**The Multiverse Loss for Robust Transfer Learning**

Written by Etai Littwin and Lior Wolf (TAU) reproduced by Hillel Mendelson

<https://arxiv.org/pdf/1511.09033v2.pdf>

<http://cmp.felk.cvut.cz/cmp/events/colloquium-2016.03.31/wolf-cmp_colloq-2016.03.31>

<https://github.com/shillel/Multiverse_Loss>

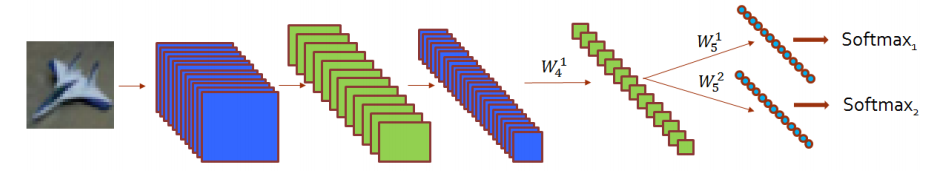
When doing transfer learning, one usually takes most of the network as-is, and adds a new representation layer which is then trained with the new data.   
The paper examines the following question: Can we determine the size of the new layer automatically from the data?

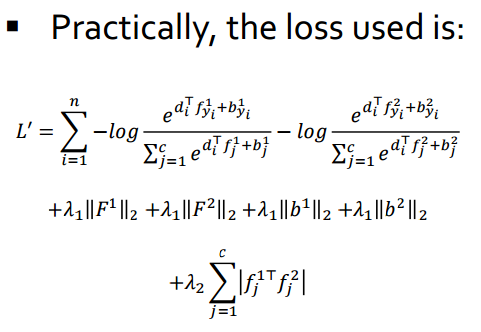
It is a well-known fact that many of the weights in a DNN are redundant, and many compression methods have been shown to reduce the network size by up to 90% (teacher-student, pruning). The cons are:

1. These methods always work after the original network has been trained, as it is impossible to attain the same results trying to train the small network directly.
2. pruning the smallest values of a weight vector results in a loss of information which leads to less accuracy.
3. It is unknown a-priori by how much the vectors could be compressed, so the user usually does a long trial and error procedure.

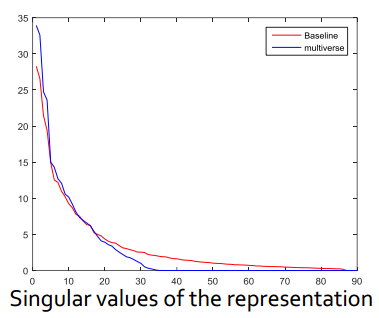
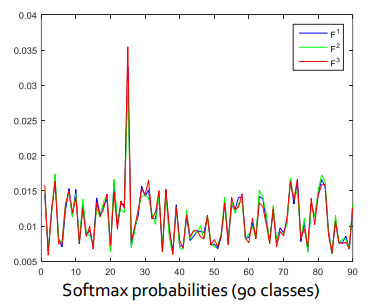
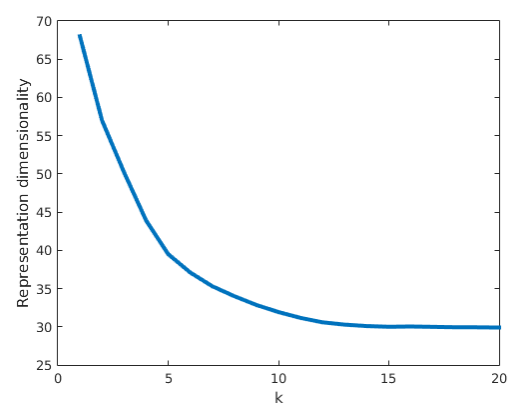
This paper takes a different approach:

1. The representation layer vector is duplicated. Before entering the LogSoftMax layer.
2. The loss is a sum of
   1. The original loss of each vector
   2. A dot product of the two vectors – this causes as many values as possible to be zeroed out, in the attempt to minimize the function.
3. The result is a “natural” dimension of the data set that emerges, without the user having to manually search for it.





**Original Paper’s Results:**

1. Per data set, the dimension of the data is determined. In the final model, a representation layer of this size could be used, without loss of “energy”.
2. The authors found that both representation vectors yield the same category. This means that in the final model, one of the vectors could be dropped, to save even more space and calculation time.
3. More than 2 representation vectors could be used, which yields an even smaller output dimension. This dimension converges as the numbers of vectors rise, as it is not possible to compress the data infinitely.

**My Work:**

I started with the mnist model we’ve built for HW1:

(1): nn.View(784)

(2): nn.Linear(784 -> 64)

(3): nn.ReLU

(4): nn.Linear(64 -> 128)

(5): nn.ReLU

(6): nn.Linear(128 -> 10)

(7): nn.LogSoftMax

Instead of layers 6-7, I separated into 2 vectors:

(1): nn.View(784)

(2): nn.Linear(784 -> 64)

(3): nn.ReLU

(4): nn.Linear(64 -> 128)

(5): nn.ReLU

(6): nn.ConcatTable {

input

|`-> (1): nn.Sequential {

| [input -> (1) -> (2) -> output]

| (1): nn.Sequential {

| [input -> (1) -> output]

| (1): nn.Linear(128 -> 10)

| }

| (2): nn.LogSoftMax **- original criterion**

| }

|`-> (2): nn.Sequential {

| [input -> (1) -> (2) -> output]

| (1): nn.Sequential {

| [input -> (1) -> output]

| (1): nn.Linear(128 -> 10)

| }

| (2): nn.LogSoftMax **- original criterion**

| }

`-> (3): nn.Sequential {

[input -> (1) -> (2) -> output]

(1): nn.ConcatTable {

input

|`-> (1): nn.Sequential {

| [input -> (1) -> output]

| (1): nn.Linear(128 -> 10)

| }

`-> (2): nn.Sequential {

[input -> (1) -> output]

(1): nn.Linear(128 -> 10)

}

... -> output

}

(2): nn.PairwiseDistance **- new criterion to make vectors orthogonal**

}

... -> output

}

}

I used parallel criterion to do a weighted sum over the three outputs:

crit1 = nn.MultiMarginCriterion()

crit2 = nn.MultiMarginCriterion()

crit3 = nn.HingeEmbeddingCriterion()

criterion = nn.ParallelCriterion():add(crit1, 0.5):add(crit2, 0.5):add(crit3, 1)

This added just 3000 parameters to the model (most of which could later be removed).

**My Results:**

I wanted to reproduce the graphs from the article. To do this I had to get the weight of each point in the representation vector (128x10). I summed over the 2nd dimension to get the total per point:

A point in the 128-length vector that has a sum ~0 is considered to have no effect on any of the 10 categories.  
It is clear, that without constraining the vector, no values are zeroed out.

When adding the 2nd vector, the picture changes:

**~30 of the 128 possible values were set to 0 by the optimization, setting the dimension at ~100.**

To verify the correctness, I forwarded randomly selected images through the model and compared the 2 vectors outputs, relative to each other and to the known label:

Unsurprisingly, the 2 vectors yield the same output which is equal to the label – 3.

I continued to experiment, adding a 3rd vector to further reduce the dimension:

Here, the results show ~50 zeroed out values, setting the dimension ~70.  
Note: vecA is on the smaller 2nd axis – most of the values on it were close to 0, and yet it yields a correct answer when used to classify:

With 4 vectors, the dimension is down to 50:

All 4 experiments on the same graph:

I also repeated the experiment for the NIN we used on the cifar10 classification assignment:

It yielded similar results, although much less values were zeroed out. This is probably because the natural dimension of the problem is very close to the size of the representation vector.

**Conclusion**

This was a very well written article. It has a strong mathematical background, and shows good experimental results (unlike many articles in this field which are more “trial-and-error”). The main difficulty was to implement the tree-like structure of the network and the parallel criterion.   
My results on the mnist data set proved to be in line with those of the writers.  
I also ran the training for a long time on the NIN model to “prove” that there is no loss of accuracy. It reached 82.5% after 250 epochs which is very much like our original NIN model.

**Some Ideas for Future Work**

This method could be expanded:

1. Linear networks: use it in every layer of the network do determine its natural size
2. Linear layers in complicated networks: wherever they are used (e.g. multi-layer LSTM) we could use this method to reduce the size of the linear layer.
3. CNNs: it could be used on filters to make them more unique, and get rid of some of them by zeroing them out. Maybe consider each filter as a point in a representation vector and use 2 orthogonal sets.