COMP5328 - Assignment 1 - Label Noise

Tutors: Jeremy Gillen, Yu Yao
Group members: Kilian Liss (500412817) - Alexander Mars (450347582) - James Wood (480344386)

Abstract

Learning from data with noise is a fundamental problem of machine learning. Many real world data sets that involve classification tasks often contain noisy labels, and thus algorithms which are robust to label noise are essential. In this study, we explore how we can make various neural network models more robust by implementing transition matrices, and label noise robust loss functions. We test different neural networks which range in their complexity, and explore three different methods for estimating the transition matrix, two of which relied on particular anchor points, and the other relied on many anchor points and averaging many transition matrices. We also test three different label noise robust loss functions which have varied properties, and are simpler to integrate into common machine learning models than transition matrices. We consistently found our models performing better on label noise when they took advantage of the transition matrices, and suspect that label noise robust loss functions require more time to fit noisy data compared to models that use transition matrices.

1 Introduction

Machine learning models are being applied to solve complex problems across a myriad of fields. For tasks such as genetic variant prediction, or cancer identification, highly accurate models are required to ensure that a model's predictions results in positive real world outcomes. However, machine learning models are commonly trained and evaluated using benchmark data sets such as ImageNet [3] or MNIST [4]. These data sets do not always reflect the context in which machine learning models are applied, where data can be corrupted and thus contain label noise. In classification problems, the data used in studying algorithms is often very clean, but much of the big data generated in the real world contains incorrect labels. In theory we could try to clean the data by manually correcting the labels, however some data sets are so large that cleaning them is infeasible. Furthermore, there may be human error introduced into the manual cleaning procedure.

Different types of label noise pose different problems to machine learning algorithms. Random classification noise (RCN) is defined by all classes having equal probability of being incorrectly labelled. For a set of classes with RCN, we say that the probability of any correct label Y having an incorrect label $\tilde{Y}$ is a constant value ρ. This can be formulated as $\rho_Y(X) = P(\tilde{Y}|Y, X) = P(\tilde{Y}|Y) = \rho$ for any label, instance pair Y, X. If the presence of RCN is known, using deep learning can provide a robust approach to learning with RCN [6], thus this type of label noise is less problematic.

Another type of label noise is class conditional noise (CCN), where the probability of a class having noisy labels is dependent on the class itself. In the context of CCN, the probability $P(\tilde{Y} = i|Y = j, X)$ where $i \neq j$ is called the flip rate, and tells us the probability that a label j will be incorrectly classified as label i.

There are a number of methods that aim to create classifiers which are robust to CCN, including using robust loss functions [7], loss-correction methods [11], and sample re-weighting methods [8]. Robust loss functions generally add a regularisation term to try and penalise noisy labels [7], and are beneficial as they can be seamlessly integrated into many existing machine learning algorithms. Sample re-weighting methods aim to re-weight the importance of data points fed into the algorithm, with the aim of making data points with clean labels more important, and data points with noisy labels less important [9]. This often involves using two algorithms: one that learns to re-weight the samples, and one that learns to predict classes, and is thus a more computationally intensive approach. Another approach is to use a transition matrix $T(x)$ where $T_{ij}(x) = P(\tilde{Y} = i|Y = j, X)$ [10]. With a transition matrix, the clean class posterior $P(Y|X)$ can be inferred from the noisy class posterior $P(\tilde{Y}|X)$ which can be learned directly from the noisy data [10, 11].

In this study, we focus on loss correction methods that use a transition matrix, and compare these methods to robust loss function approaches. We make use of two versions of the fashion MNIST data set [12] with known flip rates, each with different proportions of class conditional label noise, and a sample of the CIFAR data set [13] with unknown flip rates to test our methods with. Furthermore, we implement these approaches using neural networks with differing levels of complexity to examine how overall model complexity may impact the results of these approaches.

Notably, CCN is not likely to be the best approximation of real-world noise. This is because CCN (and RCN) assumes that the label noise is independent of the features. This is often not the case, consider the problem of trying to classify two species of cat. They are more likely to be incorrectly labelled as one another if a particular feature image looks similar to another class [14]. Recent methods aim to handle this type of noise [14], however these methods are not in the scope of this work.

2 Related Works

2.1 Transition Matrix Estimation

The idea of using a transition matrix to correct the output of a deep neural network when learning with noisy labels is introduced in [11]. They showed how this approach could be applied to multi-class classification problems such as MNIST. A key assumption of their work was that the data must contain 'perfect' examples for each class. These are examples whose labels are certain. Formally, an example $x^i \in X$ is an anchor point for the i-th class of Y if $P(Y_i|x^i) = 1$. Then the transition matrix can be obtained from

$$P(\tilde{Y}|x^i) = \sum_k^c P(\tilde{Y}|Y = k, x^i)P(Y = k|x^i) = P(\tilde{Y} = j|Y = i, x) = T_{ij}, \tag{1}$$

as presented in [11]. This holds from the anchor point assumption as $P(Y = k|x^i) = 1$ when $k = i$ and is equal to 0 otherwise. This also relies on the assumption of class conditional noise, where $P(\tilde{Y} = i|Y = j, X) = (\tilde{Y} = i|Y = j)$ as the flip rate is assumed to be instance-independent.

From equation 1, the transition matrix can be estimated directly from the noisy class posterior $P(\tilde{Y|X}$. Thus as noted in [11], the output of a softmax function from a neural network trained on data with noisy labels can be used to estimate the transition matrix. The output of the softmax $\tilde{x}^i = \arg\max_{x \in X}, P(\tilde{Y} = i|X)$ is then used to find $T_{ij} = P(\tilde{Y} = j|\tilde{x}^i)$. They then describe two methods for using the transition matrix, namely the forward and backward learning algorithms.

In the backward learning algorithm, for some loss function $l(P(Y|X)$, a backward corrected loss is given by $T^{-1}l(P(Y|X))$, yielding an unbiased loss correction. However, there is no guarantee that the transition matrix T is invertible. In the forward learning algorithm, the loss function becomes dependent on T so that a network learns the clean class posterior multiplied by T. We reproduce the

example for cross-entropy given in [11]. In cross-entropy the loss function is given by

$$\begin{aligned} l(i, P(Y|X)) &= -\log P(\tilde{Y} = i|X) \\ &= -\log \sum_{j=1}^{c} P(\tilde{Y} = i|Y = j)P(Y = j|X) \\ &= -\log \sum_{j=1}^{c} T_{ij} P(Y = j|x) \end{aligned} \tag{2}$$

The main drawback of this approach is that it assumes we have known anchor points, with a posterior probability close to 1.

It was proposed [1] that we can avoid this assumption by treating points with a high noisy class posterior probability like as though they were anchor points. A slack variable is then added which is learned with the classifier. The intuition to this approach is that a good transition matrix will yield a small classification risk when applied to clean data. Importantly, the T-revision method does not require inversion of the transition matrix, making it computationally favourable. However, it is more complicated to implement than the forward learning algorithm, or a robust loss function approach. A similar method which also aims to learn anchor points from noisy data is described in [15].

However these approaches still rely on estimating the transition matrix using the noisy class posterior. A more recent approach described in [2] also proposes to estimate the transition matrix without directly learning it. In [2], they note that the decomposition of the product $\tilde{P} = T \times P$ where $\tilde{P}$ is the noisy posterior, P and P is the clean posterior, is not always unique, making methods that estimate the transition matrix T from the noisy posterior $\tilde{P}$ unreliable. They instead introduce a regularisation on the predicted probabilities so that these predicted probabilities are more distinguishable from each other. This is under the observation that the cleanest posterior has the property that pairs of points in the cleanest posterior are more distinguishable from each other.

In this study, we draw inspiration from [1], although we do not use a slack variable and the T-revision method described. Instead we use the cutoff method proposed in [11] that yields accurate transition matrices. We also introduce what we believe to be a novel approach to transition matrix estimation, that takes inspiration from the cutoff method. See the methods section for details.

2.2 Robust Loss Functions

A number of loss functions have been proposed in order to reduce the impact of label noise. Previous work has shown that mean absolute error (MAE) is robust to label noise [16], however it tends to under fit data when compared to a non-robust loss function such as cross-entropy [7]. Seeing that MAE is theoretically robust, [17] proposed an improved MAE. They observed that the weight variance of MAE is low. Meaning that there is little difference in the weight it places on different samples. Due to this low weight variance, informative samples cannot contribute enough against non-informative ones. They hypothesised that this is the cause of the under fitting problem for MAE, and subsequently proposed an exponential weighting of the MAE sample weighting to increase the variance whilst retaining the noise robust properties. However, it requires an additional tuning step to determine the magnitude of the exponential weighting, and the implementation details are not clearly described, thus it was not implemented in this paper.

Another robust loss function is the symmetric cross entropy (SCE) loss function. For a data set with K classes, true distribution P, and noisy distribution $\tilde{P}$, the cross entropy (CE) loss is given by

$$l_{ce} = -\sum_{k=1}^{K} P(k|X) \log \tilde{P}(k|X).$$

They observed that CE loss does not learn noisy classes uniformly well, and particularly under learns hard classes. They take inspiration from the Kullback-leibler divergence to propose SCE loss, which includes a robust reverse cross-entropy (RCE) function. SCE is defined as

$$l_{sce} + l_{ce} + l_{rce},$$

where the RCE is defined as

$$l_{rce} = -\sum_{k=1}^{K} \tilde{P}(k|X) \log P(k|X).$$

They propose that the CE and RCE need to be weighted by two parameters α and β respectively, which should be tuned for the respective data set. However, due to the true distribution being included inside a $\log$ function, they note that applying this method with one hot encoded classes can produce undefined behaviour. To remedy this, they propose using a constant parameter A such that $\log 0 = A$ where $A < 0$. Thus, this is an additional parameter which requires tuning, and was subsequently not implemented in this paper.

In this study, we use the generalised cross entropy function proposed in [7] which combines properties of the MAE and cross entropy functions to create a label robust loss function. We also utilise the active passive loss framework introduced in [18], which claims that the previous works such as symmetric and generalised cross entropy are only partially robust to noise. Finally, we compare these more complex loss functions to MAE as a baseline theoretic robust loss function. The details of the loss functions used in this study are given in the methods section.

3 Methods

3.1 Data sets

We test our algorithms using three different data sets of images, each with 3 different label classes but different label noise distributions. These data sets are:

- **FashionMNIST0.3**: Which has a known transition matrix of:

$$\begin{bmatrix} 0.7 & 0.3 & 0 \\ 0 & 0.7 & 0.3 \\ 0.4 & 0 & 0.7 \end{bmatrix}, \tag{3}$$

  with which we can measure the estimation error of our estimated transition matrices.

- **FashionMINIST0.6**: Which has a known transition matrix of:

$$\begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.3 & 0.4 & 0.3 \\ 0.4 & 0.3 & 0.4 \end{bmatrix}, \tag{4}$$

  with which we can measure the estimation error of our estimated transition matrices.

- **CIFAR**: For which we do not know the exact transition matrix.

For both MNIST data sets, the labels correspond to:

- 0: Top Piece of Clothing
- 1: Shoes
- 2: Wallet and Purses

Meanwhile for the CIFAR dataset, the labels correspond to:

- 0: Planes
- 1: Cars
- 2: Cats

For the FMNIST data set, there were 3000 test images, and 18,000 training images. For the CIFAR data set there were 3000 test images and 15,000 training images.

3.2 Classifier Models

We built various classifier models with both PyTorch and Tensorflow. With our Tensorflow models, we built a simple neural network (NN) with one hidden layer, and a convolutional neural network (CNN) with two convolution layers and one fully connected layer. For each NN and CNN models, we built additional identical models but with an extra final layer that multiplies the output by transition matrices before the data gets sent to the loss function. All our tensorflow models use the Adam (adaptive moment estimation) optimiser, and the ReLU (Rectified Linear Unit) activation functions.

The structure of our tensorflow NN and CNN models are summarised bellow:

3.2.1 Simple Neural Network

- Flatten - To flatten images into vectors
- Dense layer with 128 nodes
- Drop-out with probability of 0.2
- Dense Layer with 3 nodes, one for each label
- Softmax layer to convert logits into probabilities

3.2.2 Convolution Neural Network

We will also use a convolution neural network, as it is generally considered that they work well on datasets of images such as ours:

- Convolution layer with 32 filters, each with strides of size 4
- A max pooling layer
- Convolution layer with 16 filters, each with strides of size 3
- A max pooling layer
- Flatten
- Dense layer with 128 nodes
- Drop-out with probability of 0.2
- Dense Layer with 3 nodes, one for each label
- Softmax layer to convert logits into probabilities

3.2.3 PyTorch Models

In PyTorch, we built more complex models, creating a replica of the LeNet model [19] for use with the FMNIST data, and a replica of ResNet18 [20] for use with the CIFAR data. Furthermore, PyTorch allows for implementing customised loss functions more easily than tensorflow, thus our robust loss function experiments were run exclusively with the PyTorch models. For the details of the LeNet and ResNet18 models, see the code files provided.

For all models, a softmax was applied to the output for use with the forward learning algorithm as described in section 2.1.

Building two sets of models in tensorflow and PyTorch allowed us to conduct different experiments simultaneously, and also allows for a comparison of results to be made based on model complexity, as both LeNet and ResNet18 are more complex than the simple NN and CNN models built with tensorflow, in that they have more convolutional layers, and subsequently a larger number of parameters.

3.2.4 Training Schedules

For all models, we use 5 epochs to learn the noisy class posterior and estimate the transition matrix, this is the first training stage. This is based on an experimental result explained in section 4.1. For the tensorflow models, as they are simpler, we use only 5 epochs of forward learning with the estimated transition matrix. For the LeNet and ResNet18 models we use 25 forward epochs to ensure they

appropriately fit the data. The forward learning with the estimated transition matrices is the second training stage. For all models and each training stage, we save the best model during training based on the validation loss. The best model from the first stage is then used to estimate the transition matrix, and the best model from the second stage is used to calculate the test set accuracy.

3.2.5 Adam Optimiser

Adam is an adaptive stochastic gradient based optimisation method, where the learning rate at each step is updated based on the gradient and squared gradient [21]. The update rule for the step size is $\triangle_t = \alpha \cdot \hat{m}_t / \sqrt{\hat{v}_t}$ where t is the time step, α is the step size, and $\hat{m}_t$ and $\hat{v}_t$ are the exponential moving averages of the gradient and squared gradient respectively at time step t. An important feature of Adam is that when an optimum is neared, the ratio $\hat{m}_t / \sqrt{\hat{v}_t}$ approaches zero, leading to updates such that $\alpha_{t+1} \leq \alpha_t$. In other words, the step size automatically decreases as the gradient flattens, providing a form of automatic annealing.

We used the adam optimisation method for all models and all training stages to avoid manually tuning the learning rate schedule as would be required in many other optimisation methods.

3.3 Estimating the Transition Matrix

Given a data set of C different labels, the transition matrix can be defined as:

$$T = \begin{bmatrix} P(\tilde{Y}=1|Y=1) & P(\tilde{Y}=1|Y=2) & \dots & P(\tilde{Y}=1|Y=C) \\ P(\tilde{Y}=2|Y=1) & P(\tilde{Y}=2|Y=2) & \dots & P(\tilde{Y}=2|Y=C) \\ \vdots & \vdots & \ddots & \vdots \\ P(\tilde{Y}=C|Y=1) & P(\tilde{Y}=C|Y=2) & \dots & P(\tilde{Y}=C|Y=C) \end{bmatrix} \tag{5}$$

which describes the label noise distribution within our data set.

We can relate the noisy class posterior $P(\tilde{Y}|X)$ and the clean class posteriors $P(Y|X)$ using the transition matrix:

$$P(\tilde{Y}|X) = T\ P(Y|X) \tag{6}$$

where:

$$P(\tilde{Y}|X) = \begin{bmatrix} P(\tilde{Y}=0|X) \\ P(\tilde{Y}=1|X) \\ \vdots \\ P(\tilde{Y}=C|X) \end{bmatrix} \quad \& \quad P(Y|X) = \begin{bmatrix} P(Y=0|X) \\ P(Y=1|X) \\ \vdots \\ P(Y=C|X) \end{bmatrix} \tag{7}$$

If we have a data features x^j such that $P(Y=j|X=x^j)=1$ and $P(Y\neq j|X=x^j)=0$, then $P(Y|X)$ becomes a one hot vector at the j^{th} row, and so equation 6 becomes:

$$P(\tilde{Y}|X=x^j) = \begin{bmatrix} P(\tilde{Y}=1|Y=1) & P(\tilde{Y}=1|Y=2) & \dots & P(\tilde{Y}=1|Y=C) \\ P(\tilde{Y}=2|Y=1) & P(\tilde{Y}=2|Y=2) & \dots & P(\tilde{Y}=2|Y=C) \\ \vdots & \vdots & \ddots & \vdots \\ P(\tilde{Y}=C|Y=1) & P(\tilde{Y}=C|Y=2) & \dots & P(\tilde{Y}=C|Y=C) \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \tag{8}$$

$$= \begin{bmatrix} P(\tilde{Y}=0|Y=j) \\ \vdots \\ P(\tilde{Y}=C|Y=j) \end{bmatrix} \tag{9}$$

and so the j^{th} column of the transition matrix can be read from $P(\tilde{Y}|X=x^j)$. We refer to the points x^j as anchor points.

3.3.1 Identifying Anchor Points

The approach discussed here is inspired by [11]. The simplest way of obtaining anchor points is to manually choose a small set of features for which we are absolutely certain that a particular label is correct. Although this method works quite well with our data-set, we would like to have a completely automated way of identifying anchor points. We follow the approach described in [1] by using a trained classifier model, and treating points x with a high posterior probability $P(\tilde{Y} = j|X = x)$ as anchor points.

This approach makes use of the neural network classifier models trained on data with noisy labels to obtain the list of posteriors. For both the FMNIST and CIFAR data sets where there are three classes, so the posteriors can be represented as:

$$P(\tilde{Y}|X = x) = \begin{bmatrix} P(\tilde{Y} = 0|X = x) \\ P(\tilde{Y} = 1|X = x) \\ P(\tilde{Y} = 2|X = x) \end{bmatrix}, \tag{10}$$

for each data point x in our training set. Then, for each class j, we sorted this list from lowest to highest posteriors $P(\tilde{Y} = j|X = x)$, and used the all points within a range of percentiles around $[0.95, 0.97]$ of the list as anchor points to generate many transition matrices. The final obtained transition matrix was then taken to be the mean of all obtained transition matrices.

We also generated transition matrices where the points fell on the 97-th percentile as described in [11]. We based this decision on an experiment shown in section 4.1, where we tested different percentiles in estimating the transition matrix.

We compared this approach to using the points that completely maximise $P(\tilde{Y} = j|X = x)$ as described in [1]. The posterior probabilities for these points tend to be outliers, resulting in a model that is overconfident in its classifications, leading to a an expectantly worse approximation of the transition matrix.

3.3.2 Estimation Error

To determine whether our methods for transition matrix estimation were effective, we calculated the estimation error between the given transition matrices for the FMNIST data sets, and our estimated transition matrices for these data sets. For a given transition matrix T, and an estimated transition matrix $\tilde{T}$, the estimation error is defined as

$$Est_{err} = \frac{\sum |\tilde{T} - T|}{\sum |T|}, \tag{11}$$

where the sum is over all entries in the transition matrix, and the subtraction is element wise.

3.3.3 Forward Learning Algorithm

We used the forward learning algorithm as described in section 2.1 with all transition matrices. This requires first estimating the transition matrix from noisy data, and then applying it to the output of a softmax function so that the model learns the clean class posterior multiplied by the transition matrix.

3.4 Robust Loss Functions

3.4.1 Mean Absolute Error

If the class of x is j then y is given as a one hot encoded vector e_j where $e_{ji} = 1$ if $i = j$, otherwise 0. From this definition of one hot encoding, we can define the mean absolute error (MAE) as

$$l_{mae} = ||e_j - f(x)||_1 \tag{12}$$

where $|| \cdot ||_1$ is the L_1-norm and is defined as $\sum | \cdot |$. As explained in [16], the MAE is theoretically robust to noise, but tends to under fit. However, it is simple to implement and serves as a sound baseline to compare other robust loss functions against.

3.4.2 Generalised Cross Entropy

We implemented the generalised cross entropy loss as described in [7]. The GCE is a truncated version of the negative Box-Cox transformation, which is defined as

$$L_q(f(x), j) = \frac{(1 - f_j(x)^q)}{q}, \tag{13}$$

where $q \in (0, 1]$. As $q \to 0$, the L_q loss is equivalent to cross-entropy, and when $q = 1$, it becomes the MAE [7]. They introduce a truncated L_q loss defined as

$$L_{trunc}(f(x), j) = \begin{cases} L_q(k) & \text{if } f_j(x) \leq k \\ L_q(f(x), j) & \text{if } f_j(x) > k, \end{cases} \tag{14}$$

where $0 < k < 1$ and $L_q(k) = \frac{1-k^q}{q}$. This introduces a hyper parameter k that must be tuned based on the noise level in the data set. This is because if the softmax output for a label j is below the threshold k, then the loss value is a constant value, and therefore has a gradient of zero. A larger k value thus creates tighter bounds on the function, and indicates improved robustness to noise. However, during the early training stage the value k may be too stringent and discard a mixture of clean and noisy labels. To remedy this, in [7] they propose another method which requires an alternative convex search algorithm, which is computationally intensive and thus was not implemented in this paper. Instead, we utilised $k = 0.7$ as determined in [18] on the CIFAR data set to avoid tuning the parameter k.

3.4.3 Active Passive Loss

A recent framework for creating label noise robust loss functions was proposed in [18]. They define an active loss as one that only maximises $P(Y = i|x) = 1$ where i is a class in Y, otherwise it is passive. For example, in cross-entropy, the probability $P(Y = i|x) = 1$ is maximised, making it an active loss. Whilst MAE minimises the probabilities for all $Y \neq i$ along with maximising the probability $Y = i$. An active passive loss (APL) is then simply the combination of the two, defined as $L_{APL} = L_{active} + L_{passive}$. Notably, the APL approach can overcome the difficulty of optimising robust loss functions, as explored in [7, 18]. Thus we implemented a single active passive loss by combining the GCE loss (section 3.4.2), with the mean absolute error (section 3.4.1). This is because the GCE loss is active, whilst the MAE is passive [18]. Importantly, the weight of each loss should be scaled, so that the final equation for a $GCE + MAE$ loss would be

$$L_{APL} = \alpha L_{GCE} + \beta L_{MAE} \tag{15}$$

where α and β are tuning parameters that should be determined for a specific APL on a specific data set. To avoid tuning these parameters, we used the settings described in [18] for combining cross entropy and MAE on the MNIST data set, where $\alpha = 1$ and $\beta = 100$.

3.5 Accuracy Metric

For all experiments, the accuracy reported is the top-1 accuracy. Top-1 accuracy is defined as

$$acc = \frac{\text{number of correctly classified examples}}{\text{total number of test examples}} \times 100\%. \tag{16}$$

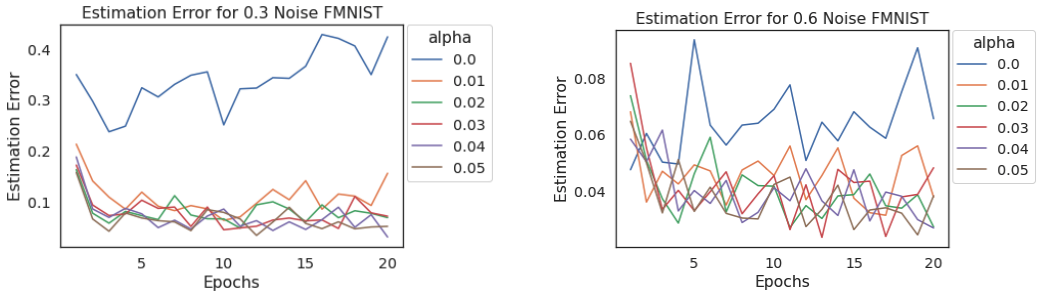
All experimental results using accuracy were calculated using 10 fold repeated trials. For each trial, 20% of the training data was randomly sampled as a validation set and used to measure model performance during training. The best model based on validation loss was selected and used to measure performance on the test set for each trial.

4 Experiments

4.1 Transition Matrix Estimation

When creating a transition matrix, there are a number of parameters that may be tuned for a data set. For example, the number of epochs for learning the transition matrix. In our approach, we also

examined the percentile of the posterior probabilities that should be used for determining anchor points as described in [11]. We used a parameter $alpha$ to represent the percentile considered. When $alpha = 0$, the points with maximum posterior probability were chosen as anchor points, as described in [1]. We were also interested in how the number of epochs spent learning the transition matrix could impact the estimation error. Thus, we ran 3-fold repeated validation across different values for $alpha$ and for a different number of epochs to try and determine an optimal combination (Figure 1).



(a) (b)

Figure 1: Estimation error for transition matrices using different percentiles, and trained for a different number of epochs. Experiments were conducted using the LeNet model.

We can see that for the less noisy FMNIST data set, there is a higher estimation error for transition matrices estimated using the maximum posterior probability ($alpha = 0$), which appears to worsen as the transition matrix is learned for a larger number of epochs. For other values of $alpha$, there is no discernible difference. However, it appears that at least 5 epochs are required for the model to effectively learn the noisy class posterior, yielding a low estimation error. We subsequently decided to learn transition matrices for 5 epochs.

We compared the approach of using $alpha = 0$, and $alpha = 0.03$ with our novel approach of averaging the transition matrices for points ranging between $alpha$ values of 0.03 and 0.05. We ran these results on the simpler tensorflow models to reduce computational cost, and provide a visual representation of the errors in approximating our transition matrices in figure 2. The colors in these plots represent the error of each estimation, and various plots are obtained for different combinations of classifier, and estimation methods used.

We can see that for the convolutional neural networks, estimating the transition matrix using a value of 0.03 or 0 for $alpha$ yields superior results to the averaging approach. However, for the neural network models without convolution layers, the averaging approach appears to yield superior results. This pattern is fixed across both FMNIST data sets. This may indicate that transition matrix estimation methods are sensitive to the choice of model, and thus thorough experimentation with different models is required to achieve the best results.

4.1.1 Example estimations of the transition matrices

Using the approaches determined in section 4.1, we produced example estimates of the transition matrices for each data set. Note that for later results, we performed 10-fold cross validation which meant 10 different transition matrices were estimated for each experiment, with the results averaged. Here we present one example of a transition matrix for each data set.

Using an $alpha$ value of 0 we estimated the transition matrices for the FMNIST0.3 data set:

$$\begin{bmatrix} 0.85 & 0.15 & 0 \\ 0 & 0.79 & 0.21 \\ 0.22 & 0 & 0.78 \end{bmatrix}, \tag{17}$$

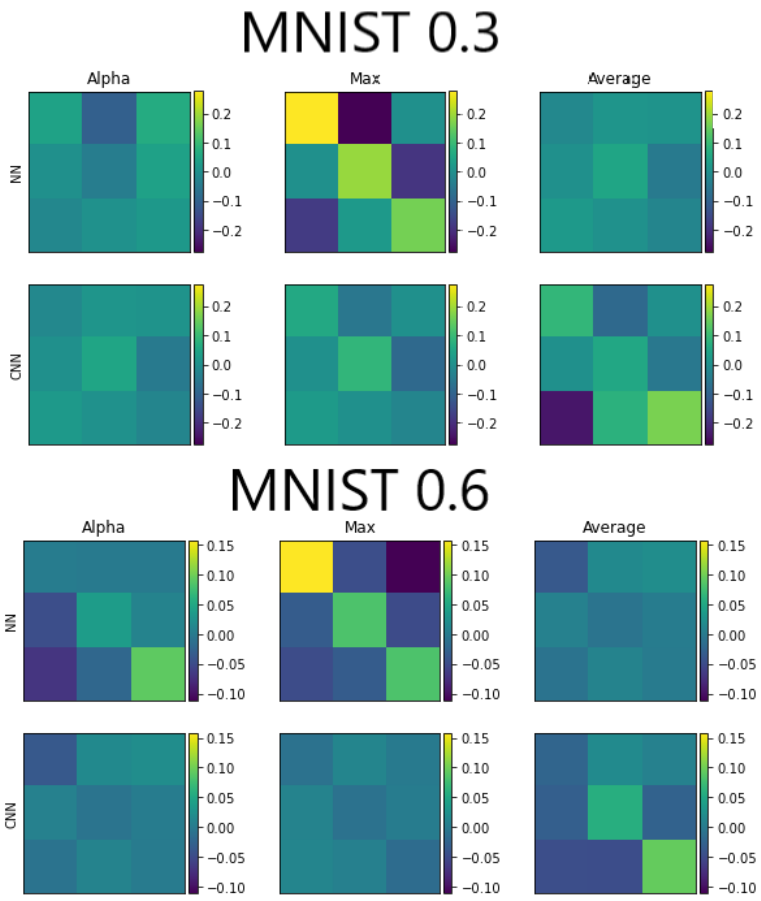


Figure 2: Comparing transition matrix estimation errors for our different estimation methods with different classifier models.

the FMNIST0.6 data set:

$$\begin{bmatrix} 0.4 & 0.3 & 0.29 \\ 0.3 & 0.41 & 0.29 \\ 0.31 & 0.29 & 0.4 \end{bmatrix}, \tag{18}$$

and the CIFAR data set:

$$\begin{bmatrix} 0.8 & 0.16 & 0.04 \\ 0.15 & 0.78 & 0.07 \\ 0.17 & 0.16 & 0.67 \end{bmatrix}. \tag{19}$$

Using an $alpha$ value of 0.03 we estimated the transition matrices for the FMNIST0.3 data set:

$$\begin{bmatrix} 0.77 & 0.21 & 0.02 \\ 0 & 0.72 & 0.28 \\ 0.31 & 0 & 0.69 \end{bmatrix}, \tag{20}$$

the FMNIST0.6 data set:

$$\begin{bmatrix} 0.39 & 0.3 & 0.31 \\ 0.3 & 0.4 & 0.3 \\ 0.31 & 0.27 & 0.42 \end{bmatrix}, \tag{21}$$

and the CIFAR data set:

$$\begin{bmatrix} 0.4 & 0.37 & 0.24 \\ 0.29 & 0.5 & 0.21 \\ 0.23 & 0.29 & 0.48 \end{bmatrix}. \tag{22}$$

Finally, using a range of $alpha$ values between 0.03 and 0.05, and the tensorflow CNN, we estimated the average transition matrix for the FMNIST0.3 data set:

$$\begin{bmatrix} 0.73 & 0.26 & 0.01 \\ 0.00 & 0.69 & 0.31 \\ 0.23 & 0.00 & 0.77 \end{bmatrix}, \tag{23}$$

the FMNIST0.6 data set:

$$\begin{bmatrix} 0.38 & 0.32 & 0.31 \\ 0.32 & 0.37 & 0.31 \\ 0.30 & 0.29 & 0.41 \end{bmatrix}, \tag{24}$$

and the CIFAR data set:

$$\begin{bmatrix} 0.35 & 0.34 & 0.31 \\ 0.28 & 0.42 & 0.31 \\ 0.30 & 0.32 & 0.38 \end{bmatrix}. \tag{25}$$

4.2 Learning with known flip rates

To determine the performance of our transition matrix estimation algorithms, we measured the accuracy achieved on the test set of the FMNIST data sets using these algorithms, and compared the results to using the supplied transition matrices. We performed 10-fold cross validation for each experiment by creating a 20% validation set from the training set. We selected the best model from the first 5 epochs of learning on the noisy labels as the model with the lowest validation loss, and used this model to create a transition matrix. We then trained each model with the forward learning algorithm and measured the final performance on the test data set.

For the transition matrix estimators using $alpha$ values of 0 and 0.03, the results for the FMNIST data set using the LeNet model are displayed in table 1.

Table 1: Results 10-fold cross-validation from using the forward learning algorithm with transition matrices on the FMNIST data set. The transition matrix type was either using an $alpha$ cutoff of 0.03 (alpha), the maximum posterior probability (max, $alpha = 0$), or the transition matrix supplied with the data set. Estimation error was calculated between the given transition matrix and the estimated one for a given level of noise. Both top-1 accuracy and standard deviation are presented. Highest accuracies are in bold.

Test Acc	Valid Acc	Train Acc	Estimation Error	Alpha	Noise
97.44(0.27)	68.27(0.58)	68.7(0.19)	0.08(0.03)	alpha	0.3
96.82(0.26)	68.4(0.51)	68.46(0.16)	0.31(0.05)	max	0.3
97.56(0.23)	68.77(0.56)	68.72(0.13)	0.0(0.0)	given	0.3
96.68(0.18)	68.46(0.45)	68.52(0.16)	—	—	0.3
94.79(0.67)	38.97(0.56)	38.93(0.12)	0.03(0.01)	alpha	0.6
92.79(2.84)	38.48(0.63)	38.98(0.11)	0.05(0.02)	max	0.6
94.15(0.69)	39.07(0.6)	38.88(0.18)	0.0(0.0)	given	0.6
92.43(4.87)	38.53(0.83)	38.8(0.26)	—	—	0.6

As expected, the estimation error across all 10-folds was low when using an $alpha$ value of 0.03. Notably, for the FMNIST0.6 data set, estimation with an alpha value of 0.03 outperformed the supplied transition matrix. As expected, all approaches using a transition matrix outperformed models that didn't use a transition matrix (control), indicating that the forward learning algorithm does provide robustness to label noise. From these results, we were confident that estimating the transition matrices using $alpha$ values of 0 and 0.03 yielded accurate transition matrices.

We then tried using the average transition matrix estimation approach with the simpler tensorflow models. The results are displayed in Table 2. We included the results of using an $alpha$ value of 0.03, and with the given transition matrices to allow for direct comparison between the LeNet model and the tensorflow models.

The results from the tensorflow models are highly similar to the results from the more complex LeNet model, indicating that the additional model complexity is not a factor in robust learning. Notably, on the FMNIST0.3 data set, the different approaches to transition matrix estimation perform

Table 2: Results of 10 fold repeated validation using the tensorflow models on the MNIST data with the given, averaged, and alpha transition matrices. Top-1 accuracy and standard deviations are displayed as the percentage of correctly classified examples. The best results for the FMNIST data set with noise 0.3 and 0.6 are highlighted in bold.

Test Acc	**Valid Acc**	**Train Acc**	**Alpha**	**Noise**	**Model**
97.91(0.25)	68.47(0.85)	69.23(0.16)	alpha	0.3	NN
88.66(11.34)	38.13(1.38)	38.68(1.19)	alpha	0.6	NN
98.35(0.36)	69.45(0.67)	69.22(0.11)	alpha	0.3	CNN
94.53(1.23)	39.18(1.41)	39.36(0.13)	alpha	0.6	CNN
98.08(0.15)	68.84(0.86)	69.26(0.08)	averaged	0.3	NN
79.69(20.08)	37.69(2.13)	37.57(2.13)	averaged	0.6	NN
98.49(0.15)	68.91(0.46)	69.30(0.10)	averaged	0.3	CNN
88.71(18.57)	38.43(2.39)	38.72(1.93)	averaged	0.6	CNN
98.07(0.24)	68.77(1.43)	69.21(0.14)	given	0.3	NN
82.66(19.65)	37.71(2.27)	37.90(2.03)	given	0.6	NN
98.24(0.26)	68.44(1.25)	69.29(0.22)	given	0.3	CNN
91.70(10.03)	38.22(0.70)	39.19(0.91)	given	0.6	CNN

similarly across the CNN and NN models. On the FMNIST0.6 data set, the CNN model performed better. Furthermore, the best result on the FMNIST0.3 data set was achieved by the averaging approach to transition matrix estimation. This is likely because the specific value of $alpha$ should be tuned [11], with the averaging approach possibly out performing an $alpha$ value of 0.03 which was only tuned using 3 fold validation. However, the estimation with an $alpha$ value of 0.03 achieved the best result for FMNIST0.6, which is consistent with the results found for LeNet.

We produced a visualisation of the results on the tensorflow models to better represent the different variation of the models from 10-fold validation. Figure 3 compares the resulting accuracies of our tensorflow models where we have used known transition matrices. The resulting test set accuracy of each trial is represented by blue dots, and statistics are represented by box plots. We can clearly see that the application of transition matrices to our model does indeed improve the average accuracy of our models.

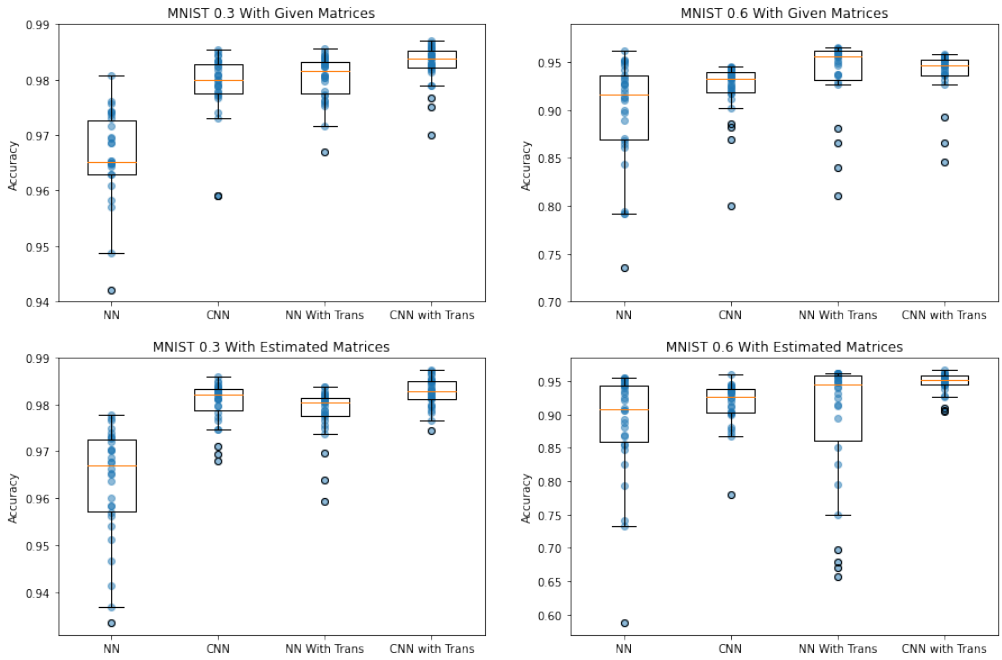


Figure 3: Resulting test data accuracies of our various tensorflow models with known flip rates and estimated. The transition matrices were estimated by averaging.

Importantly, Figure 3 highlights the large variation that can occur between repeated validation, indicating that the portion of the training sample used for learning the transition matrix can have an impact on the final result. As we still see a handful of outliers using 10 fold repeated validation, it may be more pertinent to use a larger number of repeats in future experiments to better estimate the true accuracy of each method.

4.3 Learning with unknown flip rates

Having found that our models produced accurate transition matrix estimators by comparing their performance to the supplied transition matrices in section 4.2, we applied our estimation algorithms to the CIFAR data set, for which the flip rates of each label are unknown. As in section 4.2, we used repeated validation to create 10 different transition matrices, and measured their performance on the test data set. Table 3 displays the results of using $alpha$ values of 0 and 0.03 using the ResNet18 model. Table 4 displays the results of using the averaging approach across an alpha range of 0.03 to 0.05 using tensorflow models.

Table 3: Results of 10-fold validation from using the forward learning algorithm with transition matrices on the CIFAR data set. The transition matrix type was either using an $alpha$ cutoff of 0.03 (alpha), or the maximum posterior probability (max, $alpha = 0$). Both top-1 accuracy and standard deviation are presented. Highest accuracies are in bold.

Test Acc	Valid Acc	Train Acc	Transition Matrix
44.29(3.71)	35.34(0.83)	41.31(2.22)	alpha
44.98(3.38)	36.05(0.79)	47.03(2.91)	max
49.16(2.34)	35.7(1.02)	46.12(3.4)	—

Using these approaches, we found that neither transition matrix estimator outperformed the control model with no transition matrix across both the more complex ResNet18 model, and the simpler NN and CNN models. Furthermore, the results for $alpha = 0$ (max) and $alpha = 0.03$ were almost identical. Notably, our choice of $alpha$ was based on estimation error on the FMNIST data set, along with the $alpha$ value set for CIFAR10 as used in [11]. However, the noise rate in

Table 4: Results of 10 fold validations using tensorflow models using estimated transition matrices on CIFAR data

Test Acc	Valid Acc	Train Acc	Alpha	Noise	Model
33.33(0.00)	32.27(0.88)	33.45(0.10)	alpha	CIFAR	NN
48.43(12.98)	35.44(1.26)	36.09(2.56)	alpha	CIFAR	CNN
33.33(0.00)	32.60(1.24)	33.41(0.14)	Averaged	CIFAR	NN
41.97(10.22)	35.26(1.71)	35.16(1.99)	Averaged	CIFAR	CNN

this sample of CIFAR may be vastly different from the CIFAR10 data used in [11]. An incorrect choice of $alpha$ may also be the reason why the averaging approach performs substantially worse than the $alpha = 0.03$ transition matrix estimator on the tensorflow models. Thus to improve our results, a parameter search of $alpha$ for the CIFAR data set would have been beneficial. However, we found training on CIFAR to be more computationally expensive than training on FMNIST, and subsequently did not pursue optimising $alpha$ for CIFAR.

Figure 4 visualises the results of our models with unknown transition matrices. We can see that the use of estimated transition matrices did improve prediction accuracy for the CNN model, however there is less variation for the known transition matrix. This supports the notion that our transition matrix estimators tuned to the FMNIST data set are sub optimal for the CIFAR data set.

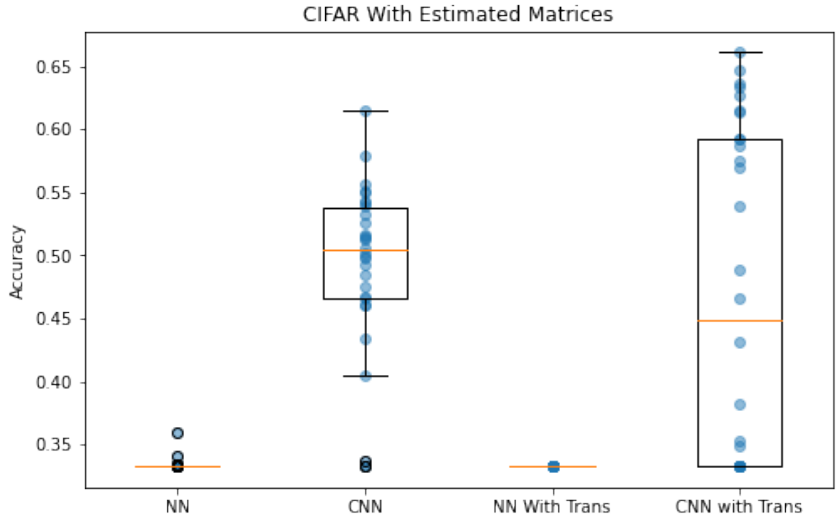


Figure 4: Resulting test data accuracies of our various tensorflow models with unknown flip rates and the CIFAR dataset

4.4 A robust loss function approach

Using a transition matrix is only one approach to learning in the presence of label noise, thus we also examined the performance of various loss functions that are purportedly robust to class conditional label noise. We tested the three loss functions presented in section 3.4 across all three data sets. For each loss function, we performed 10-fold validation and reported the mean and standard deviation of top-1 accuracy on the test set.

The results for the FMNIST data sets are displayed in Table 5, with the results for the CIFAR data set displayed in Table 6. For the FMNIST data sets, GCE achieved the highest accuracy, however for the FMNIST0.6 data set, the adaptive passive loss of GCE and MAE was close second, with half the standard deviation of GCE. Notably, when compared to the control models without transition matrices or robust losses applied (section 4.2, Table 1), all the robust loss functions appear to under fit the FMNIST data sets. For the CIFAR data set, we see a similar under fitting problem, where the

training accuracy appears to be much lower for the models with robust loss functions compared to the models with transition matrices and the control models (section 4.3, Table 3).

Table 5: Results of using robust loss functions with the PyTorch neural networks on the FMNIST data sets. Top-1 accuracy and standard deviation from performing 10-fold validation is presented.

Test Acc	Valid Acc	Train Acc	Loss Function	Noise
71.86(10.22)	58.49(3.93)	48.7(2.75)	GCE	0.3
63.06(18.08)	36.15(1.96)	35.06(0.92)	GCE	0.6
63.85(10.75)	55.1(4.73)	46.53(4.42)	MAE	0.3
39.05(11.03)	34.14(0.92)	33.93(0.27)	MAE	0.6
71.76(5.26)	58.11(2.16)	50.25(4.54)	GCE+MAE	0.3
45.35(13.41)	34.71(1.73)	34.25(0.7)	GCE+MAE	0.6

Table 6: Results of using robust loss functions with the PyTorch neural networks on the CIFAR data sets. Top-1 accuracy and standard deviation from performing 10-fold validation is presented.

Test Acc	Valid Acc	Train Acc	Loss Function
44.28(4.52)	35.48(0.95)	36.84(1.92)	GCE
37.86(3.31)	34.38(0.77)	33.88(0.82)	MAE
46.21(9.8)	34.24(1.76)	35.19(1.21)	GCE+MAE

Under fitting may be the cause of the lower final test accuracy across all robust loss function models in comparison to the transition matrix estimators [16, 7, 17, **?**]. For all the robust loss function models, we employed the same training schedule as for the models that use transition matrix estimators to ensure a fair comparison, however the robust loss functions may require longer training schedules to properly fit the data. In future, we could experiment with altering the training time of each method to try and maximise performance.

5 Conclusion

We have examined the performance of multiple different classifiers which are purportedly robust to label noise, including the forward learning algorithm using a transition matrix, and three distinct robust loss functions. We showed multiple times that the implementation of a transition matrix can be used to make neural networks more robust to class dependent label noise. Although we found transition matrix estimation to be highly sensitive to the noisy sample data. It appears that further tuning of transition matrix estimation parameters is required to achieve improved performance. As we limited the robust loss functions to using the same training schedule as the transition matrix estimators, we may have impeded their performance. Subsequent work should investigate whether robust loss functions can achieve comparable performance to transition matrix estimators using more optimised training schedules.

References

[1] Xiaobo Xia, Tongliang Liu, Nannan Wang, Bo Han, Chen Gong, Gang Niu and Masashi Sugiyama. **Are Anchor Points Really Indispensable in Label-Noise Learning?** arXiv, 2019.

[2] Yivan Zhang, Yivan Zhang, Yivan Zhang. **Learning Noise Transition Matrix from Only Noisy Labels via Total Variation Regularization** arXiv, 2021.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li; Kai Li, and Li Fei-Fei. **ImageNet: A large-scale hierarchical image database** IEEE, 2009.

[4] Deng, L. **The mnist database of handwritten digit images for machine learning research** IEEE Signal Processing Magazine, 2012.

[5] Ehsan Mohammady Ardehaly and Aron Culotta. **Learning from noisy label proportions for classifying online social data** Social Network Analysis and Mining, 2018.

[6] Gorkem Algan and Ilkay Ulusoy. **Label Noise Types and Their Effects on Deep Learning** arXiv, 2020.

[7] Zhilu Zhangand Mert Sabuncu. **Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels** NeurIPS, 2018.

[8] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. **Learning to Reweight Examples for Robust Deep Learning** ICML, 2018.

[9] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. **MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels** ICML, 2018.

[10] Tongliang Liu and Dacheng Tao. **Classification with Noisy Labels by Importance Reweighting** IEEE, 2015.

[11] Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. **Making Deep Neural Networks Robust to Label Noise: a Loss Correction Approach** CVPR, 2017.

[12] Han Xiao, Kashif Rasul, and Roland Vollgraf. **Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms** Computer Vision and Pattern Recognition, 2017.

[13] Alex Krizevsky and Geoffrey Hinton **Learning Multiple Layers of Features from Tiny Image** PhD Thesis, 2009.

[14] Yikai Zhang, Songzhu Zheng, Pengxiang Wu1, Mayank Goswami, and Chao Chen. **Learning with Feature-Dependent Label Noise: A Progressive Approach** ICLR, 2021.

[15] Yu Yao, Tongliang Liu1, Bo Han, Mingming Gong, Jiankang Deng, Gang Niu, and Masashi Sugiyama. **Dual T: Reducing Estimation Error for Transition Matrix in Label-noise Learning** arXiv, 2021.

[16] Aritra Ghosh, Himanshu Kumar, and P.S. Sastry. **Robust Loss Functions under Label Noise for Deep Neural Networks** AAAI, 2017.

[17] Xinshao Wang, Yang Hua, Elyor Kodirov, and Neil M. Robertson. **IMAE for Noise-Robust Learning: Mean Absolute Error Does Not Treat Examples Equally and Gradient Magnitude's Variance Matters** CVPR, 2020.

[18] Xingjun Ma, Hanxun Huang, Yisen Wang, Simone Romano, Sarah Erfani, and James Bailey **Normalized Loss Functions for Deep Learning with Noisy Labels** ICML, 2020.

[19] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. **Gradient-Based Learning Applied to Document Recognition** PROC. OF THE IEEE, 1998.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. **Deep Residual Learning for Image Recognition** CVPR, 2015.

[21] Diederik P. Kingma, and Jimmy Ba. **Adam: A Method for Stochastic Optimization** 3rd International Conference for Learning Representations, 2015.

A Appendix - Running Our Code

Our code is split into multiple files. The main code are in Jupyter notebooks, and can be run in a Jupyter environment. We also provide additional python files that will be imported into Jupyter environments.

The notebook *TensorflowModels.ipynb* contains the code for our tensorflow models, and some parts of it will read or write to the *Results* folder, which is a sub-directory from where this notebook is located. If for whatever reasons the notebook can not find the data, the data directories for this notebook are specified within the first few lines of code.

The notebook *evaluate.ipynb* contains the code to evaluate the PyTorch models (LeNet and ResNet18) with the transition matrices and robust loss functions. Instructions are provided in the notebooks for running the code. Utility functions for transition matrix estimation and defining the models are imported from *anchor_points.py, data_load.py, models.py*, and *tools.py*. The code to run the epoch and $alpha$ search for transition matrix estimation is in *transitionMatParams.ipynb*. Ensure that all files are in the same directory level.

B Appendix - Contributions

Alex and Kilian wrote the entire report between them, and share an equal contribution to the report writing.

James provided the code for the utility functions present in *anchor_points.py, data_load.py, models.py*, and *tools.py*, and a sample scaffold for performing transition matrix estimation.

Alex wrote the code to perform 10 fold repeated validation of the PyTorch models (*evaluate.ipynb*), and the transition matrix parameter estimation (*transitionMatParams.ipynb*). Kilian wrote the entirety of tensorflow models and their analysis code, along with creating some figures.