The Wasserstein Loss Function

Prafulla Dhariwal
prafulla@mit.edu
MIT

Jeevana Inala
jinala@mit.edu
MIT

1. Introduction

In many learning scenarios, the target variable space has a natural metric associated with it that captures the notion of semantic similarity between different target values. We can utilise this metric to define better loss functions that can incorporate the information from this metric into the learning algorithm. One such loss function is the Wasserstein Loss function, which provides a notion of the distance between two measures on a target label space with a particular metric. The Wasserstein distance between two measures is defined as the amount of "mass" that has to move times the distance by which it needs to move to make the two measures the same. The inspiration for our project was the recent NIPS paper (Frogner et al. 2015), which proposes to use the Wasserstein Loss function in a supervised learning setting, specifically, for a multi-class multi-label learning problem. In this project, we would like to explore the properties of this loss function by comparing its accuracy, convergence rates etc. against other loss functions, and by evaluating how changes in parameters and the distance metric impact its performance. We also try to reproduce some of the results described in (Frogner et al. 2015).

Before we go into the details, we provide a brief summary of our project work:

1. We implemented the Wasserstein loss function as a loss layer in Caffe (Jia et al. 2014).[1] (Previously, the only open source implementation of the loss function was in Mocha.jl, which we used as a reference)
2. We tested our implementation and benchmarked it for some simple data sets.
3. We used our implementation to replicate the results in (Frogner et al. 2015). Specifically, we re-implement their experiments in Caffe with a toy dataset as well as with the famous MNIST dataset (Lecun and Cortes)
4. We explored how changes in the distance metric and use of relaxed versions of the loss function affect the convergence and the performance.

As an extra, we have implemented our loss function in a way that will allow it to be later merged with the Caffe library, thus allowing others to explore the use of the Wasserstein loss function with any model in the Caffe model zoo.

Note that because we implement this in Caffe, we confined ourselves to work within the limits of the Caffe framework, which is essentially a framework for Convolution Neural Networks. We had also planned on evaluating the effect of using the Wasserstein layer in a multi-label setting like the Places2 data set, however, due to the lack of time and computation resources, we weren't able to do so. We, however, do give a sketch of what we would have done in section 6

[1] Our implementation is available here: `https://github.com/prafullasd/caffe-cpu`

2. The Wasserstein Loss function

In this section, we describe in brief the theory of the Wasserstein loss function, as well as its convex relaxation that has an efficient algorithm for computing the loss and the gradient. This is a brief summary of the work in (Frogner et al. 2015). For more details of related work and proofs of results, refer to (Frogner et al. 2015). We also talk about the KL divergence and the Multinomial Logistic Loss function, which are similar loss functions that will be used for comparison in this project.

2.1 Definition and Convex Relaxation

The learning problem we study is to learn a map from an input set $X \subset \mathbb{R}^d$ to a space $Y = {\mathbb{R}_+}^K$ of measures over a set $\mathcal{K}$ of K target classes. The target classes have a ground metric $d_{\mathcal{K}}$ associated with them, which indicates the semantic similarity between two target classes. Let $M \in {\mathbb{R}_+}^{\mathcal{K} \times \mathcal{K}}$ be the distance matrix on the K target labels associated with this metric i.e. $M_{k,k'} = d_{\mathcal{K}}(k, k')$ for $k, k' \in \mathcal{K}$. Let $h_\theta(\cdot|x)$ represent a hypothesis in our space of hypothesis H. Given a vector of prediction probabilities $h_\theta(\cdot|x)$ and a ground truth target measure $y(x) \in {\mathbb{R}_+}^K$, the Wasserstein's loss between them is then defined by (Definition 3.1 in (Frogner et al. 2015))

$$W(h_\theta(\cdot|x), y(x)) = \inf_{T \in \Pi(h(x), y)} \langle T, M \rangle$$

where $\Pi(h(x), y(x))$ is the set of valid transport plans given by

$$\Pi(h(x), y) = \{T \in {\mathbb{R}_+}^{\mathcal{K} \times \mathcal{K}} | T\mathbf{1} = h(x), T^T\mathbf{1} = y(x)\}$$

One can understand this my noting that the matrix T represents possible ways of transporting the mass in the measure $h(x)$ to the measure $y(x)$, and we take its dot product with the distance matrix M to penalize transportation over longer distances.

For solving the learning problem, we need to minimise the above loss function W, however, as noted in (Frogner et al. 2015), calculating the exact subgradient is computationally expensive. Hence, the authors suggest smoothened version of the loss function that is strictly convex. It is defined as

$$^{\lambda}W(h_\theta(\cdot|x), y(x)) = \inf_{T \in \Pi(h(x),y)} \langle T, M \rangle + \lambda H(T) \quad (1)$$

where $H(T) = -\sum_{k,k'} T_{k,k'} \log T_{k,k'}$. Here, the regularizer $H(T)$ uses the entropy of the transportation matrix as a measure of its complexity, thus regularizing the Hypothesis set by favouring simpler hypotheses. Efficient iterative algorithms exist for finding the transportation matrix T^* that minimises (1), as well as for finding the gradient of (1) (Frogner et al. 2015). The algorithm we used for our implementation is outlined in the Section 3

2.2 Other similar loss functions

Throughout the paper, we will be using the KL divergence as the standard multi-label multi-class loss function to compare against. It is defined as the divergence

$$D(t||p) = -\sum_i t_i \log(p_i)$$

between the target distribution t and the predicted distribution p. For single-label problems, this is equivalent to the Multinomial Logistic Loss which is given by $l = -t \log(p_t)$, where t is the target class label, and p_t is the predicted probability for the target class t. Note that all of these loss functions would need to take in a probability distribution p as input, so usually one would apply the Softmax function to map predictions to a probability distribution before applying these loss functions.

3. Implementation in Caffe

We implemented the Wasserstein loss as a separate loss layer in Caffe. This loss layer takes the distance matrix, M as an input in addition to the other parameters such as λ and the number of Sinkhorn iterations, $sinkhorn\_iter$. As in any loss layer, there are two important functions in this layer: $Forward(...)$ and $Backward(...)$ that correspond to the loss computation during forward propagation and the gradient computation during backward propagation. Since, it is efficient to compute the gradients along with the loss, we precompute the gradients during the $Forward$ phase and use them later in the $Backward$ phase. The inputs to each of these functions are the predicted labels, $pred$ and the expected ground truth, exp. Let n be the number of data points and k be the number of labels. Here is the algorithm that we used to compute the loss and the gradient (Frogner et al. 2015) (Here . is used for element-wise operations, thus .* is element-wise multiplication etc.).

Data: M, pred, exp
Result: loss,grad
K = $e^{-\lambda M}$;
KM = K .* M;
u = ones(n, k) ./ k ; `// Uniform distribution`
for *1:sinkhorn_iter* **do**
  u = pred ./ ((exp ./ (u * K)) * K);
end
v = exp ./ (u * K);
loss = sum(u .* (v * KM)) / n;
alpha = log(u) / (λ * n); `// Gradient calculation`

Algorithm 1: Wasserstein loss and gradient calculation

3.1 Testing Caffe implementation

We tested our implementation using the Gradient tester in Caffe. It performs multiple forward passes for the layer, computes the objective values, and then uses those to compute the gradient numerically using finite-differences. We note that the algorithm in 1 computes the gradient only up to a constant shifting factor, and thus, to check our implementation, we have to subtract our gradient from the numerical gradient and check if it is $c\mathbf{1_n}$ (up to some threshold) for some constant $c \in \mathbb{R}$. The gradient test was successful.

3.2 End to end test: Toy dataset 1

In addition to the above check, we also performed an end-to-end test on a small data set (this is the same as the end-to-end test in the Mocha.jl implementation). It helps to check for any run-time issues in real time, the convergence rates as well as the speed of computation of our layer. For our data set, we generate 100 uniformly random points in $[0, 1]^4$. We then set the target label of each data point $x = (x_1, x_2, x_3, x_4)$ as

$$y(x) = \arg\max_i (x_i)$$

. The model used in Caffe consists of three inner product layers with a ReLU as the activation function. The output of the final layer is passed to a softmax layer to obtain probabilities for the 4 classes. We then use the Wasserstein loss layer to check if it can learn accurately for this model, and also compare its performance against the Multinomial Logistic Loss Layer (KL divergence). The results obtained are shown in Figure 1.

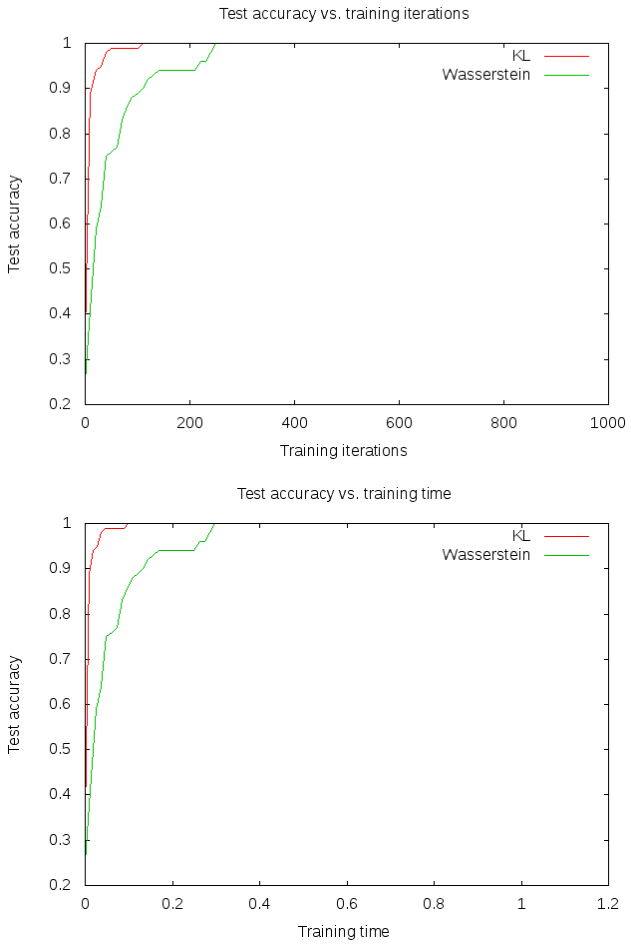

Figure 1: Performance of the multinomial logistic loss layer i.e. KL divergence vs Wasserstein loss layer in Caffe. The data set is toy dataset 1, and the Wasserstein loss layer has Sinkhorn iterations = 50.

The Wasserstein loss is able to achieve the same accuracy (1) as the KL divergence after allowing both of them to converge. The accuracy rate with respect to the number of iterations indicates convergence rate, and the Wasserstein layer is only about a factor of 2 slower. The accuracy rate w.r.t time indicates the speed of computation of our implementation. Here, we note that even though Wasserstein loss layer involves computing 50 iterations of the Sinkhorn algorithm, it is still only about a factor of 3 slower in real time.

4. Exploring properties of the Wasserstein loss function

We now replicate some of the results shown in (Frogner et al. 2015). While doing these experiments, we learned a lot more about the properties of the Wasserstein loss function, as well as about its implementation. Some of the new results we obtained such as the effect of scaling the ground metric and replacing the relaxed version by using $\lambda = 0$ version are also mentioned.

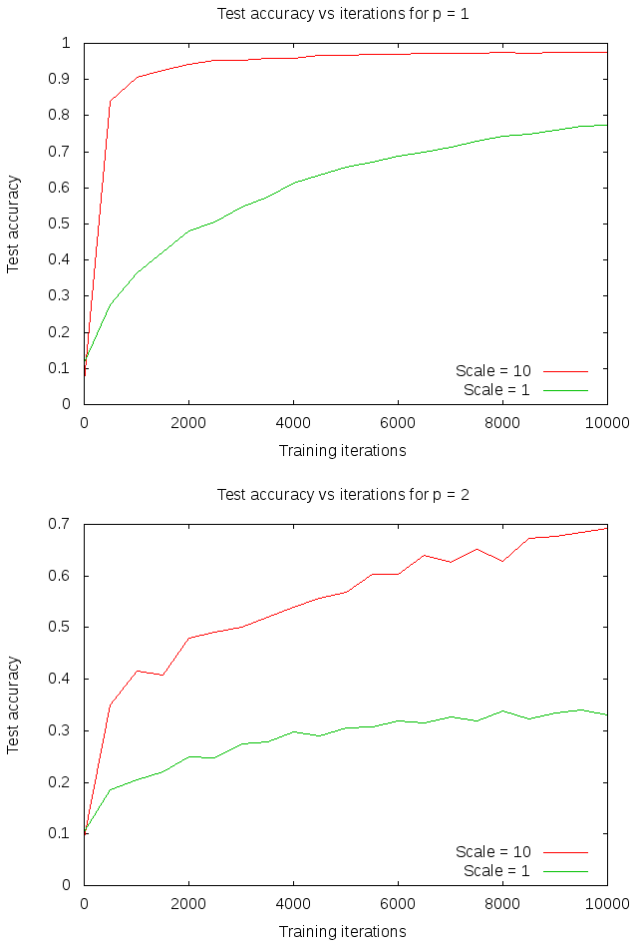

Figure 2: Accuracy vs iterations on the MNIST dataset for different scaling of the distance matrix, shown for p-norms 1 and 2. The convergence is much faster after we scale the distance matrix by a factor of 10, and this held true over multiple runs and also on changing the value of p. The plots were obtained using the LeNet model with a Wasserstein Loss layer having $\lambda = 0.1$ and sinkhorn iterations = 50

4.1 MNIST

In this section, we use the Wasserstein loss function on the MNIST handwritten digits data set. This is a multi-class learning problem with $K = 10$ target classes, and we can define an artificial metric on the target space by setting $d_p(i, j) = |i - j|^p$. Just like in (Frogner et al. 2015), we show that the use of the Wasserstein loss function with this metric imposes a smoothness on the predictions that makes the predictions be numerically close to the actual value.

We use the LeNet convolution neural network (LeCun et al. 1998) in the Caffe Model zoo and replace the top softmax loss layer with the Wasserstein loss layer. Our training set consists of 60,000 examples, and the test set consists of

10,000 examples. To evaluate the effect of the change in loss layers, we run both models (LeNet and LeNet-Wasserstein) in Caffe for 10,000 training iterations, with the same hyper-parameters in the common part of the model. For the Wasserstein loss layer itself, we used $\lambda = 0.1$ and $p = 1$. We noted that increasing the Sinkhorn iterations from 10 to 50 improved the accuracy of our Wasserstein layer considerably. With these settings, we obtained that the accuracies of both methods were almost identical (98%) when we allowed them to converge.

We then experiment with the LeNet-Wasserstein model only by changing the metric i.e. the value of p. Notice that for $p = 0$, the metric reduces to the $0 - 1$ loss as each incorrect label is penalised equally, while, as $p \to \infty$, the loss associated with the neighbouring labels converges to a uniform distribution. For numerical stability, we normalise our metric so that all distance values lie in $[0, 10)$. We initially chose the upper bound as 1 as in (Frogner et al. 2015), however, we noticed that scaling the distance values by a factor of 10 improves convergence rates, and given the limited computation resources we had this was really helpful. A graph of the convergence rates for the Wasserstein loss layer for the two scaling factors is shown in Figure 2. We could not scale beyond 10 due to numerical limits on the loss value.

Now, for each p, we train the LeNet-Wasserstein model for 10,000 iterations, and evaluate the mean probability of the predictions associated with each target label. The plots for the target labels 0, 5, and 9 are shown in Figure 3. As we can see, for small p, the probability is concentrated on the correct target label, while as p increases, the probabilities tend to a uniform distribution as expected. Note also that the probabilities are maximum for the target value, and decrease as we go to more dissimilar neighbours according to the distance metric used. Also note that even though the 9 and 0 would have different input image features, they seem to have the same nature for the plot, thus giving further evidence that the nature of the plot stems from the distance metric (which is symmetric between 0 and 9) and not from the specific nature of the input digits.

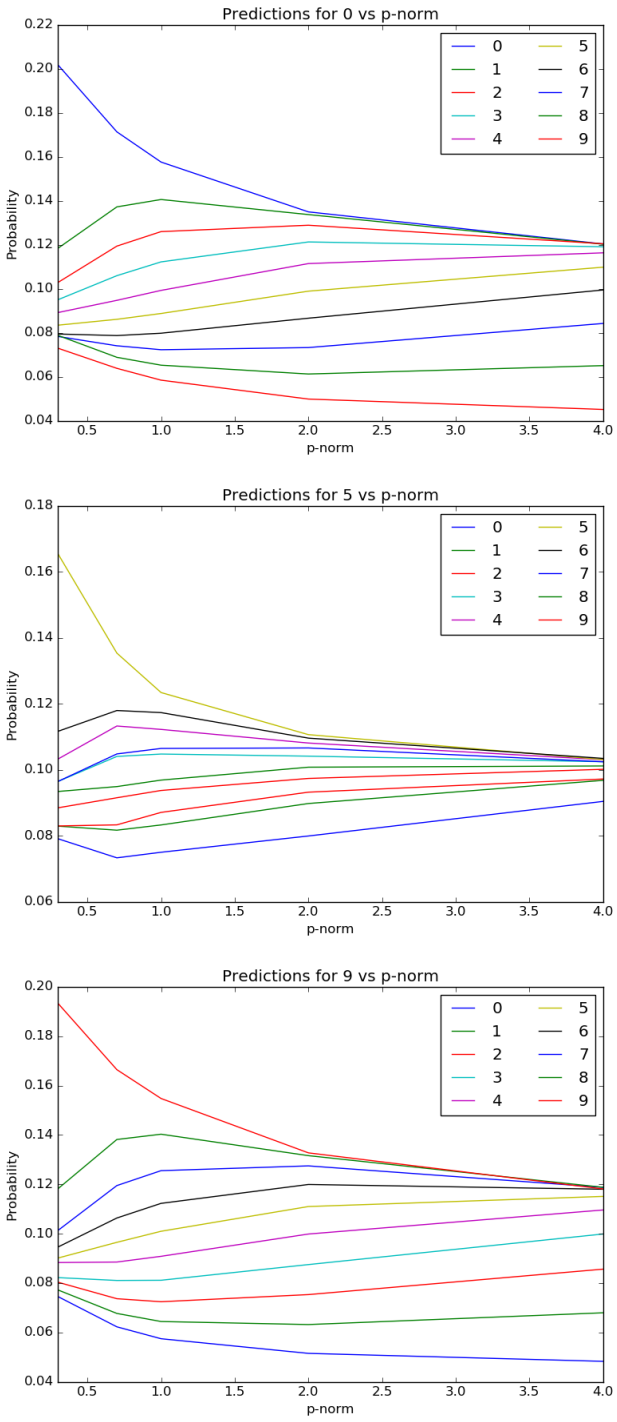

Figure 3: Mean prediction probabilities for target labels 0, 5, and 9 vs p-norm. The plots were obtained using the LeNet model with a Wasserstein Loss layer having $\lambda = 0.1$ and sinkhorn iterations = 50

In (Frogner et al. 2015), the authors obtain the same results. They, however, use a specialised simplification of the Wasserstein loss function which applies to the case of single-

label multi-class problems (Appendix C in (Frogner et al. 2015)). We briefly talk about their implementation here, as it provides more insight into the behaviour of the Wasserstein loss function for the single-label multi-class case. In this case, each target vector here is a vector e_k with 0 at all but the k-th place, where $k \in \{0, 1, ..., 9\} = \mathcal{K}$. Thus, the constraint $T^T\mathbf{1} = e_k$ forces all but the k-th column of T to be 0, while the constraint $T\mathbf{1} = h_{\hat{\theta}}(\cdot|x)$ forces the k-th column to be $h_{\hat{\theta}}(\cdot|x)$. Thus, there is only 1 valid transport plan T, and the unrelaxed loss function (ie for $\lambda = 0$) can be directly written as

$$W_p^p(h_{\hat{\theta}}(\cdot|x), e_k) = \sum_{k' \in \mathcal{K}} d_{\mathcal{K}}^p(k', k) h_{\hat{\theta}}(k'|x)$$

. Their implementation approximates this by the Softlabel Softmax layer in Mocha.jl, which is essentially the KL divergence $D(l||p)$ of the softlabels l and the predictions $p = h_{\hat{\theta}}(\cdot|x)$. Specifically, for the MNIST dataset, if a data point x had hard-label $y = k$, they set its softlabels to be a 10 dimensional vector given by $l_{k'} = s(k', k)$, where s is a similarity metric derived from the distance metric d. Thus, $s \propto -d$, which means that the KL divergence can be written as

$$D(l||p) \propto \sum_{k' \in K} d(k', k) \log(h_{\hat{\theta}}(k'|x))$$

which is the same as W_p^p except for the presence of the $\log$ factor. This formulation was favoured for numerical stability and easy of calculation of the KL gradient. Because the $\log$ function is monotonic, for predictions that are away from uniform they are almost equivalent. This is also supported by the fact that the results of our experiment using the Wasserstein layer were very similar to the results the authors in (Frogner et al. 2015) obtained using the approximate version (Our plots can be compared to Figure 4 and 9 in (Frogner et al. 2015)).

4.2 Noisy labels

In this section, we measure the performance of Wasserstein loss on a training data with noisy labels. For the classification problems where different categories are nearly indistinguishable, it is likely that the training data labels are noisy. Since Wasserstein loss encourages predictions that are close to the ground truth, it is expected that even in the presence of noisy training labels, the predictions on the test data set should not deviate much from the true labels. In order to verify this hypothesis, we simulate a toy data set whose labels are points on a 2D grid. Here, we use the Euclidean distance between any two labels as the natural metric. For each point of the grid, we generate data from a Gaussian distribution centered on that point and having a standard deviation of 0.2. Then, based on the noise level t, we randomly choose $t *$ (# of training samples) data points from the training set and flip their labels to one of their neighboring points on the grid (uniformly choosen). For instance, Figure 4 shows the training data set for a 3×3 grid with t = 0.1 and t = 0.5 respectively. The testing data set is also generated in the above manner except that their labels are not flipped.

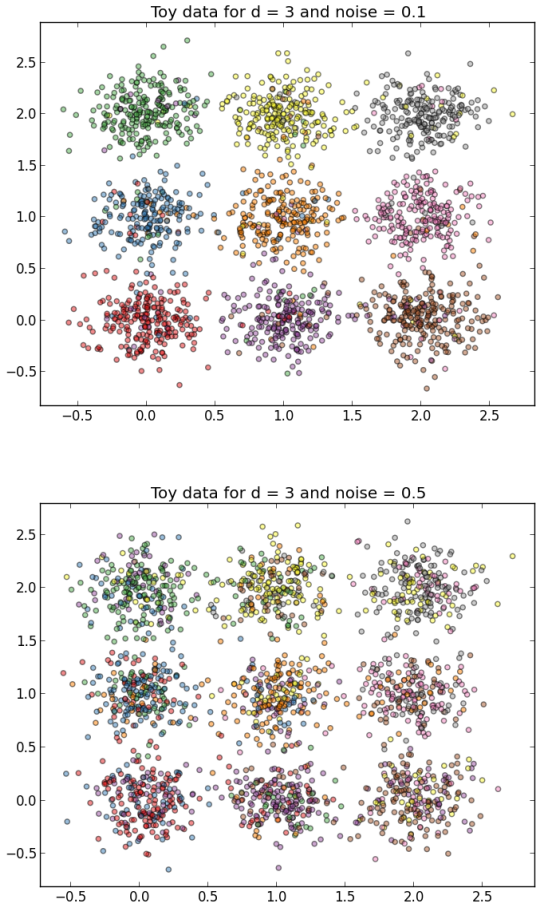

Figure 4: Training data set for a $3\times$ 3 grid with noise level = 0.1 and noise level = 0.5

Here, we experiment with different t values from 0.1 to 0.9 and different $d \times d$ grids where d ranges from 3 to 7. We run each of these configurations 10 times. Our training and test data sets have 2000 and 200 data points respectively.

We use a neural network with two hidden layers with each layer having 64 nodes. The input layer has two features corresponding to the x and y-coordinates of the point and the output layer has $d\times d$ nodes. We construct two networks with one of them using a soft-max loss layer (KL divergence) and the other using our Wasserstein loss layer. We compare the performance of these two loss layers by measuring the average Euclidean distance between the predicted values and the true labels over all the test data points.

For the Wasserstein loss layer, we found that $\lambda = 0.1$ and number of Sinkhorn iterations = 50 give the best results and hence, we used those values for all the experiments. To prevent the Wasserstein loss from diverging, we use a base learning rate of 0.001 for the SGD algorithm. Also, because

of our limiting computing resources, we use only 10,000 SGD iterations as opposed to 100,000 iterations mentioned in the original paper.

The results of our experiments can be found in Figure 5 and Figure 6. Our results do not quite match the results in (Frogner et al. 2015) (Figure 2). We found that Wasserstein loss beats KL divergence loss only for small d and higher noise levels. For the other values, both the losses perform almost equally. We made the following observations that may help explain this discrepancy:

1. When the noise level is too small, both Wasserstein and KL divergence losses are achieving a very high accuracy (about $97 \sim 99\%$). Hence, only a tiny fraction of the test data points are misclassified. Even with KL divergence, these misclassified data points are likely to be classified into any one of the adjacent classes. Hence, there is not much difference between the loss functions.

2. When the noise level is too high, there are two competing factors that seems to cancel each other. First of all, because the training data is too noisy, Wasserstein loss will make predictions that are closer to the ground truth compared to KL divergence loss. However, we found that the Wasserstein loss version achieves smaller accuracy than KL divergence loss. We believe that this might be because of the numerical instability and approximation errors that are inherent in the Wasserstein loss computation. Had we used the specialized simplification (as mentioned in Appendix c of (Frogner et al. 2015)) that uses a soft-label soft-max loss, we would have overcome these issues and would probably have got better results.

3. Finally, for large values of d, we found that the Wasserstein loss implementation takes longer to converge. In this respect, we found that normalizing and scaling the distance matrix helped to some extent (similar to what we saw in Section 4.1). However, we were not able to find a scaling factor that works best for all grid sizes – a small scaling factor worsens the convergence rate for larger grids and a large scaling factor makes the loss diverge for smaller grids. On the other hand, increasing the number of SGD iterations would have helped but we did not have enough computational resources to perform that experiment.

5. Work Division

We both divided the work more or less equally. Both of us worked on implementing the Wasserstein loss layer in Caffe and testing the implementation. Prafulla worked on analysis the MNIST data set and Jeevana worked on analysis of noisy labels data set. We made all the decisions together and both us understand all of the material presented here. A

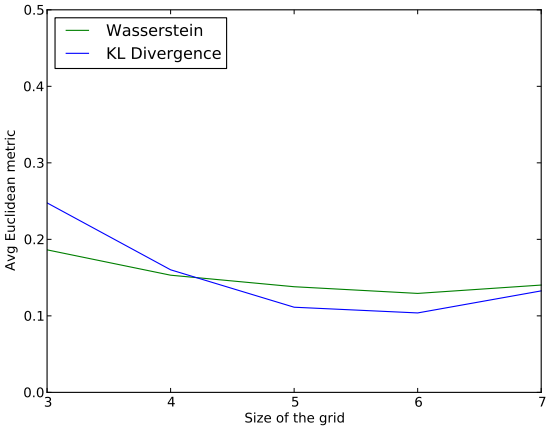

Figure 5: Effect of the size of the grid on the performance of Wasserstein loss layer and Multinominal logistic loss layer (KL-divergence).

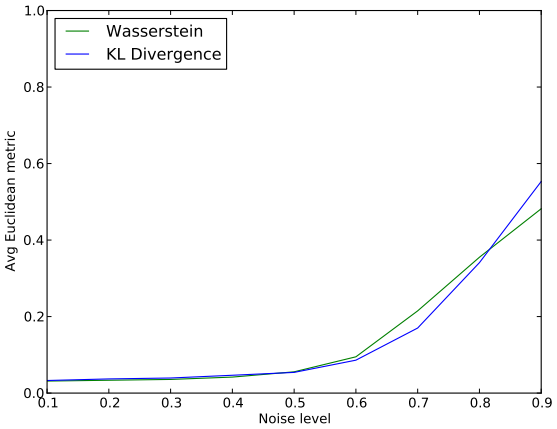

Figure 6: Effect of the noise level on the performance of Wasserstein loss layer and Multinominal logistic loss layer (KL-divergence).

majority of time during the first half of project was spent on implementing the loss layer in Caffe, as we were both new to Caffe. During the second half, we spent a lot of time trying to improve the performance of the Wasserstein layer, and it taught us a lot about the different parameters involved.

6. Future Work

An important experiment we would like to perform if we get time is to evaluate the use of the Wasserstein Loss layer when used for the multi-label multi-class problem of recognising tags associated with scenes. An important data set in this setting is the Places2 (Zhou et al. 2015) data set, which consists of 10 million images in 400+ scene categories. We wanted to work on a small subset of the data set and see

whether imposing a metric on the semantic similarity of two types of places helps the learning algorithm better predict the top 5 labels associated with a place. An initial approach at defining the metric would have been to use the word2vec library to find the cosine distance between word tags in the projected space. A similar experiment could be done with the Flikr style data set (Karayev et al. 2013). For both of these, pre-trained model's are available in the caffe-model zoo. Thus, by initialising the weight's of the network using these models, and only changing the top most loss layer to be the Wasserstein loss layer, we can save a lot of computation resources. The above highlights a key feature of our work - because we implemented this as a layer in Caffe, users can use this with any model in the Caffe Model zoo, allowing other people to also explore the properties of this loss function.

7. Conclusion

In this paper, we describe our experiences in implementing the Wasserstein loss layer and using it to perform supervised learning for multi-class classification problems. We performed our own implementation of the loss function, and this allowed us to understand key properties of it like its dependence on changing λ, sinkhorn iterations etc. We tested it and benchmarked its performance on some toy data sets, noting its convergence rates and accuracies in comparison to other loss functions. To replicate the results from (Frogner et al. 2015), we studied its performance on the MNIST data set, where the changing p norm provided a lot of insight on the properties of the loss function. We also studied its performance on the noisy label data set, where we showed that it provides robustness to noisy input label's by utilising the distance metric in the label space. Our results thus support the hypothesis that the Wasserstein loss helps the learning model when the labels space has a natural metric associated with it.

8. Acknowledgements

We would like to thank Chiyuan Zhang and Charlie Frogner (two of the authors in (Frogner et al. 2015)) for their guidance in the project. They recommended performing an implementation in Caffe as that would be most useful to the community, and provided us help whenever we had a question about the work in their paper.

References

C. Frogner, C. Zhang, H. Mobahi, M. Araya-Polo, and T. A. Poggio. Learning with a wasserstein loss. In *Advances in neural information processing systems*, 2015.

Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

S. Karayev, M. Trentacoste, H. Han, A. Agarwala, T. Darrell, A. Hertzmann, and H. Winnemoeller. Recognizing image style. *arXiv preprint arXiv:1311.3715*, 2013.

Y. Lecun and C. Cortes. The MNIST database of handwritten digits. URL `http://yann.lecun.com/exdb/mnist/`.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

B. Zhou, A. Khosla, A. Lapedriza, A. Torralba, and A. Oliva. Places2: A large-scale database for scene understanding, 2015. URL `http://places2.csail.mit.edu`.