# Unsupervised Data Augmentation Experiments

Michael Morris

November 27, 2019

## 1 Semi Supervised Learning

Semi supervised learning is a method of training machine learning models which leverages unlabelled data. This reduces the labelling cost of an expert labeller and means that datasets with large amounts of unlabelled data can be used. Historically this has been done by generating a decision boundary based on the positions of unlabelled data in relation to labelled data.

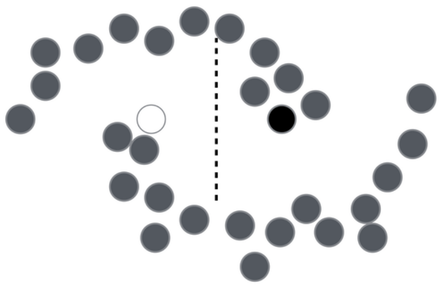

Figure 1: Decision boundary with unsupervised examples

The unlabelled datapoints can be categorised correctly by their spatial relation to the two labelled samples. The methods to do this are based on the following assumptions:

- Points which are close together are likely to be in the same class
- The data tends to form distinct clusters
- The data can be expressed in fewer dimensions than its inputs.

These lead to pseudo labelling where the model is updated as if these labels are correct.

## 2 Unsupervised Data Augmentation

Unsupervised Data Augmentation (UDA) is a method of leveraging unlabelled data based on state of the art data augmentation policies. In UDA unlabelled data is augmented: $\hat{x} = q(x, \epsilon)$ and the divergence in outputs between the original and augmented data is minimised for consistency training. This follows the assumption that the original and augmented samples have the same class, and by learning distinguishing features the distance between the labelled and unlabelled data will be minimised. The loss function for the labelled data is categorical cross entropy. For the unlabelled data is the Kullback-Leibler divergence between the original and augmented data. The unsupervised loss is weighted by a variable $\lambda$ giving the full objective function where $U$ and $L$ represent unlabelled and labelled data respectively:

$$\min_{\theta} \mathcal{J}(\theta) = \mathbb{E}_{x,y^* \in L}[-\log p_\theta(y^*|x)] + \lambda \mathbb{E}_{x \in U} \mathbb{E}_{\hat{x} \sim q(\hat{x}|x)} [\mathcal{D}_{KL}(p_{\tilde{\theta}}(y \mid x) \parallel p_\theta(y \mid \hat{x}))] \tag{1}$$

Where the KL divergence is:

$$\mathcal{D}_{KL}(P \parallel Q) = \sum_{x \in X} P_{(x)} \log \left( \frac{P_{(x)}}{Q_{(x)}} \right) \tag{2}$$

The training objective is illustrated in Figure 3 and leads to the pseudocode in Algorithm 1

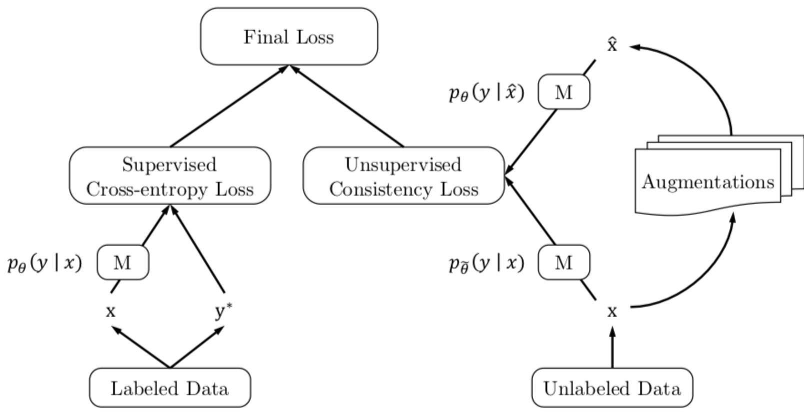
Final Loss
$p_\theta(y \mid \hat{x})$
M
$\hat{x}$
Supervised Cross-entropy Loss
Unsupervised Consistency Loss
Augmentations
$p_\theta(y \mid x)$
M
$p_{\tilde{\theta}}(y \mid x)$
M
x
y*
x
Labeled Data
Unlabeled Data

Figure 2: Training objective for UDA

## 2.1 Training Signal Annealing

A common problem when running UDA is overfitting to the few labelled training samples. To combat this a technique Training Signal Annealing (TSA) is used which gradually releases training signals from labelled data into the loss function. When the model predicts the correct class of a training example with a confidence above a set threshold it is removed from the loss function. The threshold is increased during training from a worst case of $\frac{1}{K}$ to 1. Where $K$ is the number of classes in the dataset. For the $t^t h$ iteration out of $T$ total iterations the threshold $n_t$ is $\alpha_t * (1 - \frac{1}{K}) + \frac{1}{K}$. $\alpha_t$ is either: logarithmic $1 - \exp(\frac{t}{T} * 5)$, linear $t/T$, or exponential: $\exp((tT - 1) * 5)$. The fewer labelled training samples you have the more slowly the training samples should be released.

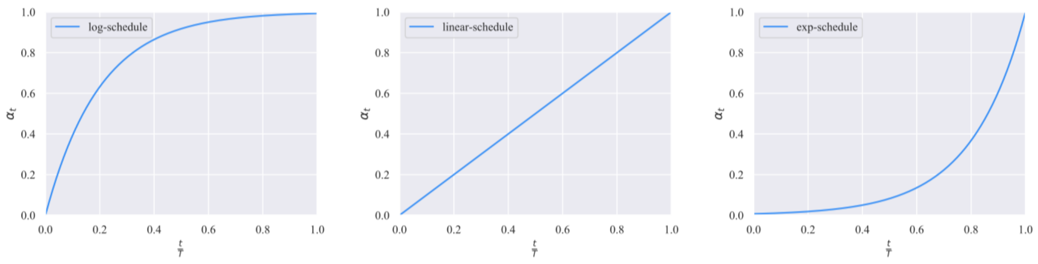

Figure 3: TSA Schedules

**Algorithm 1:** UDA Algorithm

```
duplicate data so equal number of potential batches in labelled and unlabelled sets;
shuffle and split into batches;
for epochs do
    for batches do
        if labelled batch then
            calculate $p_\theta(y^*|x)$;
            $Lloss = \mathbb{E}_{x,y^* \in L}[-\log p_\theta(y^*|x)]$;
            Apply training signal annealing
        end
        if unlabelled batch then
            create $\hat{x}$ by augmenting $x$;
            $Uloss = \lambda \mathbb{E}_{x \in U} \mathbb{E}_{\hat{x} \sim q(\hat{x}|x)}[\mathcal{D}_{KL}(p_{\tilde{\theta}}(y \mid x) \parallel p_\theta(y \mid \hat{x}))]$;
        end
        minimise $Uloss + Lloss$;
    end
    calculate accuracy from labelled validation set;
end
```

## 2.2 Experiments

Experiments were carried out on the MNIST and CIFAR10 datasets, a convolutional neural network with 3 convolutional layers, 1 hidden layer and a softmax output layer was used for both, the only difference being the shape of the inputs (MNIST: 28,28,1, CIFAR10: 32,32,3).

### 2.2.1 MNIST

The MNIST dataset was split into 42000 training samples, and 18000 testing samples, within the training set 10% was reserved for cross validation. With a fully supervised learning using cross entropy as the loss function an error rate of 0.458% was achieved. The same network was trained with a labelled/unlabelled split of 60/41800 and achieved a semi supervised error rate of 22%, The supervised learning model achieved similar results with the same error rate of 22%

## 2.3 CIFAR10

The CIFAR10 dataset was split into 10000 test samples and 50000 training samples where 4000 are labelled and 46000 unlabelled to replicate the experiments in the original paper. With this sized labelled dataset it was found that a logarithmic TSA signal gave the best results while an exponential signal increased overfitting. The best semi-supervised results obtained were a 52% error rate on the validation set - considerably less than the 4% error rate obtained in the original paper. Using this same split and dropping all of the unlabelled results gave an equal error rate of 52%. For both these models different hyper-parameters were trialled without success.

## 2.4 future work

There must be errors in the code, the same experiments will be carried out using the code written by the research team at Google on a much larger network. A similar method called MixMatch will be tried, this combines MixUp, UDA, and a sharpening algorithm, once again initial tests of this algorithm were not promising.