Optimal Transport - Report
Learning with a Wasserstein Loss

Nicolás Violante

MVA 2021-2022

Abstract

The Wasserstein distance provides a natural way of comparing two probability vectors. In their 2015 article *Learning with a Wasserstein Loss*, Fogner et al. [5] proposed to use this distance as a loss to train linear logistic regression models for classification using gradient descent in cases where the labels present a natural semantic relation. This was the first time the Wasserstein distance was used for supervised learning. The goal of this report is to evaluate their scheme and use it to train neural networks for ordinal classification, i.e. classification where the labels have a natural ordering. We use the Wasserstein distance as the loss function between the outputs of the network and the targets and test its performance on different scenarios. We also compare its performance against models trained with the widely used Cross Entropy loss.

Contents

1 Introduction

1.1 Presentation of the problem

The multi-class classification problem consists in assigning a label $y \in \{\mathcal{C}_1, \dots, \mathcal{C}_K\}$ from a set with K categories to an input vector $x \in \mathbb{R}^n$. A particular case of interest is ordinal classification [8] [6], where there exists a natural ordering between the labels, $\mathcal{C}_1 \prec \mathcal{C}_2 \cdots \prec \mathcal{C}_K$. The ordering in the labels does not imply the existence of a meaningful numerical difference between them.

Ordinal classification has many applications in practice and it often involves human-made scales. For example, consider a classification system based on movie reviews. Here the set of movie ratings as bad, regular, good, excellent is an ordered labelling. The miss-classification errors have different importance. Miss-classify a bad movie as excellent is much worse than miss-classify it as regular. This makes it natural to introduce a notion of cost depending on the miss-classification error type. This cost is related to the natural distance between the labels, also called ground truth metric.

1.2 Related work

Some naive approaches to ordinal classification are recasting the problem as a regression task where the labels are converted into real values $y_i \in \mathbb{R}$ or using nominal classification algorithms by discarding the ordering information in the labels, for example classification trees. The main drawback of this last methods is that they completely ignore the underlying ground truth metric in the labels. An example of improvement on this regard are ordinal classification trees [12]. Kotsianti and Pintelas [10] introduce a cost matrix $\lambda \in \mathbb{R}_+^{K \times K}$ that penalizes classifying a true label j to a predicted label i with a weight λ_{ij}. Correct classification has no cost $\lambda_{ii} = 0$. This is beneficial for ordinal regression because we explicitly take into account the ground truth metric. Then they minimize a weighted empirical risk. Their method is powerful because it can be paired with any nominal classification algorithm that outputs a probability vector. For an exhaustive and detailed survey of different ordinal classification approaches please refer to Gutierrez et al [8].

More recent approaches based on deep learning train neural networks to output a probability vector $h_\theta(x) \in \mathbb{R}_+^K$, where θ are the learnable parameters of the network. The final prediction is the class with maximum probability in the output, $\arg\max_k h_\theta(x)$. A common choice for the loss function for multi-class classification is the Cross Entropy function $H(p, q) = -\sum_x p(x) \log q(x)$.

1.3 Article contributions

The Optimal Transport framework automatically lifts the ground truth metric to a distance between probability vectors, the Wasserstein distance. Fogner et al. [5] proposed using this distance $W(h_\theta(x), y)$ between the model outputs $h_\theta(x)$ and the one-hot targets y as a loss function. This was the first paper to use the Wasserstein distance for training supervised-learning models. In particular, they used the Wasserstein loss to train linear logistic regression models for classification. The extension to neural networks is straightforward. At each

training step, the parameters θ of the network are updated in order to minimize the average empirical cost

$$\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} W(h_\theta(x_i), y_i) \tag{1}$$

using backpropagation. Fatras et al. [4] study convergence properties of taking the average Wasserstein distance across examples. Our experiments empirically confirms that using the Wasserstein loss this way is effective for training purposes and achieves good generalization.

The following is a brief description of the most relevant theoretical foundations used in *Learning with a Wasserstein Loss*. Given a cost matrix $C \in \mathbb{R}^{n \times m}$, Kantorovich's formulation of optimal transport aims to find an optimal transportation plan $P \in \mathbb{R}^{n \times m}$. For our classification problem $n = m = K$. The resulting optimization problem is a Linear Program for which strong duality holds [1] (this is actually a result we study in the Convex Optimization course taught by Alexandre d'Aspremont). Strong duality means that the solution for the primal problem p^* and the solution for the dual problem d^* are equal $p^* = d^*$. The solution for Kantorovich's problem is the (exact) Wasserstein distance W.

$$W(h_\theta(x), y) = \min_{\substack{P \geq 0 \\ P\mathbf{1} = h(x) \\ P^T\mathbf{1} = y}} \langle P, C \rangle = \max_{C_{ij} \geq f_i + g_j} \langle f, h_\theta(x) \rangle + \langle g, y \rangle \tag{2}$$

Here f and g are the Lagrange multipliers. Solving the dual problem allows us to directly compute the gradient of the Wasserstein distance with respect to the output of the network

$$\frac{\partial W(h_\theta(x), y)}{\partial h_\theta(x)} = f^* \tag{3}$$

However, solving Kantorovich's optimal transport is costly. The authors instead solve an approximation, the Entropic Regularization formulation [2]. The solution to this problem is the approximate Wasserstein distance

$$W^\lambda(h_\theta(x), y) = \min_{\substack{P \geq 0 \\ P\mathbf{1} = h_\theta(x) \\ P^T\mathbf{1} = y}} \langle P, C \rangle - \frac{1}{\lambda} H(P) \tag{4}$$

where H is the relative entropy. This approximated problem can be solved efficiently using Sinkhorn algorithm. In addition, we obtain a vector $u = e^{\lambda f}$. We can therefore approximate the exact gradient of Kantorovich by $\frac{\log u}{\lambda}$. As a final remark, in the Entropic formulation the multipliers are actually unique up to an additive constant, i.e. if f and g are optimal then $f + c\mathbf{1}$ and $g - c\mathbf{1}$ are also optimal. The authors suggest using

$$\frac{\partial W(h_\theta(x), y)}{\partial h_\theta(x)} = \frac{\log u}{\lambda} - \frac{\log u^T \mathbf{1}}{\lambda K} \mathbf{1} \tag{5}$$

where K is the number of classes. Even though Sinkhorn reduces the complexity for computing the Wasserstein loss, it is still more costly than simply computing, for example, a Cross Entropy loss.

1.4 Personal contributions

We train several classification models with the Wasserstein loss changing important hyperparameters such as: the number of Sinkhorn iterations, the capacity of the network, the cost matrix and the balance between classes on the dataset. In their publication, the authors do not report the impact their approach has on runtime. In our numerical experiments we specifically compare the time required for computing the Wasserstein loss with different parameters for Sinkhorn and compare it against Cross Entropy.

In addition, we also train using the Sinkhorn divergence [7]

$$SK(h_\theta(x), y) = W^\lambda(h_\theta(x), y) - \frac{1}{2}W^\lambda(h_\theta(x), h_\theta(x)) - \frac{1}{2}W^\lambda(y, y) \tag{6}$$

As for the Wasserstein distance, the gradient of this divergence with respect to the network outputs $h_\theta(x)$ can be derived from the Lagrange multipliers we get when we apply Sinkhorn algorithm to compute $W^\lambda(h_\theta(x), y)$ and $W^\lambda(h_\theta(x), h_\theta(x))$

2 Experiments

We implement the Wasserstein loss on the popular deep learning framework PyTorch [11]. This allows us to easily use the loss and its gradient to train different models. An important implementation trick is that the Sinkhorn iterations can be run in parallel for a group of B vector pairs $(h_\theta(x_i), y_i)$ by stacking the corresponding vectors u_i in a batch of size B. This allows for efficient computation of the Wasserstein distance for a batch of training examples. Moreover, since Sinkhorn consists in matrix-vector multiplications, the whole algorithm can be paralleled in GPU. For our experiments we use the GPUs provided by Google Colab.

2.1 Dataset

We use the *Car Evaluation Data Set* [3]. It consists of 1728 examples of cars labeled as unacceptable (1210), acceptable (384), good (69) and very good (65). Each example has six attributes. Since there are few good and very good examples, we merged those categories into one with 134 elements. We also create a balanced version of the dataset by keeping only 134 elements of the unacceptable and acceptable categories. We separate 20% of the dataset for validation.

For the set of labels we use 0 (unacceptable), 1 (acceptable) and 2 (good).

2.2 Baseline

In order to have a first baseline, we train a feed-forward neural network on the balanced Car Dataset. The network's architecture is detailed in Table 1. We use this model because after some preliminary experiments we found that the time required for the training to converge and achieve good classification results is relatively small. This is important because we run several numerical experiments to test the performance of the Wasserstein loss on different scenarios and the experiments must run in a reasonable amount of time.

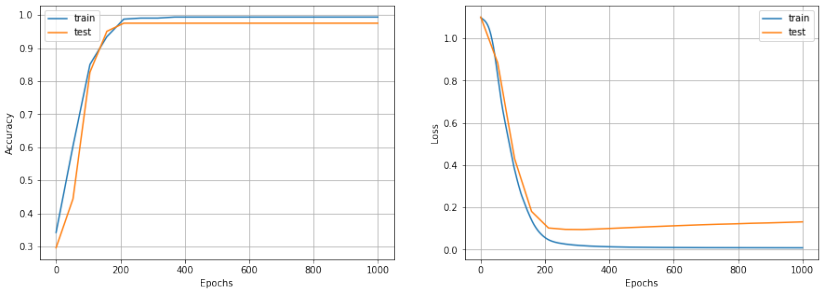
Layers
Linear(in=6, out=16)
Linear(in=16, out=32)
Linear(in=32, out=32)
Linear(in=32, out=16)
Linear(in=16, out=3)
Softmax(in=3, out=3)

Table 1: Baseline network architecture. Each linear layer, expect the last one, is followed by a ReLU activation.

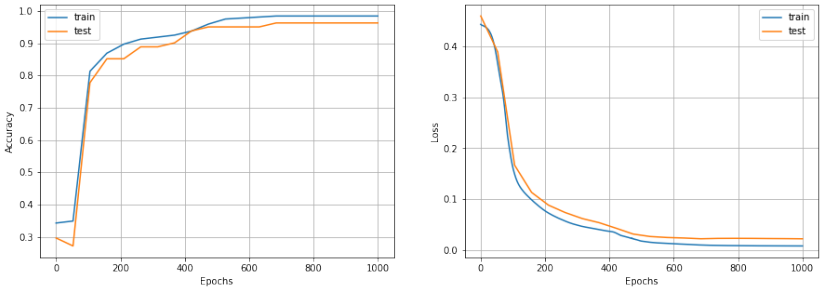
As suggested by the authors, to compute the Wasserstein loss we use Sinkhorn with a cost matrix

$$C = \begin{pmatrix} 0 & 0.5 & 1 \\ 0.5 & 0 & 0.5 \\ 1 & 0.5 & 0 \end{pmatrix} \tag{7}$$

regularization $\lambda = 50$ and $N = 10$ iterations. This cost matrix penalizes large deviations on the predictions, for example, predicting unacceptable as good is worse than predicting unacceptable as acceptable. For training we use Adam optimizer [9] with learning rate $\alpha = 0.001$, weights $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for 1000 epochs. Since the dataset is very small we use only one batch with all the examples. We also train the same model with the same set up but using Cross Entropy loss. In this case, the final soft-max layer is removed since the PyTorch implementation expects the logits as input. The training evolution and the confusion matrices for each loss are shown in Figure 2. Both losses achieves similar results in the classification task. Wasserstein is slightly better for classifying good cars whereas Cross Entropy is better for unacceptable cars. This first experiment shows that Wasserstein loss can be effectively used for training neural networks for classification and get competitive results. In the following sections we explore the effect of varying some important hyperparemeters on the performance of the network.

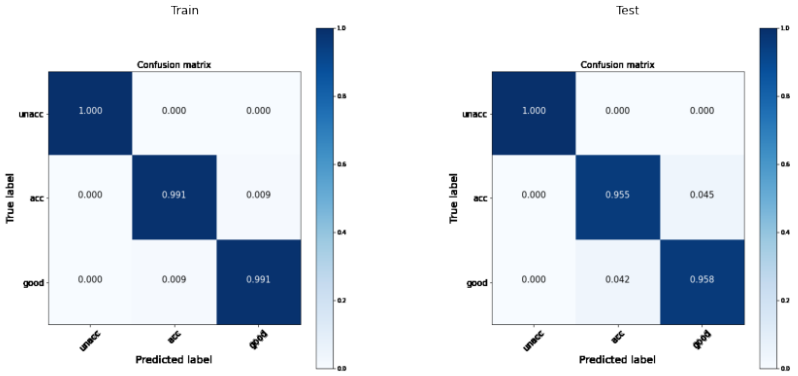


(a) Cross Entropy loss

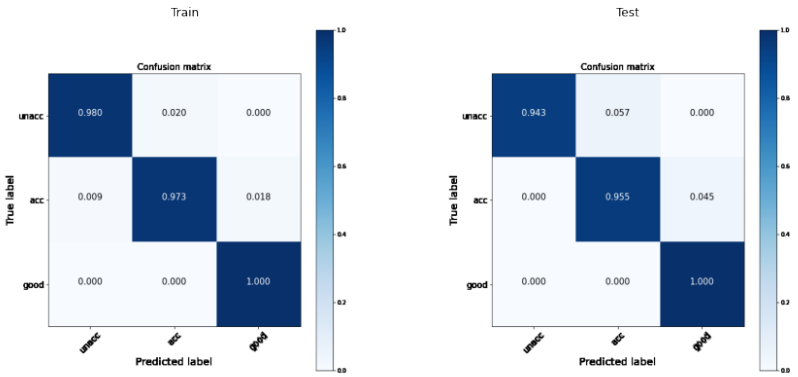


(b) Wasserstein loss

Figure 1: Loss and accuracy evolution during training on the balanced Car Dataset using Cross Entropy and Wasserstein loss



(a) Cross Entropy loss



(b) Wasserstein loss

Figure 2: Confusion matrix for the balanced Car Dataset using Cross Entropy and Wasserstein loss.

2.3 Effect of the number of Sinkhorn iterations

Since the Wasserstein distance is computed in each epoch using Sinkhorn algorithm, a major concern regarding the computational cost of the training is on the time required to get to a good solution. This time can be controlled by the number of Sinkhorn iterations N. A good value should balance the quality of the results and the runtime of the algorithm. We train the baseline network on the balanced Car Dataset for different values of N. In all cases similar we get similar values of accuracy, the inference performance does not decrease with fewer iterations. However, too many Sinkhorn iterations are harmful in terms of training time. Going from $N = 10$ to $N = 1000$ means about an $\times 3$ increase in runtime per epoch. The suggested value of $N = 10$ iterations turns out to be a more than reasonable choice in this experiment. A curious finding is that even the case $N = 1$ obtains good results in this dataset. Results are summarized in Table 2

As a reference, computing the Cross Entropy loss takes $0.000123s$ in average. This means that the Wasserstein loss computation is roughly 350 times slower for $N = 10$.

Sinkhorn Iterations	**Accuracy**	**Average loss time computation (sec)**
1	0.963	0.0446
10	0.988	0.0448
100	0.963	0.0545
500	0.988	0.089
1000	0.963	0.153
2000	0.988	0.223

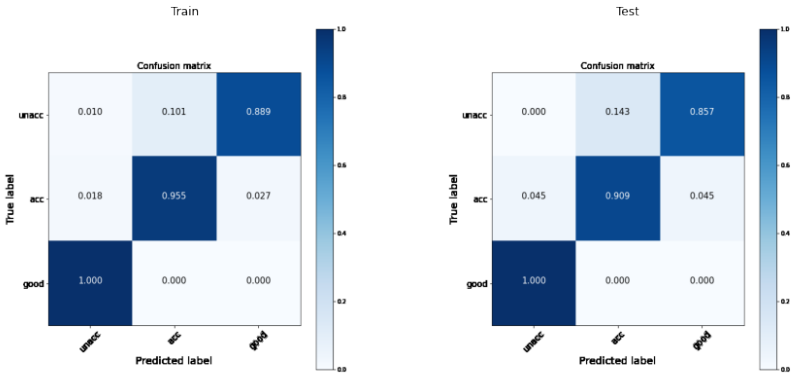
Table 2: Results obtained varying the number of Sinkhorn iterations for the balanced Car Dataset on validation.

2.4 Effect of the cost matrix

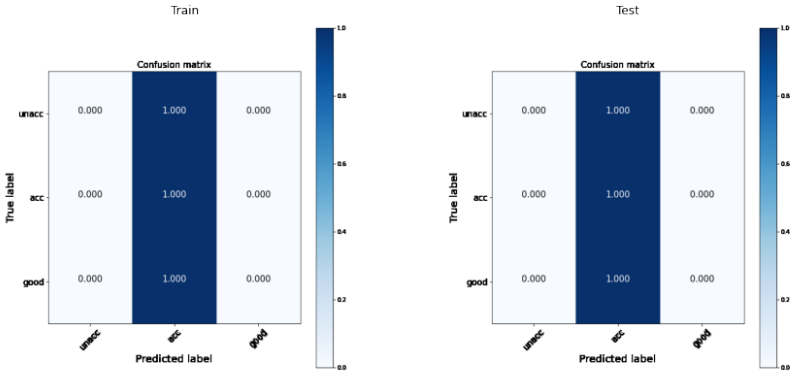
The cost matrix C offers a great deal of freedom when it comes to penalizing different errors. As an example, consider the cost matrices

$$C_1 = \begin{pmatrix} 1 & 0.5 & 0 \\ 0.5 & 0 & 0.5 \\ 0 & 0.5 & 1 \end{pmatrix} \text{ and } C_2 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \tag{8}$$

Of course these costs do not make sense for classification purposes, but they are useful for illustration. When training with the Wasserstein loss with C_1 and C_2 we get the results of Figure 3. The impact of different costs in the predictions is clear. In the case of an asymmetric matrix C_2, the resulting structure of the confusion matrix is due to the transpose C_2^T. Since C_1 is symmetric, there is no need to transpose it to find the relation with the confusion matrix. In summary, the cost matrix C allows as to control the penalization of different miss-classification errors.



(a) Results for cost C_1

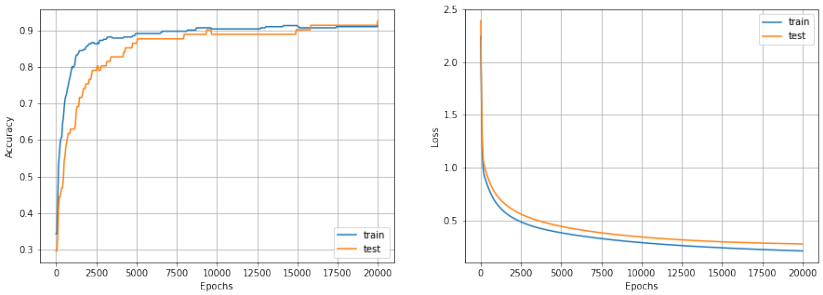


(b) Results for cost C_2

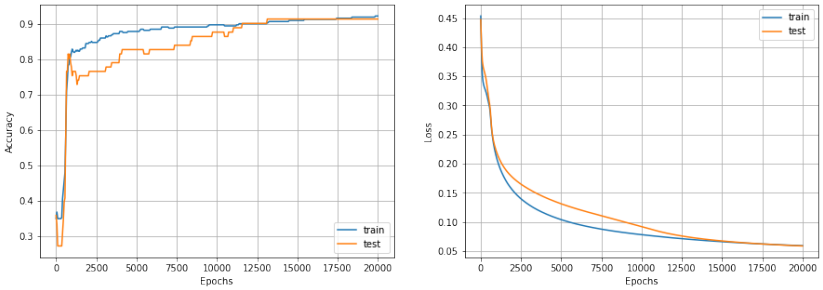
Figure 3: Confusion matrix for the balanced Car Dataset using Wasserstein loss for the cost matrices C_1 and C_2 of Equation 8.

2.5 Effect of smaller network

We study the smallest neural network for this task, a single linear layer with 6 inputs and 3 outputs. The idea is to check if the Wasserstein loss performs poorly for a network with less capacity. For training, we increase the number of epochs to 20000 to allow the learning process to converge. See Figure 4 for the training evolution and the confusion matrices. The final performance is worse than with our baseline network, something expected since this smaller model has less capacity. Both losses achieve similar final results, the corresponding confusion matrices are given in Figure 5.

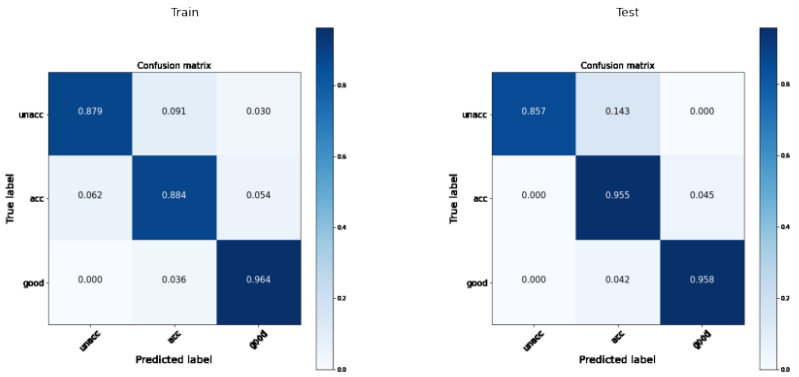


(a) Cross Entropy loss

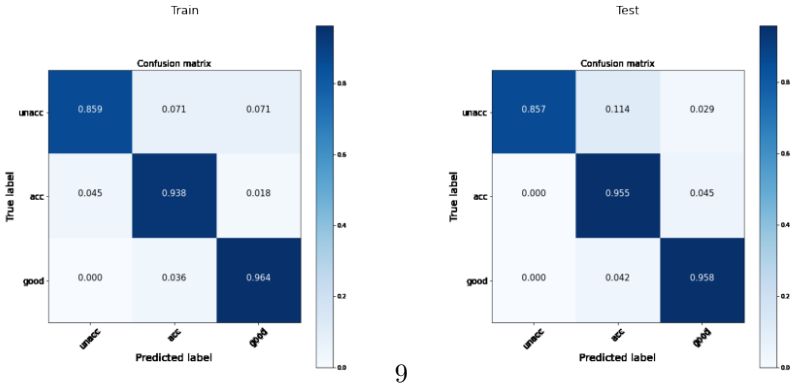


(b) Wasserstein loss

Figure 4: Loss and accuracy evolution during training on the balanced Car Dataset using Cross Entropy and Wasserstein loss for a single-layer neural network.



(a) Cross Entropy loss



(b) Wasserstein loss

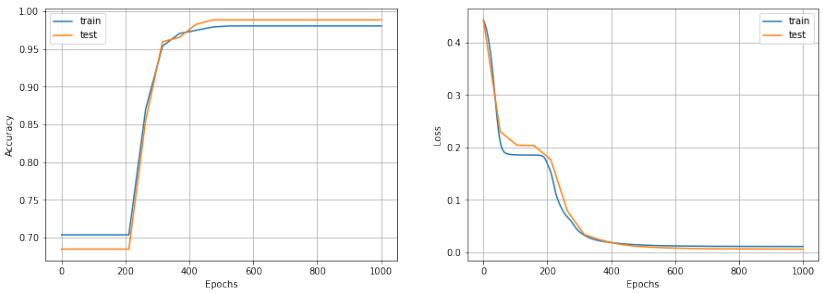
Figure 5: Confusion matrices on the balanced Car Dataset using Cross Entropy and Wasserstein loss for a single-layer neural network.

2.6 Effect of an unbalanced dataset

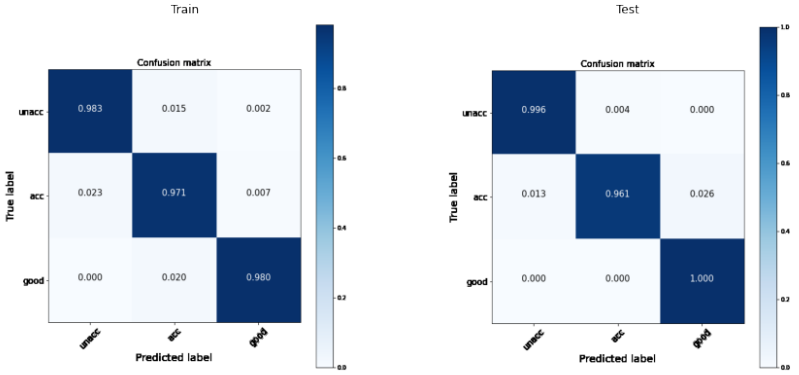
We study the effect of training the baseline network with the unbalanced version of the Car Dataset with 1728 examples of cars labeled as unacceptable (1210), acceptable (384), good (134). The results of this experiment are summarized in Figure 8. We observe that the Wasserstein loss is heavily influenced by the dominant class of unacceptable cars and its predictions collapse into that class. This is a surprising result, especially since Cross Entropy does not suffer from this problem. An important lesson is that we must give very especial attention to the class distribution of our dataset before training with a Wasserstein less.

2.7 Effect of using Sinkhorn divergence

The most interesting result we find is that training on the unbalanced dataset using Sinkhorn divergence instead of Wasserstein distance is less prone to result in predictions collapsing into the dominant class. This collapsing behaviour was observed only a couple of times for Sinkhorn divergence but most of the time it did not collapse. Of course using Sinkhorn divergence means a $\times 3$ increase in runtime, but the number of iterations for the new terms involved can be decreased. The results obtained for the unbalanced dataset are shown on Figure 6

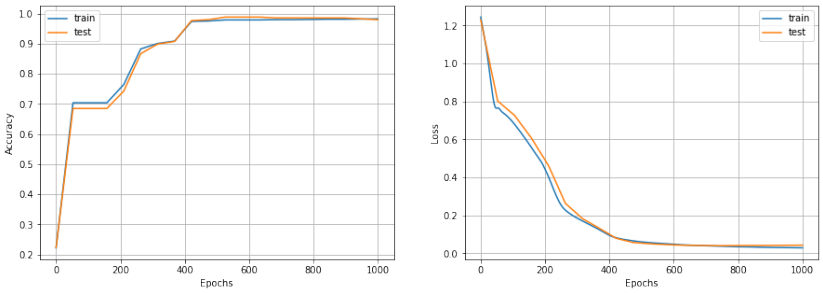


(a) Accuracy and Sinkhorn Divergence evolution

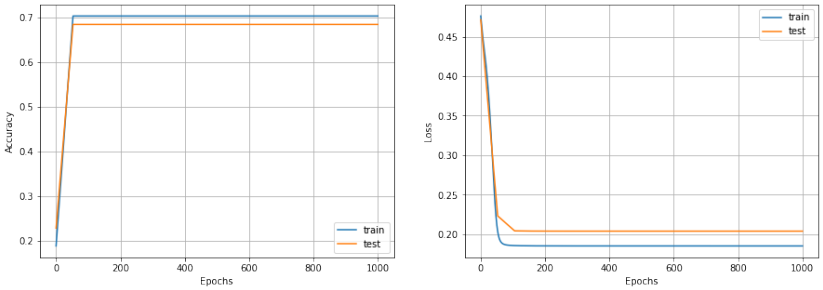


(b) Confusion matrices

Figure 6: Loss and accuracy evolution during training on the unbalanced Car Dataset and confusion matrices using Sinkhorn Divergence.

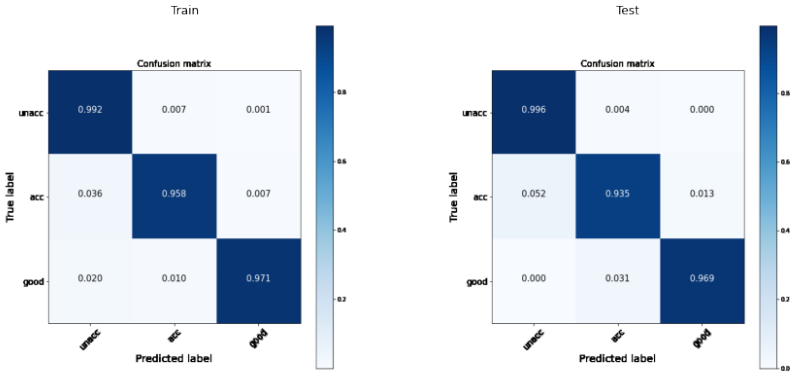


(a) Cross Entropy loss

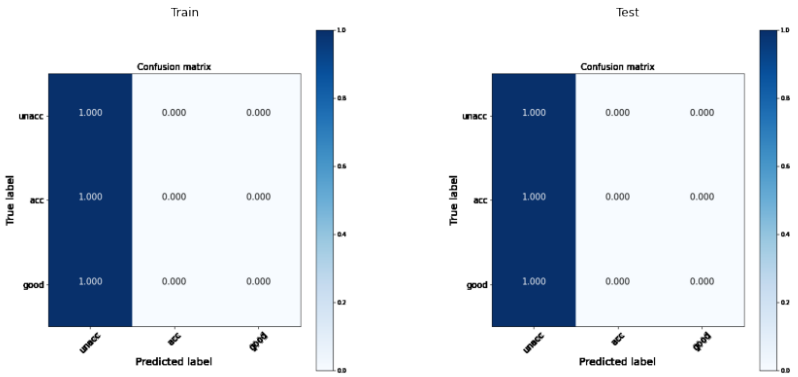


(b) Wasserstein loss

Figure 7: Loss and accuracy evolution during training on the unbalanced Car Dataset and confusion matrices using Cross Entropy and Wasserstein loss.



(a) Cross Entropy loss



(b) Wasserstein loss

Figure 8: Confusion matrix for the unbalanced Car Dataset using Cross Entropy and Wasserstein loss.

3 Conclusions

In this report we implement a Wasserstein loss as proposed in *Learning with a Wasserstein Loss* by Fogner et al. [5], and use it to train neural networks for ordinal classification. The Optimal Transport formalism directly introduces a cost matrix that enables us to take into account the ordering between the labels and lift the ground truth metric to a distance between probability vectors. Training with this loss indeed achieves different results than using Cross Entropy, one of the currently most popular losses for classification tasks. We run several experiments changing relevant hyperparameters and test how they affect the predictions. We always compare each experiment for the Wasserstein loss using the same setting but training the network with Cross Entropy loss. In addition, we also use a variation of the Wasserstein distance for training, the Sinkhorn Divergence.

The Wasserstein loss offers a direct way of customizing the penalties for misclassification errors via the cost matrix C. This is very powerful, as we can freely set the penalization for each type of error. We also include some experiments regarding the size of the neural network and the dataset. We achieve good classification results when training a very small neural network consisting of only one linear layer, but it require substantially more training epochs. This is also the case for Cross Entropy.

We train our neural network on a heavily unbalanced dataset and we obtain poor results with the Wasserstein loss. The network collapses and ends up always predicting the dominant class of the dataset. Using Sinkhorn divergence alleviates the problem, but the collapsing behaviour still happens in some cases. This problem does not happen when training with Cross Entropy loss. This suggests that the Wasserstein distance is more prone to get stuck at a local minimum during the learning process on unbalanced datasets.

Another limitation of using the Wasserstein loss is that the training process is computationally more expensive because for each output of the network we need to run Sinkhorn algorithm. Although Sinkhorn can be run in parallel for a batch of vector pairs, the process is still more expensive than simply using a Cross Entropy loss. Our experiments show that the computation of the Wasserstein loss is 350 times slower than the computation of the Cross Entropy loss. This could be prohibitive for training large models on massive datasets.

4 Connection with the course

Learning with a Wasserstein Loss is directly based on several topics covered in the course. Firstly, authors propose to use the Wasserstein distance from the Kantorovich's formulation as a loss function between the output of the network and the target. Moreover, the dual problem of the Kantorovich's formulation allows us to easily obtain a closed form for the gradient of the Wasserstein loss with respect to the outputs of the network. This is a fundamental step in the training process, since we can chain this gradient and use backpropagation to train any model we want using a framework with automatic differentiation such as PyTorch.

Since directly solving this formulation is costly, we use the Entropic Regularization approximation. For computing the loss and its gradient we use Sinkhorn,

an algorithm presented in the lectures and studied in several Numerical Tours. This algorithm is especially attractive because it is computationally less expensive than directly solving the Linear Program in Kantorovich's formulation. In addition, the Sinkhorn computations at each iteration can be done in parallel for several pairs $(h_\theta(x_i), y_i)$ by stacking them in a batch, which makes it a very suitable algorithm for training a neural network using batch gradient descent.

To sum up, *Learning with a Wasserstein Loss* uses of some of the most important topics covered during the course, namely Kantorovich's formulation of Optimal Transport, the Entropic Regularization for faster but approximate optimization and the Sinkhorn algorithm.

References

[1] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[2] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[3] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[4] Kilian Fatras, Younes Zine, Rémi Flamary, Rémi Gribonval, and Nicolas Courty. Learning with minibatch wasserstein : asymptotic and gradient properties, 2021.

[5] Charlie Frogner, Chiyuan Zhang, Hossein Mobahi, Mauricio Araya-Polo, and Tomaso A. Poggio. Learning with a wasserstein loss. *CoRR*, abs/1506.05439, 2015.

[6] Lisa Gaudette and Nathalie Japkowicz. Evaluation methods for ordinal classification. In Yong Gao and Nathalie Japkowicz, editors, *Advances in Artificial Intelligence*, pages 207–210, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[7] Aude Genevay, Gabriel Peyré, and Marco Cuturi. Learning generative models with sinkhorn divergences, 2017.

[8] Pedro Antonio Gutiérrez, María Pérez-Ortiz, Javier Sánchez-Monedero, Francisco Fernández-Navarro, and César Hervás-Martínez. Ordinal regression methods: Survey and experimental study. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):127–146, 2016.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Sotiris B. Kotsiantis and Panagiotis E. Pintelas. A cost sensitive technique for ordinal classification problems. In George A. Vouros and Themistoklis Panayiotopoulos, editors, *Methods and Applications of Artificial Intelligence*, pages 220–229, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[12] Rob Potharst and Jan Bioch. Decision trees for ordinal classification. *Intell. Data Anal.*, 4:97–111, 03 2000.