frac{1}{B}(\mathbf{T}_{B,j} - \mathbf{Y}_{B,j}^{(\tau)})^T \cdot \mathbf{\Phi}_B) \quad (\forall j = 1, \dots, K)$
    Decrease $\eta$
    return $\mathbf{w}_1 \dots, \mathbf{w}_K$
end
```

Algorithm 1: Mini-batch gradient descent for logistic regression

with properties: $\mathbf{Y}_B = \begin{bmatrix} y_1(\phi_1) & \dots & y_K(\phi_1) \\ \vdots & \ddots & \vdots \\ y_1(\phi_B) & \dots & y_K(\phi_B) \end{bmatrix}$, which also depends on the weights obtained in the previous round, as $y_j(\phi_n) = \frac{exp(\mathbf{w}_k^T \phi)}{\sum_i exp(\mathbf{w}_i^T \phi})$. Also $\mathbf{\Phi}_B$ refers to the matrix only consisting of the B rows of the original $\mathbf{\Phi}$ randomly chosen for that batch. And lastly, the errors are averaged over the number of data points within that batch (B). Regarding the shape of the weights, as described in 2.1, its $\mathbb{R}^{1 \times M}$ for each $\mathbf{w}_j$.

One advantage over single point SGD, is that fewer updates to the model have to be made. This implies more computational efficiency than stochastic gradient descent, as well as allows for parallel implementations, as the calculation of the prediction error and the model update are separated, which means even more efficiency compared to the single point SGD.
The mini-batching leads to the case that not the entire training data needs to be stored in the memory for calculation of the updates and thus is computationally less costly. On the other side, mini-batch inhabits the a hyperparameter “mini-batch size”, which needs to be set and thus opens up room for decision.

2.4

Consider now the Multilayer Perceptron in Fig 1. with input features of dimension 2, hidden units and 3 output classes. For which weight values and type of activation functions can we fall back to the multiclass logistic regression?

If the ouput layers activation function is set to be softmax and the activation function of the hidden layer is linear, it is the case of logistic regression. In terms of the weights, in my opinion, they should be fixed and non zero, as otherwise no information of x is passed through. Fixed, as they otherwise would have to be estimated. That way, z_1 and z_2 are, due to the linear activation function of the hidden layer, just linear combinations of x. Thats the same case as in logistic regression, as the output layer represents y_k in the same way

2.5

Consider again the given network. We now consider the case where the activation function on the hidden layer is a ReLU, the activation for the output is a Softmax and we consider again the cross-entropy loss E. The weights of the network are initialized as follows:

$$\mathbf{W}_1 = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} = \begin{bmatrix} 0.4 & 0.87 \\ 0.58 & 0.34 \end{bmatrix} \quad \mathbf{W}_2 = \begin{bmatrix} w_5 & w_6 \\ w_7 & w_8 \\ w_9 & w_{10} \end{bmatrix} = \begin{bmatrix} 0.12 & 0.87 \\ 0.82 & 0.31 \\ 0.77 & 0.9 \end{bmatrix}$$

Perform the following steps.

- Compute the forward pass for the data point $x = (0.3, 0.7)^T$ with label $y = (0, 0.1)^T$. Evaluate the loss.

  In order to do that, we need to identify the necessary steps. First, the input $\mathbf{x}$ is multiplied with the weights $\mathbf{W}_1$ and then plugged into the activation function of the hidden layer: $h^{(1)}(x) = max\{0, x\}$

(ReLU), which leads to the following:

$$\mathbf{W}_1\mathbf{x} = \begin{bmatrix}0.4 & 0.87\\0.58 & 0.34\end{bmatrix}\begin{bmatrix}0.3\\0.7\end{bmatrix} = \begin{bmatrix}0.4\cdot 0.3 + 0.87\cdot 0.7\\0.58\cdot 0.3+0.34\cdot 0.7\end{bmatrix} = \begin{bmatrix}0.729\\0.412\end{bmatrix}$$
$$\Rightarrow h^{(1)}(\mathbf{W}_1\mathbf{x}) = \begin{bmatrix}0.729\\0.412\end{bmatrix}$$

Next, the obtained result is multiplied to $\mathbf{W}_2$ and fed into the activation function of the output layer, here representing the k-th output class y_k: $h_k^{(2)}(\mathbf{x}) = \frac{exp(x_k)}{\sum_i exp(x_i)}$. Leading to:

$$\mathbf{W}_2\cdot h^{(1)}(\mathbf{W}_1\mathbf{x}) = \begin{bmatrix}0.12 & 0.87\\0.82 & 0.31\\0.77 & 0.9\end{bmatrix}\cdot\begin{bmatrix}0.729\\0.412\end{bmatrix} = \begin{bmatrix}0.12\cdot 0.729+0.87\cdot 0.412\\0.82\cdot 0.729+0.31\cdot 0.412\\0.77\cdot 0.729+0.9\cdot 0.412\end{bmatrix} = \begin{bmatrix}0.446\\0.726\\0.932\end{bmatrix}$$
$$\Rightarrow h^{(2)}(\mathbf{W}_2\cdot h^{(1)}(\mathbf{W}_1\mathbf{x})) = \frac{1}{exp(0.446)+exp(0.726)+exp(0.932)}\cdot\begin{bmatrix}exp(0.446)\\exp(0.726)\\exp(0.932)\end{bmatrix} = \frac{1}{6.168}\cdot\begin{bmatrix}1.562\\2.067\\2.540\end{bmatrix}$$
$$= \begin{bmatrix}0.253\\0.335\\0.412\end{bmatrix},$$

which represents the output layer for the first pass-forward of the given **x**.
The loss is calculated by the the cross-entropy error:

$$E = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\cdot ln\; y_k(\phi_n)$$
$$\text{(in this case)} = -\sum_{k=1}^{K} t_n\cdot ln\; y_k(\phi_n)$$
$$= -(0\cdot ln0.253 + 0\cdot ln0.335 + 1\cdot ln(0.412)) = 0.887$$

- Compute the derivative $\frac{\partial E}{\partial \mathbf{w}_5}$. Write down the formula first and then compute the numerical result.
  The general form was shown before to be: $\frac{\partial E}{\partial \mathbf{w}_j} = \sum_{n=1}^{N}(y_j(\phi_n)-t_{nj})\phi_n^T = (\mathbf{Y}_j - \mathbf{T}_j)^T\cdot\mathbf{\Phi}$. As we only have one data point **x** at this point, the sum over n disappears and ϕ. The left two terms are known by the difference: $(y_1 - t_1)$.
  Which means for the task at hand: When referring to $h_1^{(2)}(\mathbf{W}_2\cdot h^{(1)}(\mathbf{W}_1\mathbf{x}))$ as $y_1(\mathbf{x},\mathbf{W_1},\mathbf{W_2})$, then:

$$\frac{\partial E}{\partial w_5} = (y_1 - t_1)\cdot z_1$$
$$= (0.253 - 0)\cdot 0.729 = 0.184$$

  Also, ϕ_n turns into z_1 at this point, as $\frac{\partial a_1}{\partial w_5} = \frac{\partial w_5\cdot z_1 + w_6\cdot z_2}{\partial w_5} = z_1$

- the other derivatives are given by:

$$\begin{bmatrix}\frac{\partial E}{\partial w_1} & \frac{\partial E}{\partial w_2}\\ \frac{\partial E}{\partial w_3} & \frac{\partial E}{\partial w_4}\end{bmatrix} = \begin{bmatrix}-0.044 & -0.103\\-0.062 & -0.144\end{bmatrix}$$
$$\begin{bmatrix}\frac{\partial E}{\partial w_5} & \frac{\partial E}{\partial w_6}\\ \frac{\partial E}{\partial w_7} & \frac{\partial E}{\partial w_8}\\ \frac{\partial E}{\partial w_9} & \frac{\partial E}{\partial w_{10}}\end{bmatrix} = \begin{bmatrix}\mathbf{0.184} & 0.104\\0.244 & 0.138\\-0.429 & -0.242\end{bmatrix}$$

Perform a weight update with learning rate 0.05.
Using the gradient descent formula:

$$\mathbf{w}_j^{(\tau+1)} = \mathbf{w}_j^{(\tau)} - \eta(\nabla_{w_j}E)$$
$$\mathbf{w}_j^{(\tau+1)} = \mathbf{w}_j^{(\tau)} - \eta\left((\mathbf{Y}_j - \mathbf{T}_j)^T\cdot\mathbf{\Phi}\right)$$

WIth the gradients given in the task before, the weight updates are given by:

$$\mathbf{w}_1^{(2)} = \begin{bmatrix} 0.4 & 0.87 \\ 0.58 & 0.34 \end{bmatrix} - 0.05 \cdot \begin{bmatrix} -0.044 & -0.103 \\ -0.062 & -0.144 \end{bmatrix} = \begin{bmatrix} 0.402 & 0.875 \\ 0.583 & 0.347 \end{bmatrix}$$

$$\mathbf{w}_2^{(2)} = \begin{bmatrix} 0.12 & 0.87 \\ 0.82 & 0.31 \\ 0.77 & 0.9 \end{bmatrix} - 0.05 \cdot \begin{bmatrix} \mathbf{0.184} & 0.104 \\ 0.244 & 0.138 \\ -0.429 & -0.242 \end{bmatrix} = \begin{bmatrix} 0.111 & 0.865 \\ 0.808 & 0.303 \\ 0.791 & 0.912 \end{bmatrix}$$

- compute the loss again, what happens?

  In order to compute the loss, at first another pass forward needs to be computed:

$$\mathbf{W}_1^{(2)}\mathbf{x} = \begin{bmatrix} 0.402 & 0.875 \\ 0.583 & 0.347 \end{bmatrix} \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix} = \begin{bmatrix} 0.733 \\ 0.418 \end{bmatrix}$$
$$\Rightarrow h^{(1)}(\mathbf{W}_1\mathbf{x})) = \begin{bmatrix} 0.733 \\ 0.418 \end{bmatrix}$$

Next, the obtained result is multiplied to $\mathbf{W}_2$ and fed into the activation function of the output layer, leading to:

$$\mathbf{W}_2^{(2)} \cdot h^{(1)}(\mathbf{W}_1^{(2)}\mathbf{x})) = \begin{bmatrix} 0.111 & 0.865 \\ 0.808 & 0.303 \\ 0.791 & 0.912 \end{bmatrix} \cdot \begin{bmatrix} 0.733 \\ 0.418 \end{bmatrix} = \begin{bmatrix} 0.443 \\ 0.719 \\ 0.961 \end{bmatrix}$$

$$\Rightarrow h^{(2)}(\mathbf{W}_2^{(2)} \cdot h^{(1)}(\mathbf{W}_1^{(2)}\mathbf{x}))) = \frac{1}{exp(0.443) + exp(0.719) + exp(0.961)} \cdot \begin{bmatrix} exp(0.443) \\ exp(0.719) \\ exp(0.961) \end{bmatrix} = \frac{1}{6.224} \cdot \begin{bmatrix} 1.557 \\ 2.052 \\ 2.614 \end{bmatrix}$$
$$= \begin{bmatrix} 0.250 \\ 0.330 \\ 0.420 \end{bmatrix},$$

which represents the output layer for the second pass-forward of the given $\mathbf{x}$.
The loss is calculated by the the cross-entropy error:

$$E = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \cdot ln\ y_k(\phi_n)$$
$$\text{(in this case)} = -\sum_{k=1}^{K} t_n \cdot ln\ y_k(\phi_n)$$
$$= -(0 \cdot ln 0.250 + 0 \cdot ln 0.330 + 1 \cdot ln(0.420)) = 0.868$$

The error decreased from 0.887 to 0.868.