DEEP LEARNING

Supervised by:

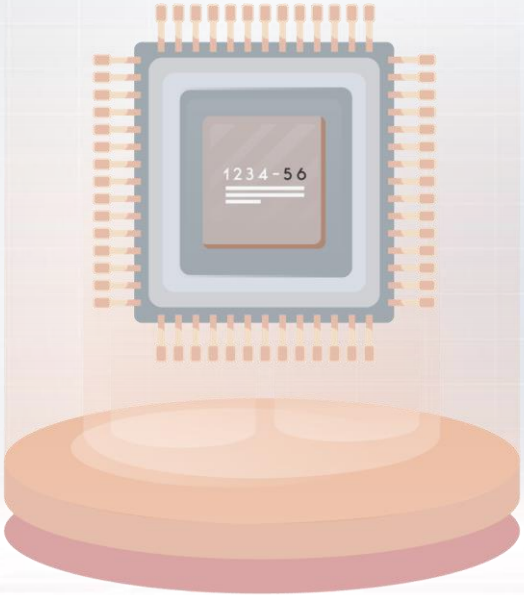
Pr. Belcaid Anass

Multiclass Support Vector Machine loss

Prepared by: **KINDA Abdoul Latif**

Date: 10/09/2024

TABLE OF CONTENTS



01

INTRODUCTION

Of SVM

INTRODUCTION

In this section, we will introduce the **Multiclass Support Vector Machine (SVM) loss**, a commonly used function in machine learning for classification tasks. The idea behind the SVM loss is to ensure that the classifier assigns a higher score to the correct class than to any incorrect classes, with a specified margin, denoted as Δ (delta). This margin is designed to encourage the classifier to clearly separate correct classifications from incorrect ones.

To better understand this, we can anthropomorphize the concept: the SVM "wants" the score for the correct class to be higher by at least Δ than the scores for all other incorrect classes. Achieving this goal minimizes the loss, which leads to better classification performance.

For each data point x_i (the pixels of the i-th image) and its corresponding correct class label y_i the SVM computes a score vector **s**, which is the result of the score function $f(x_i,W)$. This score vector **s** contains a score $\mathbf{s}_j$ for each class, with

$$\mathbf{s}_j = f(xi,W)_j$$

where j refers to the j-th class.

The SVM loss is structured so that, for each image, the score for the correct class should be higher than the scores for the other classes by at least a fixed margin Δ. This forms the basis for the **Multiclass SVM loss**, which encourages the model to create a large margin between the correct and incorrect classifications.

FORMULE DEFINITION

$$Li = \sum_{j \neq yi} max(0, sj - syi + \Delta)$$

With linear score functions ($f(xi, W) = W_{xi}$) so we can also rewrite the loss function in this equivalent form:

Loss functions:

Linear scores functions

$$Li = \sum_{j \neq yi} max(0,\ WjTxi\ - WyiTxi + \Delta)$$

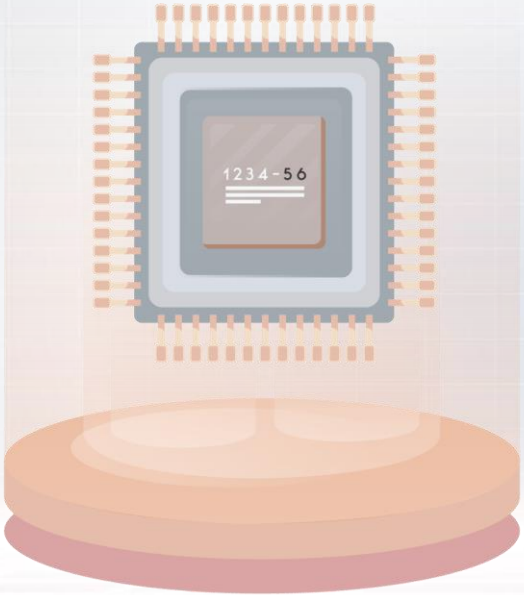
Where w_j is the j-th row of W reshaped as colum

Hinge Loss:

max(0, –) function is also called the **hinge loss**.

However you could also see the use of the squared hinge loss SVM (or L2-SVM) which is: **max(0, –)^2** that penalizes violated margins more **strongly** (quadratically instead of linearly). The unsquared version is more standard, but in some datasets the squared hinge loss can work better. This can be determined during cross-validation.

The Multiclass Support Vector Machine "wants" the score of the correct class to be higher than all other scores by at least a margin of delta. If any class has a score inside the red region (or higher), then there will be accumulated loss. Otherwise the loss will be zero. Our objective will be to find the weights that will simultaneously satisfy this constraint for all examples in the training data and give a total loss that is as low as possible.



02

REGULARIZATION

Of SVM

There is one bug with the loss function we presented above. Suppose that we have a dataset and a set of parameters W that correctly classify every example (i.e. all scores are so that all the margins are met, and $L_i=0\ \forall i$).

The issue is that this set of W is not necessarily unique: there might be many similar W that correctly classify the examples. One easy way to see this is that if some parameters W correctly classify all examples (so loss is zero for each example), then any multiple of these parameters λW where $\lambda>1$ will also give zero loss because this transformation uniformly stretches all score magnitudes and hence also their absolute differences.

For example, if the difference in scores between a correct class and a nearest incorrect class was 15, then multiplying all elements of W by 2 would make the new difference 30.

In other words, we wish to encode some preference for a certain set of weights WW over others to remove this ambiguity. We can do so by extending the loss function with a **regularization penalty** R(W)R(W) . The most common regularization penalty is the squared L2L2-norm that discourages large weights through an elementwise quadratic penalty over all parameters:

$$\boldsymbol{R(W) = \sum_k \sum_l W_{k\ l}^2}$$

In the expression above, we are summing up all the squared elements of W . Notice that the regularization function is not a function of the data, it is only based on the weights

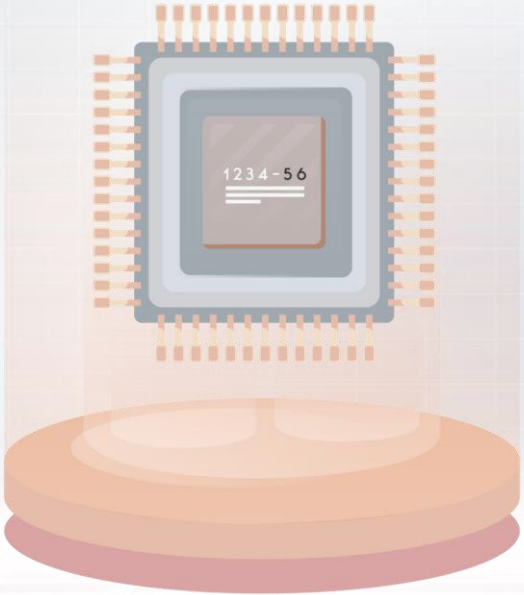
Including the regularization penalty completes the full Multiclass Support Vector Machines loss, which is made up of two components: the data loss (which is he average loss Li over all examples) and the regularization loss. That is, the full Multiclass SVM loss becomes

$$L = (1 \div N) \sum_i L_i + \sum_k \sum_l W^2{}_{k,l}$$

Data loss *regularization loss*

Or in the full form:

$$L = (1 \div N) \sum_i \sum_{j \neq yi} max(0, sj - syi + \Delta) + \sum_k \sum_l W^2{}_{k,l}$$



03

SOFTMAX

& SVM vs Softmax

It turns out that the SVM is one of two commonly seen classifiers. The other popular choice is the Softmax classifier, which has a different loss function. If you've heard of the binary Logistic Regression classifier before, the Softmax classifier is its generalization to multiple classes. Unlike the SVM which treats the outputs **f(xi,W)** as (uncalibrated and possibly difficult to interpret) scores for each class, the Softmax classifier gives a slightly more intuitive output (normalized class probabilities) and also has a probabilistic interpretation that we will describe shortly. In the Softmax classifier, the function mapping f(xi;W)=Wxi stays unchanged, but we now interpret these scores as the unnormalized log probabilities for each class and replace the hinge loss with a cross-entropy loss that has the form:

$$Li = log\left(\frac{e^{f_{yi}}}{\sum_j e^{f_j}}\right) or\ Li = -fyi + log\sum_j e^{f_j}$$

Where we are using the notation fj to mean the j-th element of the vector of class scores f. As before, the full loss for the dataset is the mean of Li over all trainings exmples together with a regularization term R(W).

The function $f_j(z) = \frac{e^z}{\sum_k e^{z_k}}$ is called the softmax function: it takes a vector of arbitrary real-valued scores (in z) and squashes it to a vector of values between zero and one that sum to one. The full cross-entropy loss that involves the softmax function might look scary if you're seeing it for the first time but it is relatively easy to motivate

Probabilistic interpretation

$$P(yi \mid xi; W) = \frac{e^{f_{yi}}}{\sum_k e^{fk}}$$

can be interpreted as the (normalized) probability assigned to the correct label yi given the image xi and parameterized by W . To see this, remember that the Softmax classifier interprets the scores inside the output vector f as the unnormalized log probabilities. Exponentiating these quantities therefore gives the (unnormalized) probabilities, and the division performs the normalization so that the probabilities sum to one

Probabilistic interpretation

In the probabilistic interpretation, we are therefore minimizing the negative log likelihood of the correct class, which can be interpreted as performing Maximum Likelihood Estimation (MLE). A nice feature of this view is that we can now also interpret the regularization term R(W) in the full loss function as coming from a Gaussian prior over the weight matrix W , where instead of MLE we are performing the Maximum a posteriori (MAP) estimation. We mention these interpretations to help your intuitions, but the full details of this derivation are beyond the scope of this class.

Pratical issues

Numeric stability. When you're writing code for computing the Softmax function in practice, the intermediate terms $\mathbf{e^{fyi}}$ and $\sum_j \mathbf{e^{fj}}$ may be very large due to the exponentials. Dividing large numbers can be **numerically unstable**, so it is important to use a **normalization trick**. Notice that if we multiply the top and bottom of the fraction by a constant **C** and push it into the sum, we get the following (mathematically equivalent) expression

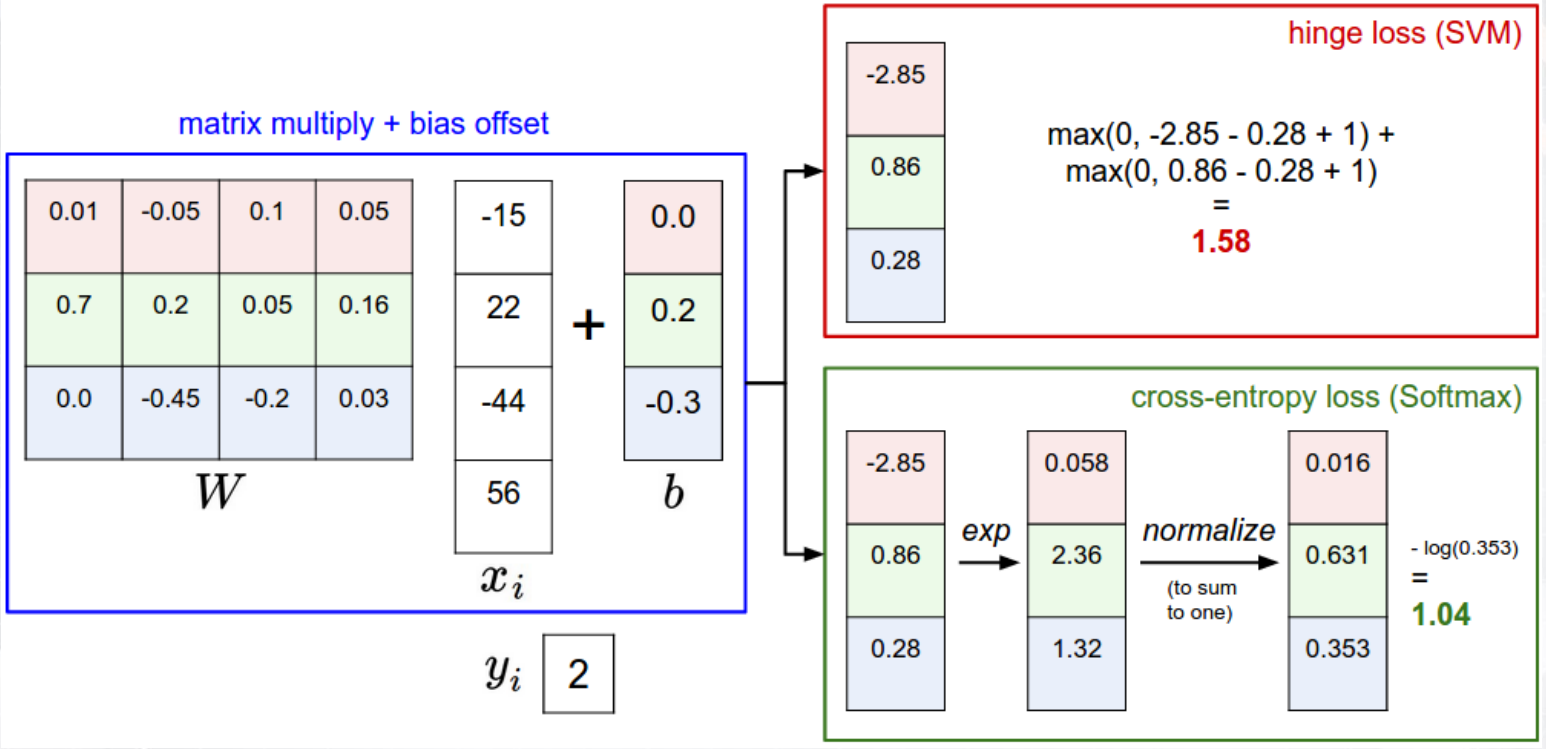
$$\frac{e^{f_{yi}}}{\sum_k e^{f_k}} = \frac{Ce^{f_{yi}}}{C\sum_k e^{f_k}} = \frac{e^{f_{yi}+logC}}{\sum_k e^{f_k+logC}}$$

We are free to choose the value of **C** . This will not change any of the results, but we can use this value to improve the **numerical stability** of the computation. A common choice for **C** is to set $\mathbf{logC=-max_j f_j}$. This simply states that we should shift the values inside the vector f so that the highest value is zero. In code:

```
f = np.array([123, 456, 789]) # example with 3 classes and each having large sc
p = np.exp(f) / np.sum(np.exp(f)) # Bad: Numeric problem, potential blowup

# instead: first shift the values of f so that the highest number is 0:
f -= np.max(f) # f becomes [-666, -333, 0]
p = np.exp(f) / np.sum(np.exp(f)) # safe to do, gives the correct answer
```

SVM vs SOFTMAX



Example of the difference between the SVM and Softmax classifiers for one datapoint. In both cases we compute the same score vector f (e.g. by matrix multiplication in this section). The difference is in the interpretation of the scores in f: The SVM interprets these as class scores and its loss function encourages the correct class (class 2, in blue) to have a score higher by a margin than the other class scores. The Softmax classifier instead interprets the scores as (unnormalized) log probabilities for each class and then encourages the (normalized) log probability of the correct class to be high (equivalently the negative of it to be low). The final loss for this example is 1.58 for the SVM and 1.04 (note this is 1.04 using the natural logarithm, not base 2 or base 10) for the Softmax classifier, but note that these numbers are not comparable; They are only meaningful in relation to loss computed within the same classifier and with the same data.