The Effect of Mislabeled Data on Neural Network Classification Accuracy

Connor Bramhall

Stephen Satalino

December 7th, 2018

Abstract

Machine learning systems use training data to find correlations in input data. Programmers usually build these systems on the assumption that the data is reliable. Those who cannot guarantee the validity of testing data must minimize the impact unreliable data has on their systems. In this study, I aimed to determine the effect of mislabeled training data on the classification accuracy of a neural network, and how the training method impacts that effect. I predicted the relationship between classification accuracy and noise data would be negatively logarithmic and become more pronounced with larger batch sizes. In this study, a controlled amount of training data was mislabeled and used to train a neural network with multiple methods of gradient descent. The networks made 55,000 updates to every weight before being tested on unaltered data. Higher batch sizes led to a closer match to the classification accuracy given with no mislabeled data when mislabeled data was between 10% and 50%. Higher batch sizes generally tended to yield more accurate networks when mislabeled data was used. R^2 values in linear regression were much higher for higher batch sizes, showing a near linear relationship. The a-values were much higher in quadratic regression for smaller batch sizes. Lower batch sizes are unfit to be used on systems with questionable data reliability. With this in mind, Programmers should seek to use the highest batch size possible when working with unreliable data.

Many modern companies owe their success to the power of machine learning. Superiority in these machine learning systems give companies a major advantage over their competitors. Companies recognise the value their self-learning machines bring, and have sparked a pseudo arms race for AI (artificial intelligence) technology in a continuous effort to strengthen this edge. A large contributor of the quality of a modern AI system is the data the system is trained on. This relationship has sparked the growth of "Big Data", collections of data too large for traditional data processing programs to handle. If you had more raw data than your competitor, you will theoretically create a better AI system. This generalization overlooks a variable in training, however: the reliability of the data. So much data is necessary to create a reliable system that there usually cannot be a human involved in the entire collection process. Consequently, the validity of the data cannot always be assured. This raises questions. How important is it for AI developers to assure that all their data is reliable? How does unreliable data affect a machine learning algorithm? Does the method by which a machine learning system is trained affect the effect of unreliable data? In my research, I hoped to answer these questions using a Back Propagation Neural Network, and seeing how modifying the descent method and accuracy of the data given would affect the classification accuracy of the resulting network.

Neural Networks are structures composing connections between layers of "neurons". To use the network, the beginning layer, the input layer, is given a set of data. Those values are then passed through multipliers called weights, and then used with the multiplied values from all the other neurons and an additional value called a bias to calculate, using an activation function, the value of the next layer of nodes. This process repeats layer by layer until the final layer is

reached. The final layer can represent a variety of things, but in the case of classification, the values are confidence levels in predictions.

Upon creation, neural networks are unintelligent. Neural networks learn by adjusting the weights so that its results better match that of the training data. A common method of adjusting these weights is called gradient descent. Gradient descent is an iterative algorithm that finds a relative minimum of a function. In machine learning, this algorithm is used on an n-dimensional function (where n is the number of connections + 1) to minimise one value: cost. Cost can be calculated in many ways, but in a general sense, the cost of a network is the error in classification for the tested training data. The absolute minimum of the function would represent the ideal settings of the weights to minimize cost. However, finding the absolute minimum is unfeasible, so gradient descent is used to find a relative minimum much more expediently.

The gradient descent process depends on the data used to calculate the cost value. If that training data is unreliable, the entire process is adversely affected, and the process is unlikely to result in an accurate network. Just one unreliable piece of data moves the minima of the cost function, so the algorithm would be optimising for different local minima than on the gradient formed from reliable data. The inclusion of these unreliable data points make the network account for edge cases that do not exist in the real world.

However, these false data points likely impact networks differently based on the method of gradient descent used. Batch, mini-batch and stochastic gradient descent are common methods used for machine learning (Kogan, 2018).

Batch gradient descent (BGD) takes every piece of data into account for every update of the network's weights. As a result, every update of the neural network's weights is costly in time

and computation power. This cost is exchanged for reliability, as the steps are much less erratic and there is a much smaller chance to lose progress. When unreliable data is used in a batch system, it is used in every update of the weights, so the unreliable data influences the entire training process. There is no opportunity to make an update with only reliable data even if only a single point is false.

Stochastic gradient descent (SGD) takes the opposite approach to BGD, using only a single data point to make each of its updates (Kogan, 2018). As such, SGD has nearly the opposite set of pros and cons. Updates with SGD take much less computing time. However, steps made with only single points tend to be very erratic because the gradients are unique to each piece of data, and can even lead to a loss in accuracy in some circumstances. These erratic steps are not strictly negative, however, because the shifting direction of the steps make it harder to get "stuck" in local minima (Kogan, 2018). SGD distributes the impact of false data, as each piece only affects the steps of the algorithm once per cycle of data. However, dedicating one whole step to an unreliable data point may counteract the bonuses the distribution gives.

The median between these two methods, mini-batch gradient descent (MBGD), has most of the benefits of the two previous methods while minimising their weaknesses. Neural networks using MBGD use multiple, but not all, data points for each update of weights. These portions are usually randomly selected and equally sized (Kogan, 2018), but my networks will group data in a predictable way for reproducibility. MBGD has a less erratic pattern compared to SGD, but still varies enough to overcome some local minima. It takes less computation time than BGD, but still has a lower chance to become less accurate with each step.

Though data modified from reality may seem like it would only adversely affect a network, modified data can actually be used to better train a neural network. Maaten, Chen, Tyree and Weinberger (2013) developed a method for purposefully corrupting training data called MCF and found "that MCF improves over standard predictors for both blankout corruption (for all corruption levels q) and Poisson corruption on five out of six tasks" and "MCF with Poisson corruption leads to significant performance improvements over standard classifiers" (p.5). One likely explanation for the improvement in performance is that corrupted examples help combat overfitting, the situation in which a Neural Network is over-optimized to accurately predict the training set, which hinders the network's ability to generalize and classify new data (Kogan, 2018). The variation provided by the corruption may have made overfitting impossible and therefore improved the network's accuracy. However, extreme corruption levels did adversely affect classification error in some document classification tests (Maaten et al., 2013, p. 5). Additionally, there is a difference between the modification of data between Maaten et al.'s tests and this study. Maaten et al.'s tests aim to use corrupted data, specifically attribute noise, to help the neural network improve its classification accuracy. The variable tested in this study would not be classified as attribute noise, but rather class noise, more specifically misclassification noise (Sáez, Galar, Luengo, & Herrera, 2013).

My experiment uses a common first-time neural network project: a network that can distinguish between handwritten numbers from the MNIST database. In this database, the numbers are 28px * 28px in grayscale. Every pixel is an input into the network, so 784 inputs are needed. The activation of each input is determined by the relevant pixel's brightness (Sanderson, 2017).

The two independent variables in my experiment are the descent method and the % of data mislabeled. In addition to learning mislabeled data's effect on classification accuracy, I wanted to learn how the type of descent affected the process, so I decided to change the descent methods. The dependent variable is the classification accuracy, with the percentages for the test being 0, 1, 5, 10, 20, 50 and 100. I chose these variables instead of a linear rate so if the largest effects occurred at lower mislabeling levels I would be able to observe it. Also, additional tests were run on SGD to further study an odd result after these first tests. The rest of the variables are constants. The network has 784 neurons in the input layer, 20 in the two hidden layers, and 10 in the output layer. The number and size of the hidden layers is mostly arbitrary, except being small enough to run efficiently and large enough to be somewhat accurate. At first, the network's hidden layers had 7 neurons, but that number was too low to get useful results. The starting weights are generated by a seed, so they are consistent. The data, except for the labels, are the same, and constants like learning rate and momentum are constant (and chosen arbitrarily), as well as number of steps per test, which was 55,000, the number of data points.

My experiment aimed to show to those building artificial intelligence systems how important reliability of data is in training, and if there was a critical point where unreliable data would overtake the training process. I also wanted to see how or if the type of training used affects how a network handles false data, so developers could take the information into account when building their systems. I hypothesized that the relationship between false data and classification accuracy would be negatively logarithmic, and if the type of gradient descent used was related to the effect error data had on accuracy, then SGD would resist the effects of mislabeled data the most of the three.

Methods and Materials

Because this experiment is mostly virtual, there are not any physical materials, except for my laptop. I used a Virtual Machine running Ubuntu to run my programs. My code is written in Python in PyCharm, and uses the MNIST dataset as well as tensorflow. For the specific code used in the trials, see the Appendix.

Cost was defined by mean squared error, defined as

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \widehat{y_i})^2$$

where N is the number of data points, y_i is the observed value for data point i and $\widehat{y_i}$ is the predicted value for data point i. The updates to each weight are using a momentum optimizer, where the changes to each weight can be defined as

$$\Delta w_t = \alpha \Delta w_{t-1} - \eta Q(w)$$

where Δw_t is the change in a weight, Δw_{t-1} is the weight's previous change, α is momentum (.5), η is learning rate (.01) and $Q(w)$ is the weight gradient.

For each test, the batch size for the type of gradient descent being tested was set. In my tests, MBGD had 100 data points to a batch unless stated otherwise, and BGD had 55,000 data points to a batch. Next, the corruption level was set. Then the code was run, the network (as described on page 7) was created, and the classification accuracy was returned by the program.

A total of 62 tests were run. Tests for BGD, MBGD for batch sizes 10, 100 and 1000, and SGD were all run with the 0%, 1%, 5%, 10%, 20%, 50% and 100% mislabeling levels with a seed of 24. Additional tests were performed with SGD at .5%, 2%, 15%, 17.5%, 30% and 75% mislabeling levels, also with a seed of 24. Redundancy tests were also run for BGD, MBGD for

batch size 100, and SGD with the 0%, 1%, 5%, 10%, 20%, 50% and 100% mislabeling levels, this time with a seed of 15.

For each of the five batch sizes tested, Linear regression tests were run to determine if the results fit a linear model. Graphs were created for all tests and compared, and multiple types of regression were also used to attempt to find a modelable correlation between factors like batch size, mislabeling percentage and classification accuracy.

Results

		Mislabeled data (as a % of the dataset)						
		0%	1%	5%	10%	20%	50%	100%
Batch Size	55000	100.00%	99.54%	98.09%	95.43%	88.43%	70.42%	24.86%
	1000	100.00%	99.63%	98.58%	95.33%	87.98%	67.60%	28.18%
	100	100.00%	99.50%	99.09%	95.33%	87.54%	62.06%	28.92%
	10	100.00%	99.60%	94.84%	84.57%	85.73%	29.93%	23.98%
	1	100.00%	98.57%	96.79%	73.94%	78.11%	27.08%	26.97%

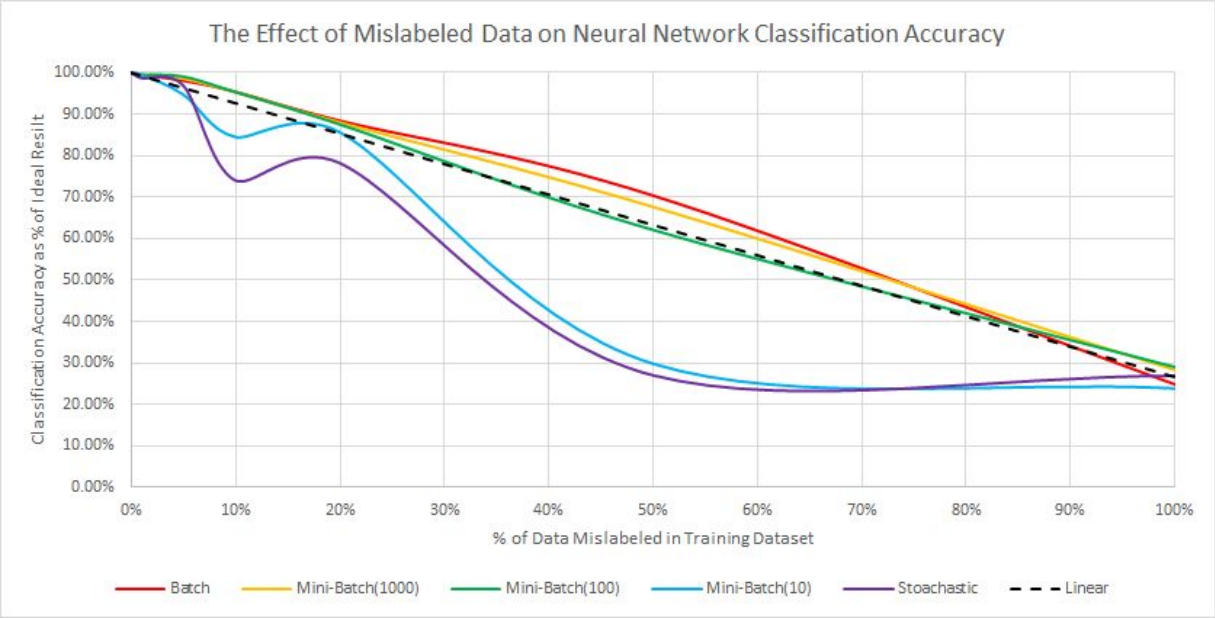


Figure 1A and 1B: Classification accuracy expressed as a % of "ideal conditions", or the classification accuracy returned with 0% mislabeling data. For raw data, see Appendix B.

x_5	y_5
0	100
0.5	99.45
1	98.57
2	97.53
5	96.79
10	73.94
15	66.22
17.5	78.11
20	78.11
30	29.11
50	27.08
75	26.97
100	26.97

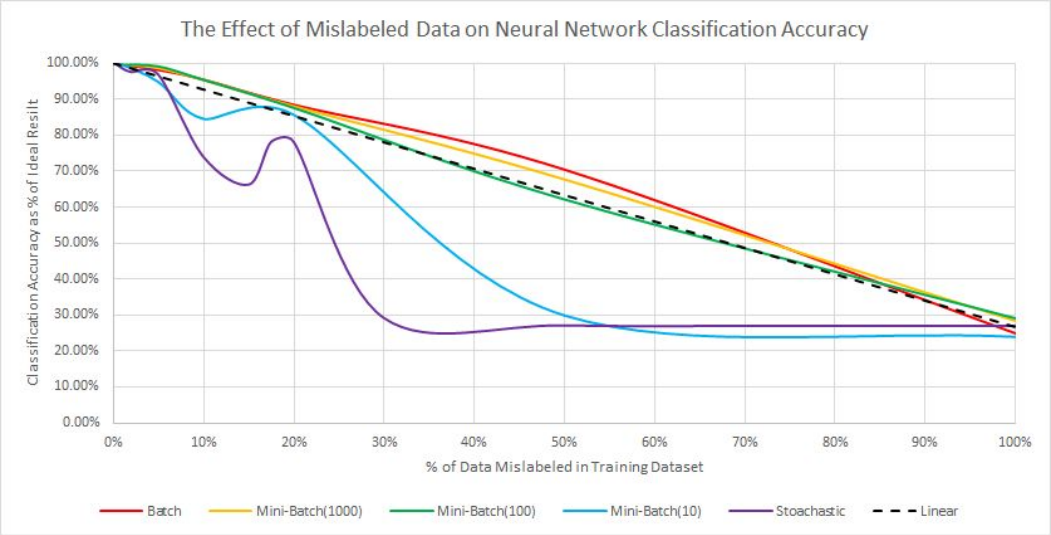


Figure 2A (left): Extra SGD Classification accuracy data expressed as a % of ideal conditions.

Figure 2B (right): Same as Figure 1B, but includes the extra SGD data.

		Mislabeled data (as a % of the dataset)						
		0%	1%	5%	10%	20%	50%	100%
Batch Size	55000	100.00%	99.65%	97.21%	94.12%	85.45%	59.78%	20.32%
	100	100.00%	99.75%	97.20%	92.33%	83.19%	58.41%	22.19%
	1	100.00%	97.41%	92.45%	64.61%	72.16%	27.30%	26.49%

Figure 3: Redundancy testing results expressed as a % of ideal conditions (of this 15 seeded trial, not the 24 seeded trial. The 24 seeded trials have different ideal conditions).

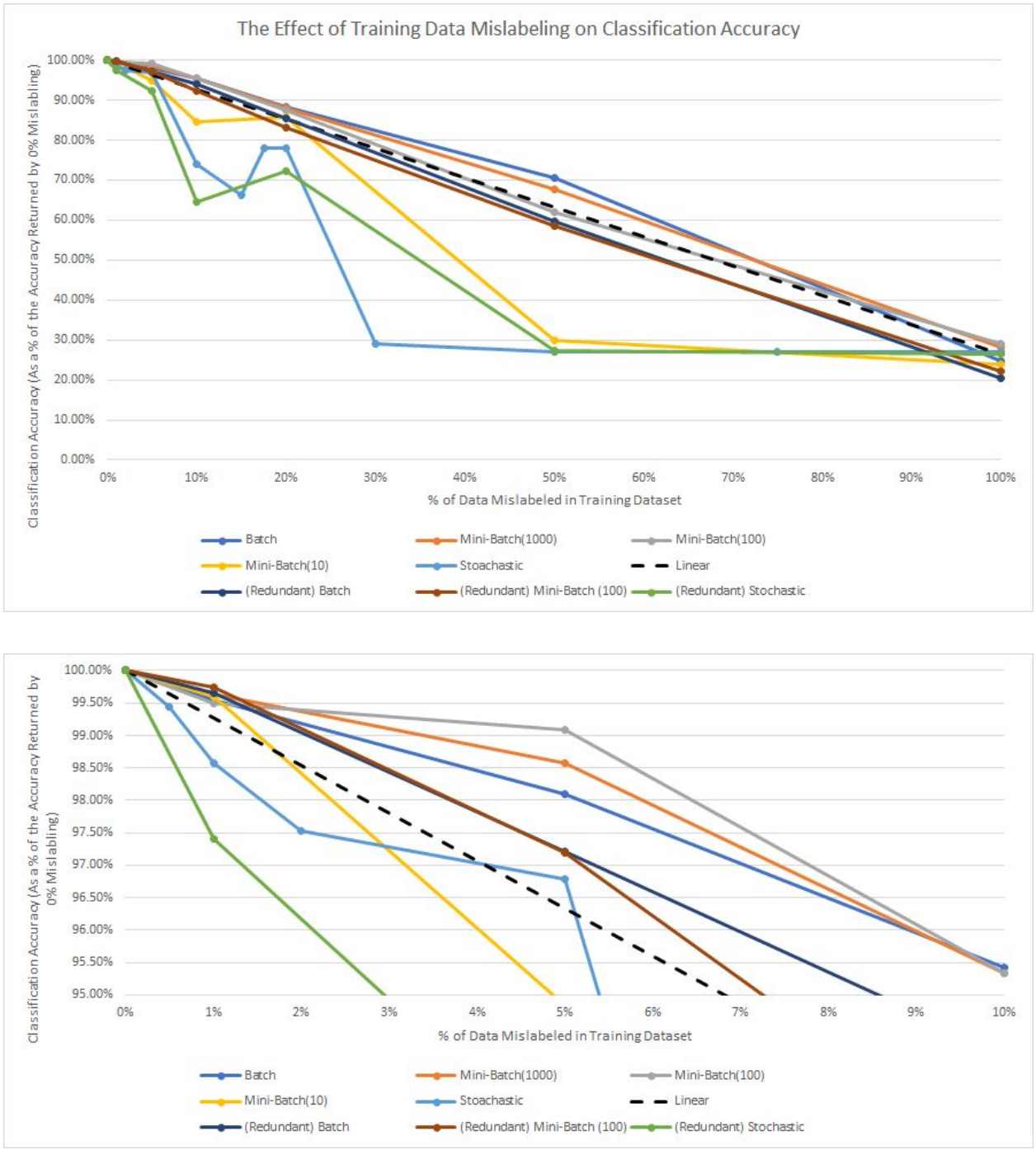
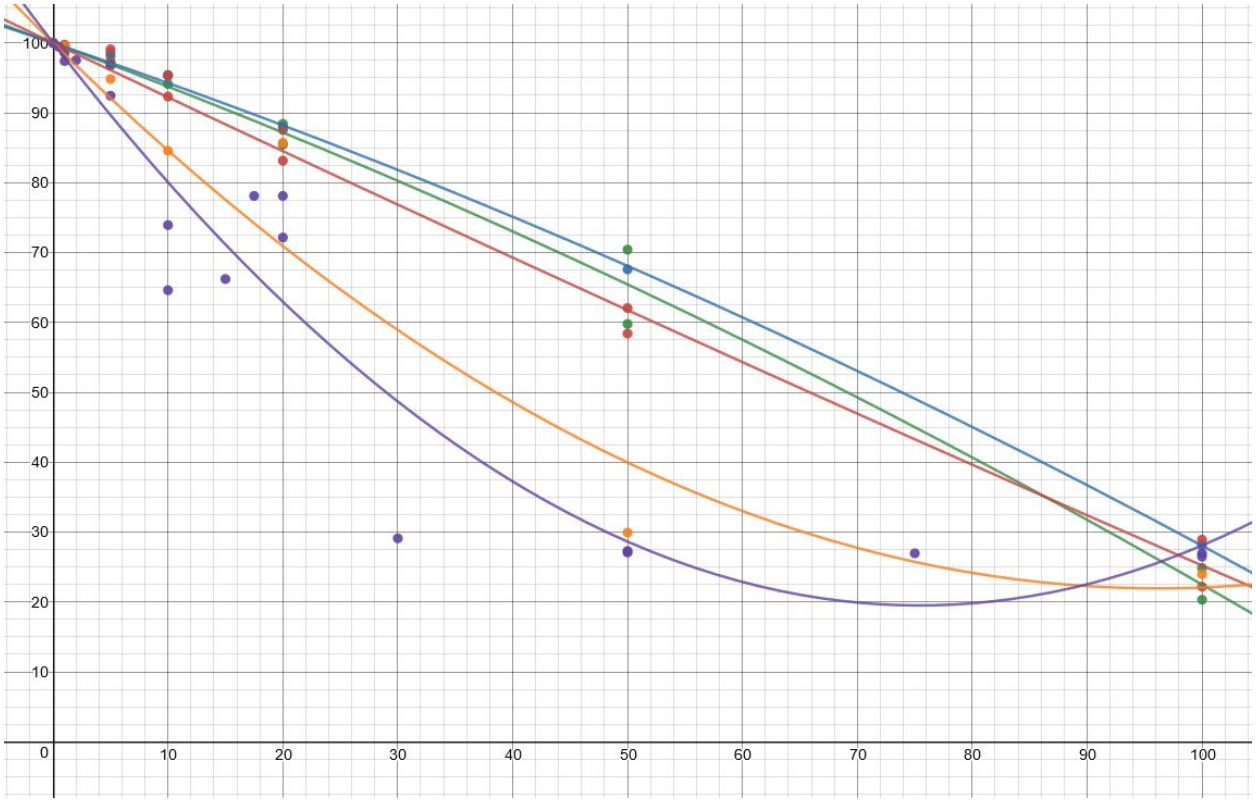


Figure 4A and 4B: All collected data plotted on a linked scatter graph expressed as a % of ideal conditions (of their relative seed and batch size).



	a-value	b-value	R^2 value
Batch	-.00167985	-.606974	.9919
Mini-batch (1000)	-.00162913	-.0556497	.9992
Mini-batch (100)	.000340168	-.781736	.9927
Mini-batch(10)	.0084315	-1.62243	.9485
Stochastic	.0141606	-2.13517	.9235

Figure 5A and 5B: Quadratic regression on all data.

Purple - SGD, Orange - MBGD(10), Red - MBGD(100), Blue - MBGD(1000), Green - BGD

	R value	R^2 value	Significance F (P value)
Batch	.994907	.989841	$2.57*10^{-12}$
Mini-batch (1000)	.998589	.99718	$1.44*10^{-7}$

Mini-batch (100)	.996463	.992938	$3.48*10^{-13}$
Mini-batch(10)	.934511	.873311	$2.03*10^{-3}$
Stochastic	.866228	.75035	$1.63*10^{-6}$

Figure 6: Linear regression important figures.

Discussion

Figure 5 shows the main finding of the experiments. Higher batch sizes show more resilience to mislabeled data. Figure 6 supports this claim, as the higher R^2 values from linear regression show that higher batch sizes have more linear correlations with the mislabeling percentage. Batch, however, has a slightly lower R^2 value than Mini-batch size 1000 and 100, but in Figure 1 and Figure 5 the graph is shown to rise slightly above a linear correlation, so it does not refute my conclusion. In addition, quadratic regression on the data returns the highest a-value for SGD. Because of this, the predictor function has the most pronounced curve, suggesting that SGD's results are modeled the least accurately with a linear function. The P-values gathered from the linear regression tests also suggest that the results are significant, but that is expected, as you would expect the neural network to get less accurate with more mislabeled data and that the accuracy would not fluctuate.

There is one outlier to this previous expectation, however. Stochastic and, to a lesser effect, size 10 mini-batch, have an odd "bump" in classification accuracy between 10% and 30% mislabeling. SGD and MBGD remain less accurate than the other methods even with the accuracy gain because of the bump, however. Figure 4A shows the effect is reproducible. Despite this reproducibility, This phenomenon can likely be explained by randomness and the Law of Large Numbers, which states that as the number of identically distributed, randomly

generated variables increases, their sample mean approaches their theoretical mean (Routledge, 2016). The small sample sizes causes the gradient to change wildly between steps, resulting in a huge variability between trials. The bumps are likely just the result of this variation. If given enough trials with varying seeds or starting data points, the accuracies would likely trend around a "true" relationship, as the Law of Large Numbers says. The larger batch sizes likely do not show this property because the sample (batch) size for each trial keeps the results much closer to the "true" trend line. In order to know for certain whether my results show a trend due to chance or have some previously unknown correlation that cannot be explained by the Law of Large Numbers, more testing of the lower batch sizes would be needed.

My results for the larger batch sizes seemed to correlate with the trends of less complex machine learning algorithms. Brodely and Freidl (1999) conducted research on a possible method for removing noise data. In their tests, they trained classifiers like a 1 - Nearest Neighbor algorithm, a Linear Machine and a Decision Tree. They used control groups with no filter that would be trained on data with varying amounts of noise data. From 0% to 40% noise, the accuracy drops between 10 and 20 percent, varying on the classifier. In addition, all three algorithms' classification accuracy seemed to have a linear correlation with the noise level, with the slope varying on the classifier used (p. 147 - 148).

The mostly linear relationship between noise data and accuracy is maintained at higher batch sizes. For the relationship between the gradient descent method, my original hypothesis was proven false. I had believed that SGD would be affected the least by mislabeled data, but the inverse was true. BGD seems the most promising for developers who are concerned with the validity of their data without an efficient way of validating the data.

New research supports this conclusion. Rolnich, Viet, Belongie and Shavit (2018) tested a similar network to the network tested in this study, and found, similarly to this study, that higher batch sizes are more resilient to label noise. They propose a "possible explanation for... [the greater resilience of higher batch sizes] ...is that within a batch, gradient updates from randomly sampled noisy labels roughly cancel out, while gradients from correct examples that are marginally more frequent sum together and contribute to learning. By this logic, large batch sizes are more robust to noise since the mean gradient over a larger batch is closer to the gradient for correct labels" (p.7).

Neural networks are complex and delicate digital structures. The ambiguous nature at a fundamental level makes it hard to debug and keep good track of. There were many opportunities for my program to be flawed, and I may not even have the ability to know that a flaw existed at all. I did what I could at a high level to make sure the the network ran as expected. However there were still opportunities for errors to occur at low levels or in areas of tensorflow I do not know very well. I encourage further research into this topic to corroborate my findings. The results make logical sense however, and as a result I feel confident in my methods, but I acknowledge that there are many factors that could have influenced my data.

I believe widening the scope of my project would be the best way to improve my research. Testing more advanced neural networks would determine whether the results pointed out in this paper are universal or case-by-case, and would lead to a more definite conclusion regarding SGD's jump in accuracy. I would also suggest to those with the resources perform a similar test with a larger network with more steps. None of the raw classification accuracies broke 45% (see Appendix B), so the changes in accuracy may have a different effect on a larger

and better trained network that had more of an opportunity to reach a minima. In addition, in my research I have assumed that the results given by tensorflow would be the similar when tested with a different base or coded from scratch. Testing the same methods on a different framework like PyTorch would add more credibility to my results.

Conclusion

I began this study with the hope of discovering useful data that developers could take into consideration when assembling a neural network with data that they had little control over the reliability of, like user-generated data or "big" data. The findings of my study indicate that any network built with concerns of data reliability should aim for higher batch sizes, as higher batch sizes have nearly linear relationships with false data, which are preferable to the steep drops of the lower batch sizes' relationships.

My original hypotheses were not supported. A majority of the relationships between mislabeled data and classification accuracy seemed linear. Those that did not, the smaller batch sizes, seemed to have a plausibility of being negatively logarithmic, but the accuracy is too variable to support any assertion. In addition, SGD was not the least impacted by mislabeled data, countering my hypothesis. Even though both of my hypotheses were rejected, my data still shows useful correlations that can help developers further decide on the methods they use to train their networks.

References

Brodley, C. E., & Freidl, M. A. (1999). Identifying Mislabeled Training Data. Journal of Artificial Intelligence Research II, 11, 131-167. doi:https://doi.org/10.1613/jair.606

Kogan, G. (2018, January 24). How neural networks are trained. Retrieved November 8, 2018, from https://ml4a.github.io/ml4a/how_neural_networks_are_trained/

Maaten, L. V., Chen, M., Tyree, S., & Weinberger, K. Q. (2013). Learning with Marginalized Corrupted Features. International Conference on Machine Learning, 28. Retrieved from http://proceedings.mlr.press/v28/vandermaaten13.pdf

Rolnich, D., Veit, A., Belongie, S., & Shavit, N. (2018). Deep Learning is Robust to Massive Label Noise. Retrieved November 13, 2018, from https://arxiv.org/pdf/1705.10694.pdf.

Routledge, R. (2016, October 12). Law of large numbers. Retrieved December 6, 2018, from https://www.britannica.com/science/law-of-large-numbers

Sáez, J. A., Galar, M., Luengo, J., & Herrera, F. (2013). Tackling the problem of classification with noisy data using Multiple Classifier Systems: Analysis of the performance and robustness. *Information Sciences, 247*, 1-20. doi:10.1016/j.ins.2013.06.002

Sanderson, G. (Writer). (2017, October 7). But what *is* a Neural Network? | Chapter 1, deep learning [Video file]. Retrieved May 11, 2018, from https://www.youtube.com/watch?v=aircAruvnKk

Acknowledgements

I would like to thank my scientific research teacher Mr. Shuder for helping me narrow down my topic and giving advice for how I should structure my project. I thank my mentor, Mr. Satalino, for giving advice on my project, keeping me on track, and helping clean up my research notebook and data. I thank Mr. Clark for helping me understand data analysis figures and suggesting the linear regression analysis I used in my project. I would like to thank my parents

for the support they gave throughout this project. Finally, I would like to thank the youtube user sentdex for the tutorials that made learning both python and tensorflow much easier.

Appendix A

Code Used in Experimentation

The following is a screenshot of the code used for the tests.

```
# Plenty of help from sentdex on YouTube.
import ...
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
n_nodes_hl1 = 20
n_nodes_hl2 = 20
classes = 10
batch_size = len(mnist.train.images)
x = tf.placeholder('float', [None, 784])  # input
y = tf.placeholder('float')
corruption_level = 1
def neural_network_model(data):
    tf.set_random_seed(24)
    hidden_1_layer = {'weights': tf.Variable(tf.random_normal([784, n_nodes_hl1])),
                      'biases': tf.Variable(tf.random_normal([n_nodes_hl1]))}
    hidden_2_layer = {'weights': tf.Variable(tf.random_normal([n_nodes_hl1, n_nodes_hl2])),
                      'biases': tf.Variable(tf.random_normal([n_nodes_hl2]))}
    output_l = {'weights': tf.Variable(tf.random_normal([n_nodes_hl2, classes])),
                'biases': tf.Variable(tf.random_normal([classes]))}
    l1 = tf.add(tf.matmul(data, hidden_1_layer['weights']), hidden_1_layer['biases'])
    l1 = tf.nn.sigmoid(l1)
    l2 = tf.add(tf.matmul(l1, hidden_2_layer['weights']), hidden_2_layer['biases'])
    l2 = tf.nn.sigmoid(l2)
    output = (tf.matmul(l2, output_l['weights']) + output_l['biases'])
    return output
def training(x):
    prediction = neural_network_model(x)
    cost = tf.losses.mean_squared_error(predictions=prediction, labels=y )
    optimizer = tf.train.MomentumOptimizer(learning_rate=.01, momentum=0.5).minimize(cost)
    print("beginning training!")
    starttime = time.time()
    d_epochs = 55000
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        #print(mnist.train.labels[43][8])
        to_corrupt = corruption_level * len(mnist.train.labels)
        if to_corrupt != 0:
            every_x = int(len(mnist.train.labels) / to_corrupt)
            ii = every_x
            current_dis = 0
            while ii < len(mnist.train.labels):
                to_null = 10
                iii = 0
                while iii < 10:
                    if mnist.train.labels[ii][iii] == 1.0:
                        to_null = iii
                    iii += 1
                if to_null == current_dis :
                    current_dis += 1
                if current_dis == 10 :
                    current_dis=0
                mnist.train.labels[ii][to_null] = 0.0
                mnist.train.labels[ii][current_dis] = 1.0
                ii += every_x
```

```
        '''...'''
        s = 0
        cc = 0
        while s < d_epochs:
            epoch_l = 0
            '''...'''
            i = 0
            while i < len(mnist.train.images):
                start = i
                end = i+batch_size
                if end >= len(mnist.train.images):
                    end = len(mnist.train.images)
                epoch_x = mnist.train.images[start:end]
                epoch_y = mnist.train.labels[start:end]
                _, c = sess.run([optimizer, cost], feed_dict={x: epoch_x, y: epoch_y})
                epoch_l += c
                i += batch_size
                s += 1
                print_thing = 'Step ' + repr(s) + '/' + repr(d_epochs) + ' completed. loss:' + repr(epoch_l) + ' '
                print(print_thing)
            cc+=1
        print(cc)
        endtime = time.time()
        time_elapsed= endtime - starttime
        print(time_elapsed)
        correct = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))
        acc = tf.reduce_mean(tf.cast(correct, 'float'))
        print('accuracy:', acc.eval({x: mnist.test.images, y: mnist.test.labels}))
training(x)
```

Appendix B

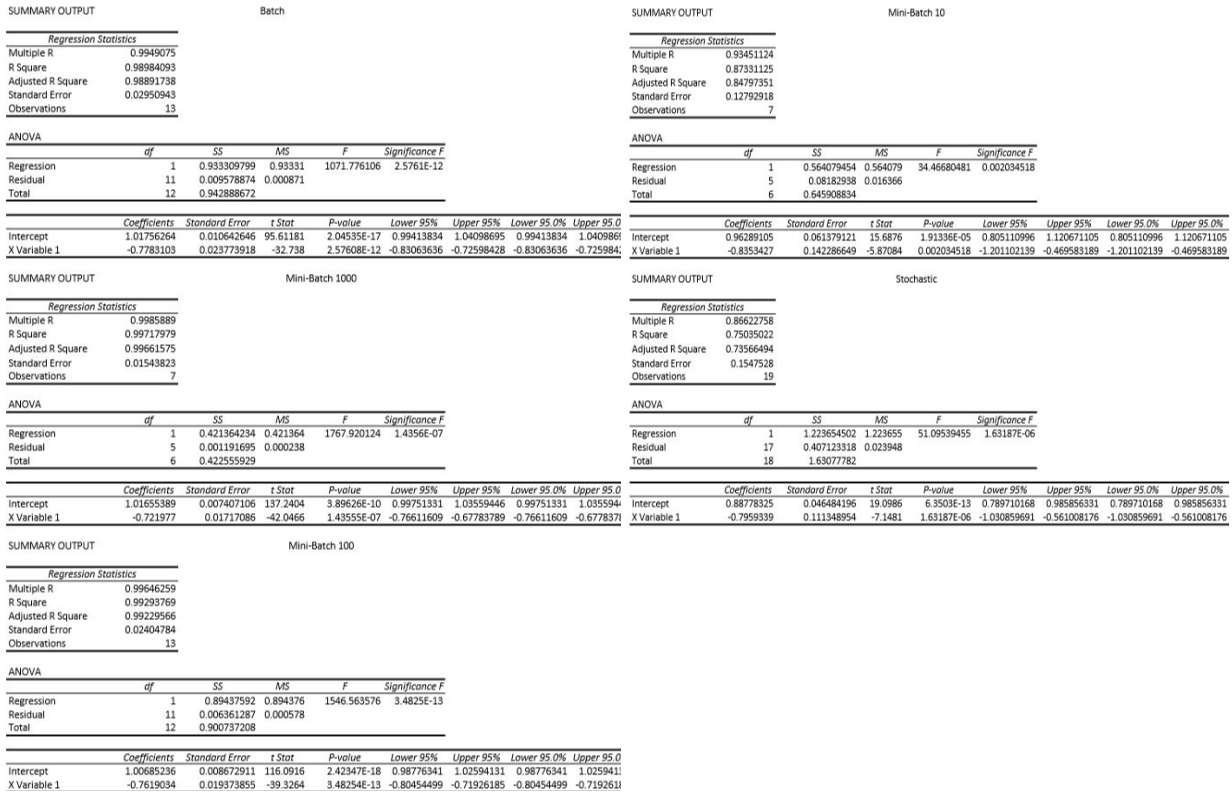
Raw Accuracy and Linear Regression Data

		Mislabeled data (as a % of the dataset)						
		0%	1%	5%	10%	20%	50%	100%
Batch Size	55000	37.25%	37.08%	36.54%	35.55%	32.94%	26.23%	9.26%
	55000 red.	39.46%	39.32%	38.36%	37.14%	33.72%	23.59%	8.02%
	1000	37.44%	37.30%	36.91%	35.69%	32.94%	25.31%	10.55%
	100	38.35%	38.16%	38.00%	36.56%	33.57%	23.80%	11.09%
	100 red.	39.26%	39.16%	38.16%	36.25%	32.65%	22.93%	8.71%
	10	40.50%	40.34%	38.41%	34.25%	34.72%	12.12%	9.71%
	1	36.41%	35.89%	35.24%	26.92%	28.44%	9.86%	9.82%
	1 red.	37.07%	36.11%	34.27%	23.95%	26.72%	10.12%	9.82%

	Mislabeled data (as a % of the dataset)					
	0.50%	2%	15%	17.5%	30%	75%

1 (cont.)	99.45%	97.53%	66.22%	78.11%	29.11%	26.97%

Figure A1: All collected data expressed as raw classification accuracy



SUMMARY OUTPUT — Batch

Regression Statistics	
Multiple R	0.9949075
R Square	0.98984093
Adjusted R Square	0.98891738
Standard Error	0.02950943
Observations	13

ANOVA

	df	SS	MS	F	Significance F
Regression	1	0.933309799	0.93331	1071.776106	2.5761E-12
Residual	11	0.009578874	0.000871		
Total	12	0.942888672			

	Coefficients	Standard Error	t Stat	P-value	Lower 95%	Upper 95%	Lower 95.0%	Upper 95.0
Intercept	1.01756264	0.010642646	95.61181	2.04535E-17	0.99413834	1.04098695	0.99413834	1.040986
X Variable 1	-0.7783103	0.023773918	-32.738	2.57608E-12	-0.83063636	-0.72598428	-0.83063636	-0.725984

SUMMARY OUTPUT — Mini-Batch 10

Regression Statistics	
Multiple R	0.93451124
R Square	0.87331125
Adjusted R Square	0.84797351
Standard Error	0.12792918
Observations	7

ANOVA

	df	SS	MS	F	Significance F
Regression	1	0.564079454	0.564079	34.46680481	0.002034518
Residual	5	0.08182938	0.016366		
Total	6	0.645908834			

	Coefficients	Standard Error	t Stat	P-value	Lower 95%	Upper 95%	Lower 95.0%	Upper 95.0%
Intercept	0.96289105	0.061379121	15.6876	1.91336E-05	0.805110996	1.120671105	0.805110996	1.120671105
X Variable 1	-0.8353427	0.142286649	-5.87084	0.002034518	-1.201102139	-0.469583189	-1.201102139	-0.469583189

SUMMARY OUTPUT — Mini-Batch 1000

Regression Statistics	
Multiple R	0.9985889
R Square	0.99717979
Adjusted R Square	0.99661575
Standard Error	0.01543823
Observations	7

ANOVA

	df	SS	MS	F	Significance F
Regression	1	0.421364234	0.421364	1767.920124	1.4356E-07
Residual	5	0.001191695	0.000238		
Total	6	0.422555929			

	Coefficients	Standard Error	t Stat	P-value	Lower 95%	Upper 95%	Lower 95.0%	Upper 95.0
Intercept	1.01655389	0.007407106	137.2404	3.89626E-10	0.99751331	1.03559446	0.99751331	1.035594
X Variable 1	-0.721977	0.01717086	-42.0466	1.43555E-07	-0.76611609	-0.67783789	-0.76611609	-0.677837

SUMMARY OUTPUT — Stochastic

Regression Statistics	
Multiple R	0.86622758
R Square	0.75035022
Adjusted R Square	0.73566494
Standard Error	0.1547528
Observations	19

ANOVA

	df	SS	MS	F	Significance F
Regression	1	1.223654502	1.223655	51.09539455	1.63187E-06
Residual	17	0.407123318	0.023948		
Total	18	1.63077782			

	Coefficients	Standard Error	t Stat	P-value	Lower 95%	Upper 95%	Lower 95.0%	Upper 95.0%
Intercept	0.88778325	0.046484196	19.0986	6.3503E-13	0.789710168	0.985856331	0.789710168	0.985856331
X Variable 1	-0.7959339	0.111348954	-7.1481	1.63187E-06	-1.030859691	-0.561008176	-1.030859691	-0.561008176

SUMMARY OUTPUT — Mini-Batch 100

Regression Statistics	
Multiple R	0.99646259
R Square	0.99293769
Adjusted R Square	0.99229566
Standard Error	0.02404784
Observations	13

ANOVA

	df	SS	MS	F	Significance F
Regression	1	0.89437592	0.894376	1546.563576	3.4825E-13
Residual	11	0.006361287	0.000578		
Total	12	0.900737208			

	Coefficients	Standard Error	t Stat	P-value	Lower 95%	Upper 95%	Lower 95.0%	Upper 95.0
Intercept	1.00685236	0.008672911	116.0916	2.42347E-18	0.98776341	1.02594131	0.98776341	1.025941
X Variable 1	-0.7619034	0.019373855	-39.3264	3.48254E-13	-0.80454499	-0.71926185	-0.80454499	-0.719261

Figure A2 : Raw data from linear regression analysis.

Appendix C

Additional Regression

I attempted to make an equation that predicted the effect any given noise level would have on any given batch size. The following is a result using all data where b is the batch size, x is the mislabeling % and y is classification accuracy in % of ideal.

$$y_1 \sim \left(.0184828b_1^{-0.258896} - .00376774\right)x_1^2 + \left(\frac{-.000001524636b_1^2 - .522616b_1 - 40.4593}{b_1 + 18.1876}\right)x_1 + 100$$

STATISTICS

$R^2 = 0.9368$

RESIDUALS

e_1