Supplementary Material

A Overview of Popular Regularisation Methods

First we give a brief overview of the regularisation methods tested. The main goal here is to compare a wide variety of different schemes to see if they have any common behaviour.

A.1 Weight decay

One of the most used methods for regularisation is placing a ℓ_2-penalty on the parameters of the network. Colloquially this is known as *Weight Decay* [Krogh and Hertz, 1991]. It is a well understood method from classical regression models, where estimation variance is reduced by regularising large weights or regression coefficients. Despite its easy interpretation for these classical models, its application to neural networks is not completely understood.

A.2 Dropout

Dropout [Srivastava et al., 2014] is another commonly used regularisation method that works by overlaying noise during the learning procedure. The output from each neuron in the network is discarded at random during training according to a set distribution. Intuitively, by adding this noise, the network is forced to not rely on any given path through the network which could be used to memorise data.

A.3 Learning rate algorithms and optimisers

There exist many optimisers other than standard SGD that also offer regularisation benefits. Optimisers such as Adam [Kingma and Ba, 2015] set an adaptive learning rate by using estimations of first and second order terms. On the other simple but effective learning rate algorithms such as *Cyclical Learning Rate* [Li and Yang, 2020] simply adjust the learning rate in a more nuanced way that leads to a better learning procedure without using any past training information.

A.4 Loss gradient norm penalties

Loss gradient norm penalties add an additional penalty term to the loss: the norm of the gradient of the loss itself. This will lead the gradients of the gradients being calculated in the final back-propagation step, hence why it is commonly referred to as double back-propagation. First, a back-propagation pass must be done to obtain the gradients, then a second pass to evaluate the weight updates with the gradient norm included. Two versions exist: one which calculates the gradient of the loss with respect to the input data [Drucker and Le Cun, 1991, Hoffman et al., 2019] and another that instead does so with respect to the parameters [Barrett and Dherin, 2021, Zhao et al., 2022]:

$$\tilde{\mathcal{L}} = \mathcal{L} + ||\nabla_x \mathcal{L}||$$
$$\tilde{\mathcal{L}} = \mathcal{L} + ||\nabla_\theta \mathcal{L}||.$$

We note one important detail in the relation of the latter, the loss-parameter gradient norm, to the MGS trace penalty:

$$||\nabla_\theta \mathcal{L}|| = ||\nabla_\theta f \cdot \nabla_f \mathcal{L}|| \leq ||\nabla_\theta f|| \, ||\nabla_f \mathcal{L}||.$$

Therefore, $||\nabla_\theta L||$ is proportional to the MGS trace penalty as $||\nabla_\theta f||_2^2 = \sqrt{\operatorname{tr} K_\theta}$. However depending on how the second term $||\nabla_f \mathcal{L}||$ behaves the two might not be coupled at all. This might explain why in some instances in the experiments, the loss-parameter gradient penalty performed significantly worse. However, it is still the closest to directly optimising MGS and when the loss-model gradient is well behaved, can perform just as well as MGS. Still, the issue arises as to how this penalty performs when any minima is approached, as the second term will diminish, causing the whole penalty to lose influence over training. This might then lead to the model starting to overfit after landing in a minima, which we have observed in our experiments.

A.5 Functional regularisation

Recently it has also been shown that *Kernel Ridge Regression* can be applied to neural networks. This places a penalty on function complexity by trying to minimise the norm of the function given by its *Reproducing Kernel Hilbert Space* (RKHS) (e.g. [Bietti et al., 2019]). For neural networks, the NTK can be used to define such a RKHS, however the actual function realised by the network is not necessarily close to its RKHS counterpart. Additionally, in our experience, these penalties are the most costly to calculate and were not efficient enough even to run on small networks.

B Connection to Neural Tangent Theory

It should most certainly be noted that the MGS kernel is indeed the same as the *empirical Neural Tangent Kernel* $\hat{\Theta}$ which is derived from the *Neural Tangent Kernel* (NTK) itself. From the NTK literature, the NTK describes the evolution of the network function f_θ in function space in a kernel gradient descent setting. Therefore we can also make use of findings that have been made from the NTK perspective.

A observation made by about the empirical NTK is the notion of "feature learning speed" [Jacot et al., 2018, Baratin et al., 2021, Ronen et al., 2019]. Neural networks seem to learn functions/features of increasing complexity starting with low frequency content first and then sequentially higher frequency information as training progresses. The spectral decomposition of the empirical NTK describes the primary directions of learning with large eigenvalues corresponding to learning directions with low complexity which is also where convergence is the fastest. While most of the ideas from the NTK side are formulated by viewing the network as learning in function space, it can be useful to see what practical implications this has on the model gradients and therefore MGS.

Due to the NTK growing in popularity there exist a number of performant libraries for calculating the empirical NTK. We make use of *Neural Tangents* [Novak et al., 2019] and *Fast Finite Width NTK* [Novak et al., 2022] libraries which are built on *JAX* [Bradbury et al. [2018]].

C Using MGS metrics for adversarial sample detection

The authors Martin and Elster [2020] used $\operatorname{tr} K_\theta$ for detection of adversarial samples when running a neural network. They do this by calculating the trace of what they call the *approximate Fisher Information Matrix* (FIM) which is the actually same as K_θ in our case. They observed that if adversarial samples were present in the input, then the trace of the approximate FIM would be large. Their reasoning behind this behaviour, however, was deduced from the properties of the complete FIM. From the gradient similarity perspective it also makes sense. Adversarial samples are intended to look very similar to trained samples, but corrupt the model by causing it to produce different outputs. Therefore, if for adversarial data the trace is large, then the network considers that data to be very different from existing data. This also makes sense as that is one of the objectives of adversarial attacks in the first place.

D Experiments

All experiments using a FCN were run locally on a laptop with a Nvidia Quadro T2000 graphics card. For convolutional networks, the laptop graphics card was not sufficient so instead a single Nvidia K80 was used on another machine.

D.1 Experiment details

Classification: MNIST A corrupted version (motion blur) of MNIST was used from the MNIST-C dataset [Mu and Gilmer, 2019]. Each method was tuned using a simple grid search over a set of parameters. The scenario in which the tuning took place was using 6000 randomly stratified sampled training points and 50% of the target labels were randomly flipped. Each method was trained for 100 epochs and also run 5 times over, with differently randomly sample data each time. The final test loss was used to determine the optimal parameter. The architectures tested included a standard fully connected architecture with 6 hidden layers, 300 neurons wide, and ReLU activations as well as a LeNet-5 style convolutional network. Cross-entropy with softmax was used as the loss function. In the case of Dropout, a Dropout layer was added between all fully connected layers in both architectures as well as a Batchnorm layer between convolutional layers.

After tuning, each method retained the same parameters and was then run over a large testbench, for 400 epochs for both networks, where the training size and label noise were varied between 250 and 9000 training samples and 0% to 100% respectively. The test accuracy was calculated on the predetermined test sample found in MNIST and not the data that was left over from the training sample. Each scenario was run 10 times, again with new network initialisations and sampled training data. A slow exponential decay was used for the learning rate starting at 0.1 and a batch size of 32 was used.

Each of the metrics, including test accuracy, were sampled at 100 evenly spaced points during training, so as to not run too slowly. The final test accuracy was calculated as an average of the previous 5 metric samples and then averaged between the training runs. The max value was calculated as the max of the averaged runs.

Regression: Facial Keypoints A similarly corrupted version (motion blur) of the Facial Keypoints dataset [1] was used. The same architecture types were used as in the MNIST experiment. Standard MSE loss is used. Unlike the MNIST experiment, a grid of scenarios was not run for this problem. Instead the training size was held fixed at 30% of the total data, while the amount of noise was changed. Gaussian noise was added to the target coordinates with a scale indicated by the "noise" column in the tables. Each method was tuned with a noise scale of 15 for 50 epochs.

Each scenario was then run for 400 epochs in the case of the FCN architecture and 50 epochs for the LeNet-5 architecture. This was due to the larger size of the input and output, causing the convolutional network to run much slower. A batch size of 128 was used for both architectures in an attempt to lessen the training wall-clock time. This could explain why MGS in some cases achieved slightly better performance for the FCN architecture over the LeNet-5 architecture as it was allowed to train for longer.

The metrics were tracked in the same way as the MNIST experiment, however test MSE was used instead of test accuracy.

Training parameter robustness To test robustness with regards to different training parameters, a benchmark was run where each method is first tuned using the same scenario. Then, the tuned scenario is used as a starting point from which each of the 5 training parameters were changed independently. Both larger and smaller parameter values were selected to test as broad of a spectrum as possible. For each parameter changed, multiple runs were performed, again by resampling the data and using a different initialisation. The final performance results were aggregated, with the quantiles being drawn, based on all the results. The test was run using the same classification problem based on the corrupted MNIST dataset described previously. Specifically, each method was tuned using 3000 training samples, 50% label noise, and for 100 epochs with all the other settings such as learning rate the same as per section D.1.

D.2 Results for MNIST and Facial Keypoints for a FCN network

---

[1] https://www.kaggle.com/competitions/facial-keypoints-detection/

Table A.1: **Test accuracy for corrupted MNIST dataset using a FCN architecture.** Noise column represents percentage of training labels that have been randomly flipped. MGS is able to handle large amounts of noise. If the max accuracy is compared to the final accuracy, MGS is also the most consistent and is not susceptible to overfitting. The FCC results are worse than those of LeNet-5.

Noise	Unregularised	Dropout	Weight	Loss grad.	MGS
0%	82.1 ±1.9 (82.3)	79.5 ±1.2 (80.0)	72.8 ±9.7 (79.6)	74.9 ±22.0 (79.8)	**84.6** ±1.2 (84.8)
30%	59.5 ±3.7 (75.3)	73.7 ±2.5 (75.5)	58.7 ±4.0 (74.6)	70.3 ±4.3 (77.4)	**77.7** ±1.8 (79.4)
60%	36.9 ±2.8 (65.6)	55.6 ±4.8 (57.1)	10.0 ±0.5 (63.1)	41.2 ±6.4 (43.4)	**67.8** ±3.4 (70.0)
80%	24.8 ±2.7 (53.0)	26.5 ±3.7 (29.6)	10.1 ±0.6 (41.3)	12.1 ±2.5 (14.8)	**51.4** ±3.8 (55.0)

Table A.2: **Test loss for the corrupted Facial Keypoints dataset using a FCN architecture.** MGS is the only regulariser that is able to attain convergence with good performance.

Noise	Unregularised	Dropout	Weight	Loss grad.	MGS
0	281.1 ±264.6 (181.5)	208.2 ±220.9 (74.3)	276.4 ±262.3 (180.2)	337.9 ±303.8 (184.7)	**10.6** ±0.3 (10.5)
10	305.7 ±344.7 (176.8)	240.2 ±291.1 (191.7)	305.2 ±342.2 (175.0)	335.3 ±332.1 (208.2)	**11.0** ±0.5 (10.7)
20	325.2 ±371.7 (158.6)	287.1 ±321.9 (88.0)	328.5 ±379.4 (157.7)	353.1 ±382.7 (174.2)	**13.6** ±1.3 (12.8)
30	390.8 ±453.2 (162.6)	210.4 ±270.9 (91.7)	393.0 ±454.2 (162.2)	411.8 ±442.8 (175.6)	**22.4** ±2.7 (20.9)

E Two circles problem decision boundaries

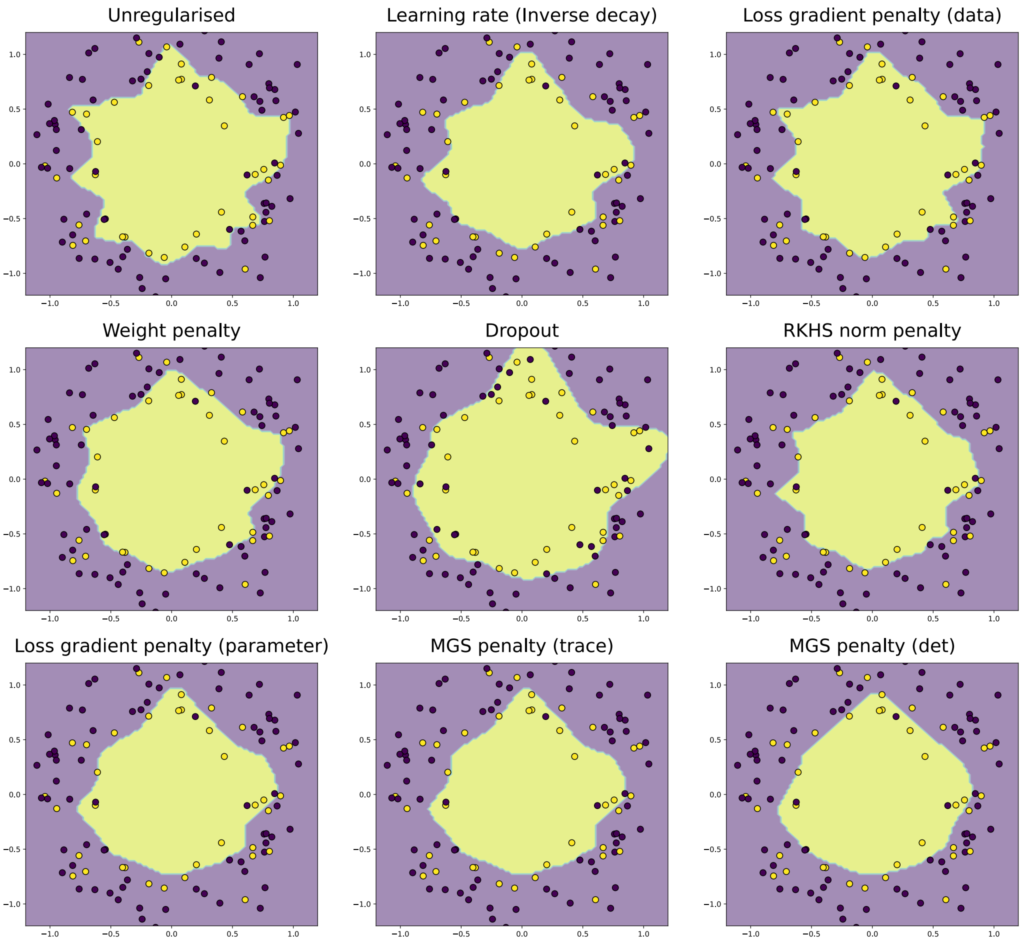


Figure A.1: Decision boundaries for all regularisation methods.

F MNIST testbench results

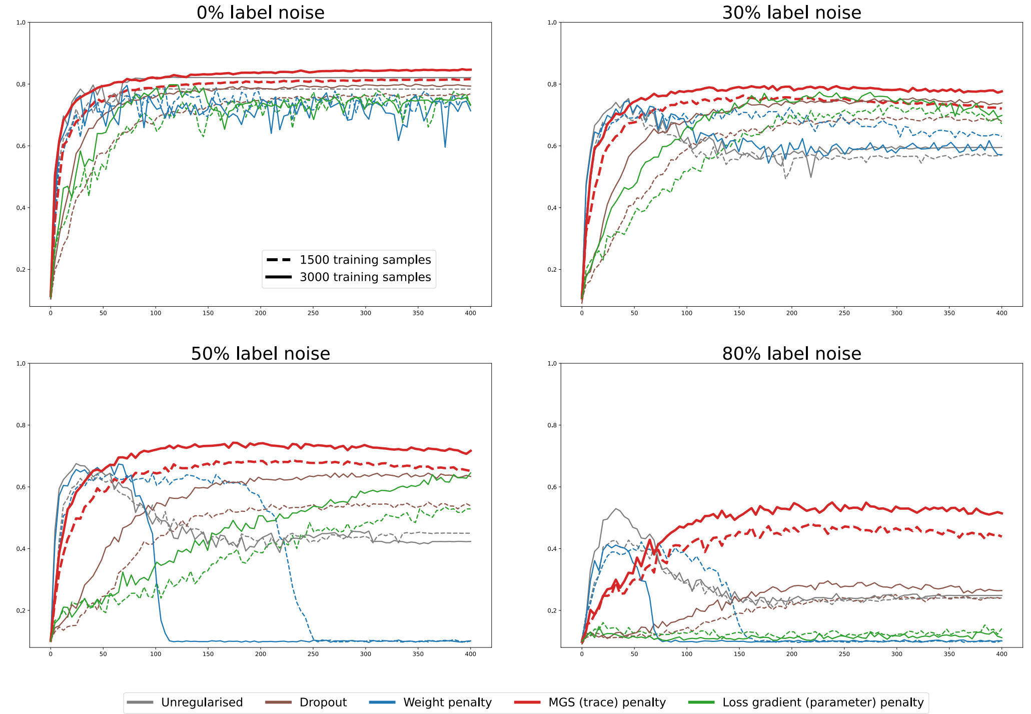


Figure A.2: **Test accuracy during training using a FCN architecture.** Test accuracy plotted during training for two training sizes and different levels of label noise.

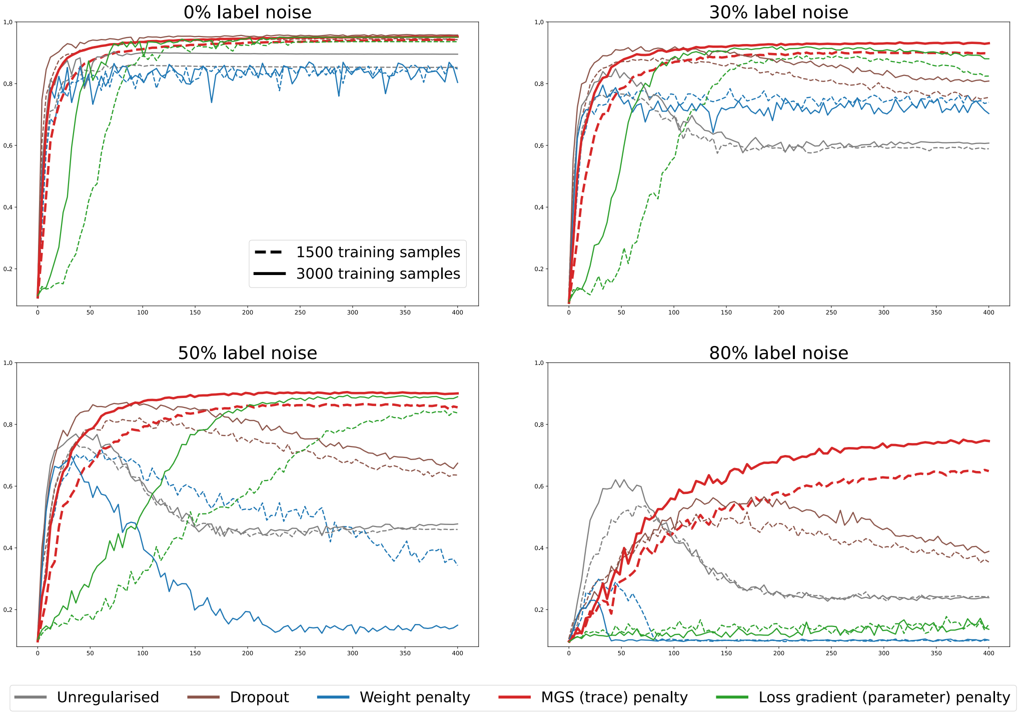
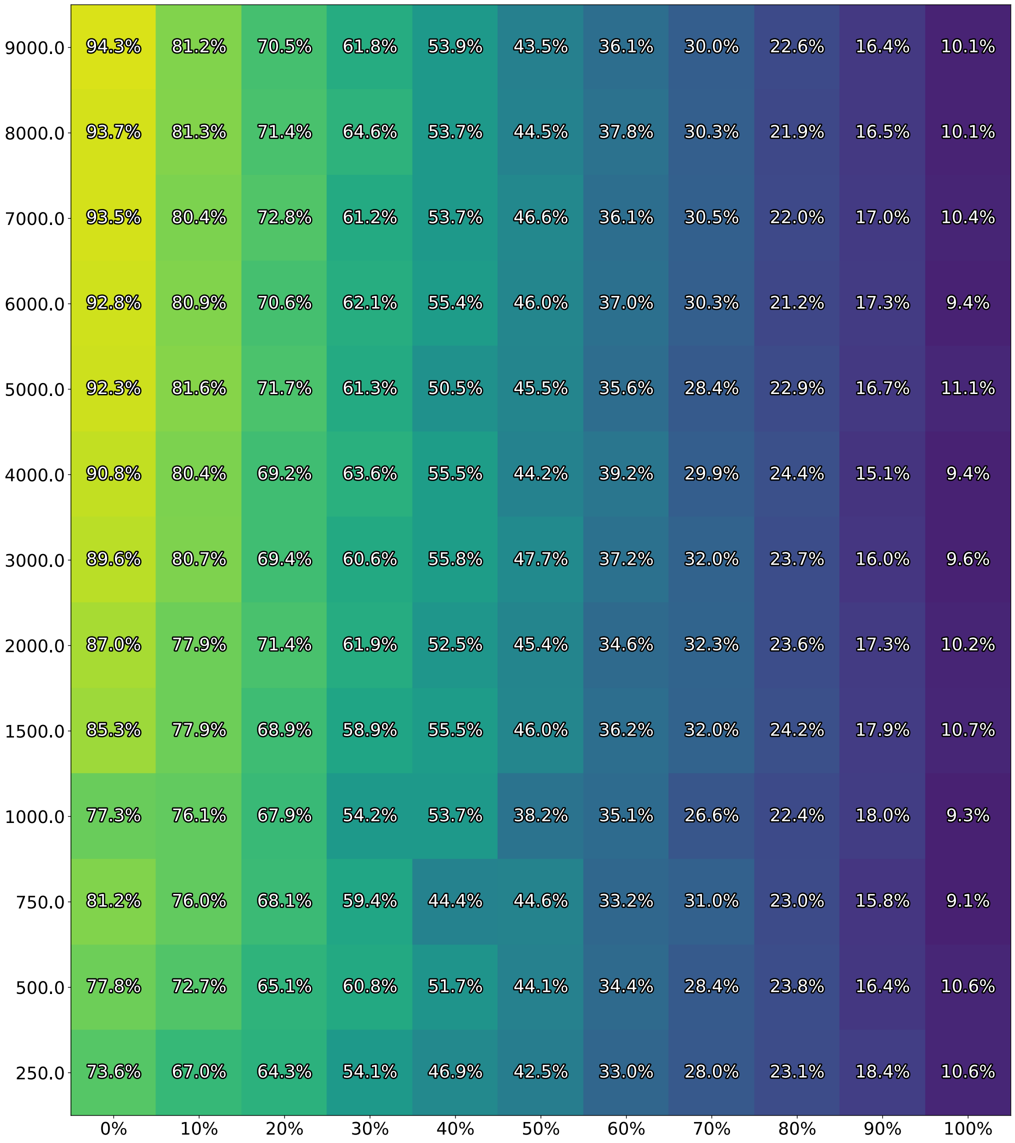
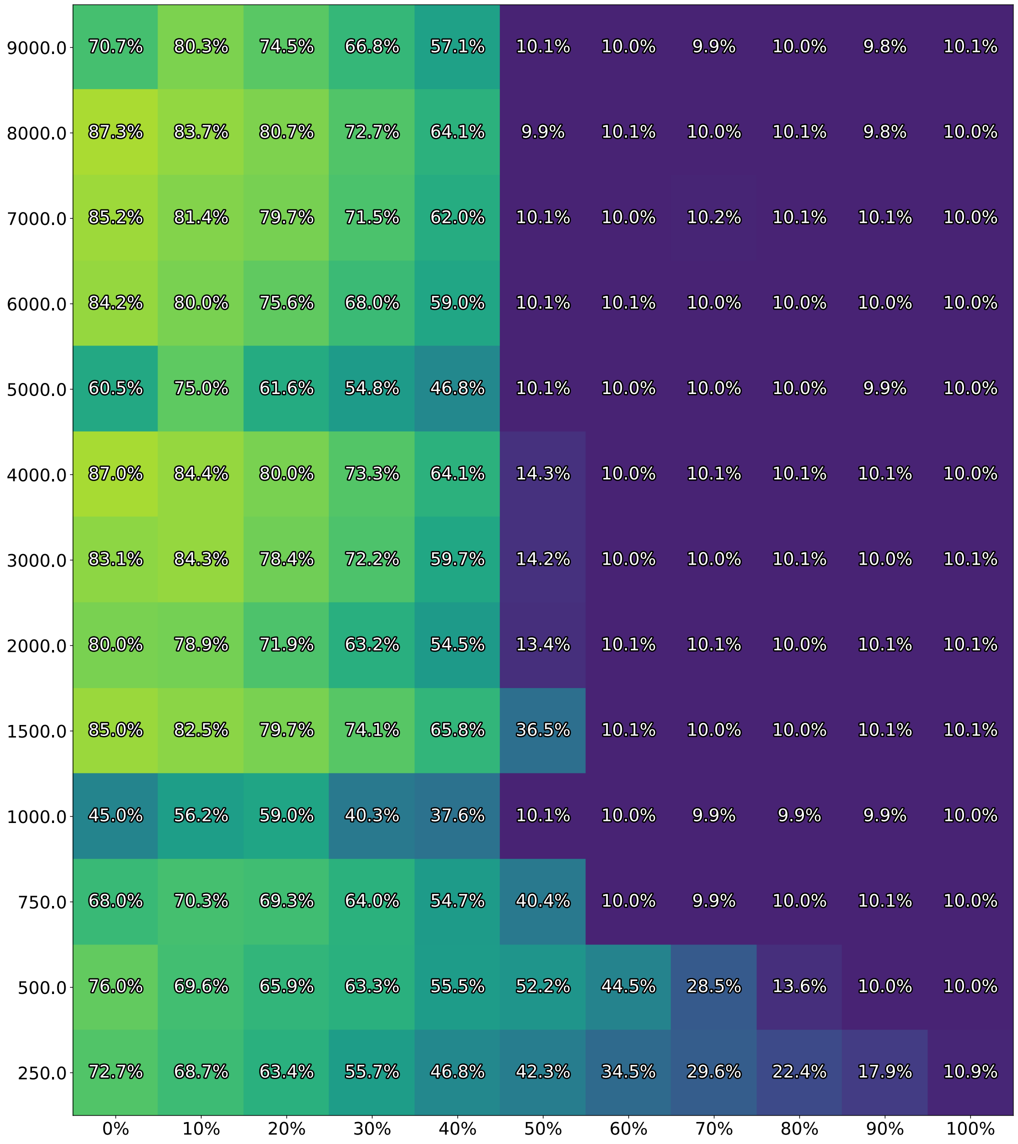


Figure A.3: **Test accuracy during training using a LeNet-5 architecture.** Test accuracy plotted during training for two training sizes and different levels of label noise.



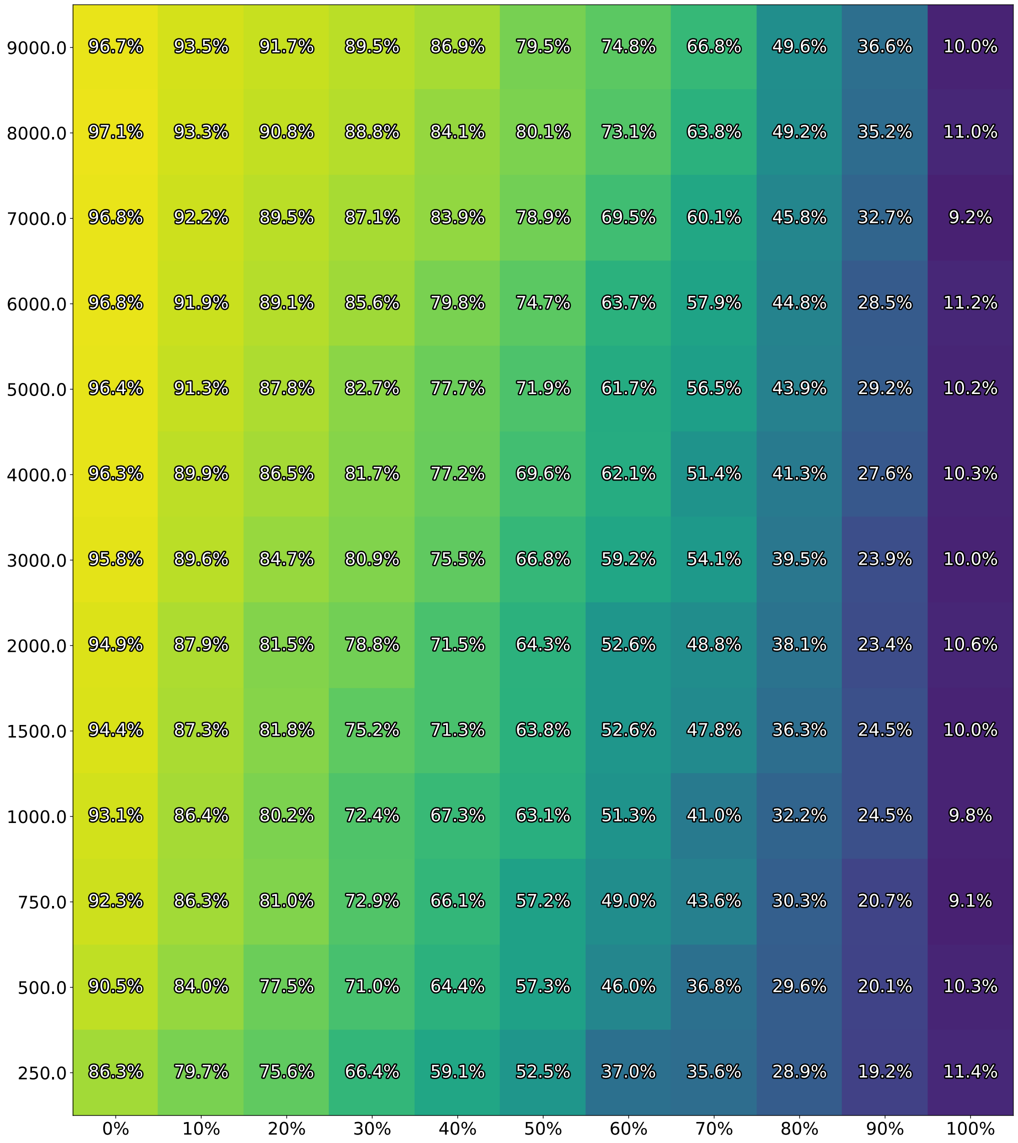
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
9000.0	94.3%	81.2%	70.5%	61.8%	53.9%	43.5%	36.1%	30.0%	22.6%	16.4%	10.1%
8000.0	93.7%	81.3%	71.4%	64.6%	53.7%	44.5%	37.8%	30.3%	21.9%	16.5%	10.1%
7000.0	93.5%	80.4%	72.8%	61.2%	53.7%	46.6%	36.1%	30.5%	22.0%	17.0%	10.4%
6000.0	92.8%	80.9%	70.6%	62.1%	55.4%	46.0%	37.0%	30.3%	21.2%	17.3%	9.4%
5000.0	92.3%	81.6%	71.7%	61.3%	50.5%	45.5%	35.6%	28.4%	22.9%	16.7%	11.1%
4000.0	90.8%	80.4%	69.2%	63.6%	55.5%	44.2%	39.2%	29.9%	24.4%	15.1%	9.4%
3000.0	89.6%	80.7%	69.4%	60.6%	55.8%	47.7%	37.2%	32.0%	23.7%	16.0%	9.6%
2000.0	87.0%	77.9%	71.4%	61.9%	52.5%	45.4%	34.6%	32.3%	23.6%	17.3%	10.2%
1500.0	85.3%	77.9%	68.9%	58.9%	55.5%	46.0%	36.2%	32.0%	24.2%	17.9%	10.7%
1000.0	77.3%	76.1%	67.9%	54.2%	53.7%	38.2%	35.1%	26.6%	22.4%	18.0%	9.3%
750.0	81.2%	76.0%	68.1%	59.4%	44.4%	44.6%	33.2%	31.0%	23.0%	15.8%	9.1%
500.0	77.8%	72.7%	65.1%	60.8%	51.7%	44.1%	34.4%	28.4%	23.8%	16.4%	10.6%
250.0	73.6%	67.0%	64.3%	54.1%	46.9%	42.5%	33.0%	28.0%	23.1%	18.4%	10.6%

Figure A.4: **Unregularised LeNet-5**. Average final test accuracy for MNIST. Label noise on the x-axis and training size on the y-axis.



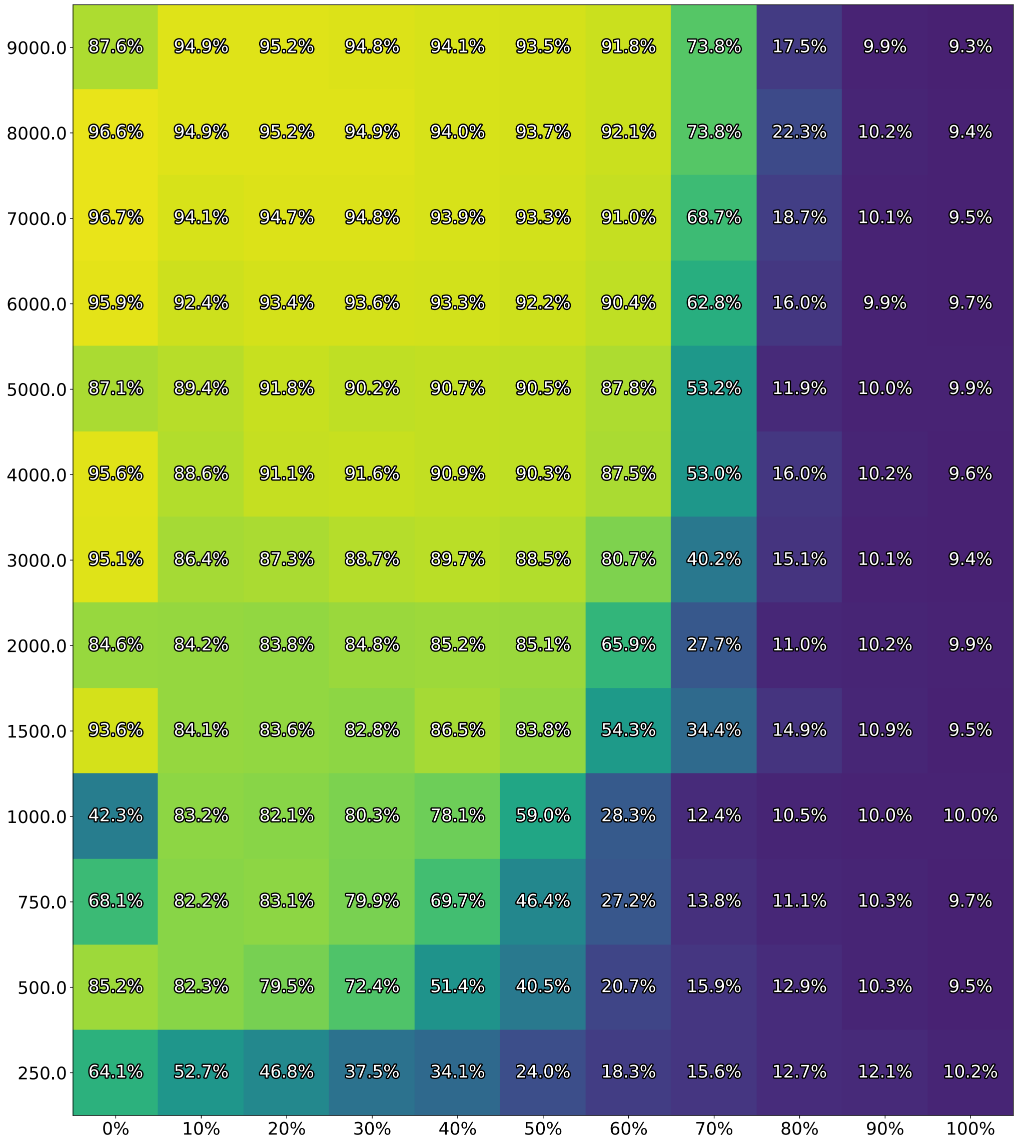
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
9000.0	70.7%	80.3%	74.5%	66.8%	57.1%	10.1%	10.0%	9.9%	10.0%	9.8%	10.1%
8000.0	87.3%	83.7%	80.7%	72.7%	64.1%	9.9%	10.1%	10.0%	10.1%	9.8%	10.0%
7000.0	85.2%	81.4%	79.7%	71.5%	62.0%	10.1%	10.0%	10.2%	10.1%	10.1%	10.0%
6000.0	84.2%	80.0%	75.6%	68.0%	59.0%	10.1%	10.1%	10.0%	10.0%	10.0%	10.0%
5000.0	60.5%	75.0%	61.6%	54.8%	46.8%	10.1%	10.0%	10.0%	10.0%	9.9%	10.0%
4000.0	87.0%	84.4%	80.0%	73.3%	64.1%	14.3%	10.0%	10.1%	10.1%	10.1%	10.0%
3000.0	83.1%	84.3%	78.4%	72.2%	59.7%	14.2%	10.0%	10.0%	10.1%	10.0%	10.1%
2000.0	80.0%	78.9%	71.9%	63.2%	54.5%	13.4%	10.1%	10.1%	10.0%	10.1%	10.1%
1500.0	85.0%	82.5%	79.7%	74.1%	65.8%	36.5%	10.1%	10.0%	10.0%	10.1%	10.1%
1000.0	45.0%	56.2%	59.0%	40.3%	37.6%	10.1%	10.0%	9.9%	9.9%	9.9%	10.0%
750.0	68.0%	70.3%	69.3%	64.0%	54.7%	40.4%	10.0%	9.9%	10.0%	10.1%	10.0%
500.0	76.0%	69.6%	65.9%	63.3%	55.5%	52.2%	44.5%	28.5%	13.6%	10.0%	10.0%
250.0	72.7%	68.7%	63.4%	55.7%	46.8%	42.3%	34.5%	29.6%	22.4%	17.9%	10.9%

Figure A.5: **LeNet-5 regularised with weight penalty**. Average final test accuracy with label noise on the x-axis and training size on the y-axis.



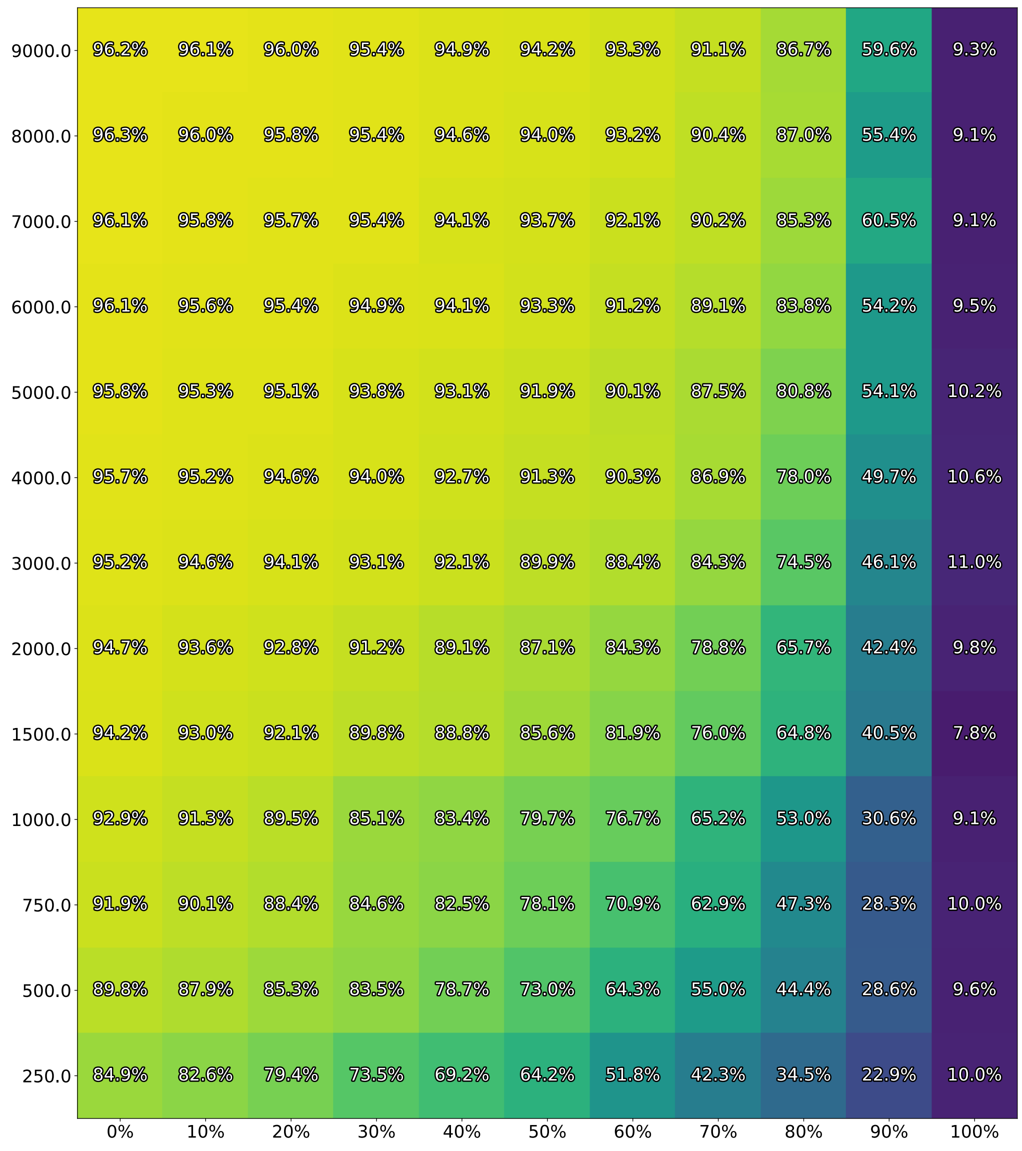
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
9000.0	96.7%	93.5%	91.7%	89.5%	86.9%	79.5%	74.8%	66.8%	49.6%	36.6%	10.0%
8000.0	97.1%	93.3%	90.8%	88.8%	84.1%	80.1%	73.1%	63.8%	49.2%	35.2%	11.0%
7000.0	96.8%	92.2%	89.5%	87.1%	83.9%	78.9%	69.5%	60.1%	45.8%	32.7%	9.2%
6000.0	96.8%	91.9%	89.1%	85.6%	79.8%	74.7%	63.7%	57.9%	44.8%	28.5%	11.2%
5000.0	96.4%	91.3%	87.8%	82.7%	77.7%	71.9%	61.7%	56.5%	43.9%	29.2%	10.2%
4000.0	96.3%	89.9%	86.5%	81.7%	77.2%	69.6%	62.1%	51.4%	41.3%	27.6%	10.3%
3000.0	95.8%	89.6%	84.7%	80.9%	75.5%	66.8%	59.2%	54.1%	39.5%	23.9%	10.0%
2000.0	94.9%	87.9%	81.5%	78.8%	71.5%	64.3%	52.6%	48.8%	38.1%	23.4%	10.6%
1500.0	94.4%	87.3%	81.8%	75.2%	71.3%	63.8%	52.6%	47.8%	36.3%	24.5%	10.0%
1000.0	93.1%	86.4%	80.2%	72.4%	67.3%	63.1%	51.3%	41.0%	32.2%	24.5%	9.8%
750.0	92.3%	86.3%	81.0%	72.9%	66.1%	57.2%	49.0%	43.6%	30.3%	20.7%	9.1%
500.0	90.5%	84.0%	77.5%	71.0%	64.4%	57.3%	46.0%	36.8%	29.6%	20.1%	10.3%
250.0	86.3%	79.7%	75.6%	66.4%	59.1%	52.5%	37.0%	35.6%	28.9%	19.2%	11.4%

Figure A.6: **LeNet-5 regularised with dropout**. Average final test accuracy with label noise on the x-axis and training size on the y-axis.



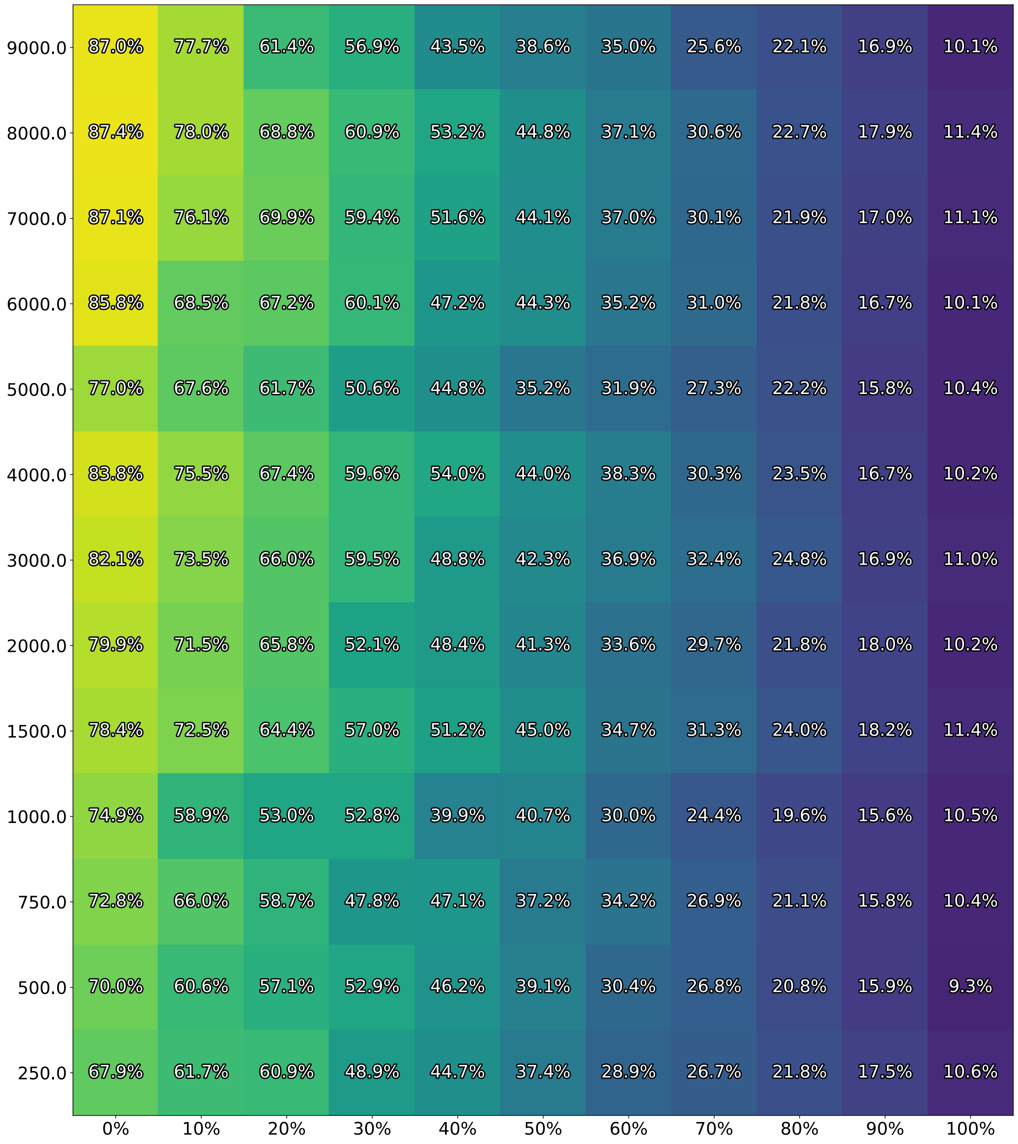
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
9000.0	87.6%	94.9%	95.2%	94.8%	94.1%	93.5%	91.8%	73.8%	17.5%	9.9%	9.3%
8000.0	96.6%	94.9%	95.2%	94.9%	94.0%	93.7%	92.1%	73.8%	22.3%	10.2%	9.4%
7000.0	96.7%	94.1%	94.7%	94.8%	93.9%	93.3%	91.0%	68.7%	18.7%	10.1%	9.5%
6000.0	95.9%	92.4%	93.4%	93.6%	93.3%	92.2%	90.4%	62.8%	16.0%	9.9%	9.7%
5000.0	87.1%	89.4%	91.8%	90.2%	90.7%	90.5%	87.8%	53.2%	11.9%	10.0%	9.9%
4000.0	95.6%	88.6%	91.1%	91.6%	90.9%	90.3%	87.5%	53.0%	16.0%	10.2%	9.6%
3000.0	95.1%	86.4%	87.3%	88.7%	89.7%	88.5%	80.7%	40.2%	15.1%	10.1%	9.4%
2000.0	84.6%	84.2%	83.8%	84.8%	85.2%	85.1%	65.9%	27.7%	11.0%	10.2%	9.9%
1500.0	93.6%	84.1%	83.6%	82.8%	86.5%	83.8%	54.3%	34.4%	14.9%	10.9%	9.5%
1000.0	42.3%	83.2%	82.1%	80.3%	78.1%	59.0%	28.3%	12.4%	10.5%	10.0%	10.0%
750.0	68.1%	82.2%	83.1%	79.9%	69.7%	46.4%	27.2%	13.8%	11.1%	10.3%	9.7%
500.0	85.2%	82.3%	79.5%	72.4%	51.4%	40.5%	20.7%	15.9%	12.9%	10.3%	9.5%
250.0	64.1%	52.7%	46.8%	37.5%	34.1%	24.0%	18.3%	15.6%	12.7%	12.1%	10.2%

Figure A.7: **LeNet-5 regularised with loss-gradient penalty (parameter)**. Average final test accuracy with label noise on the x-axis and training size on the y-axis.



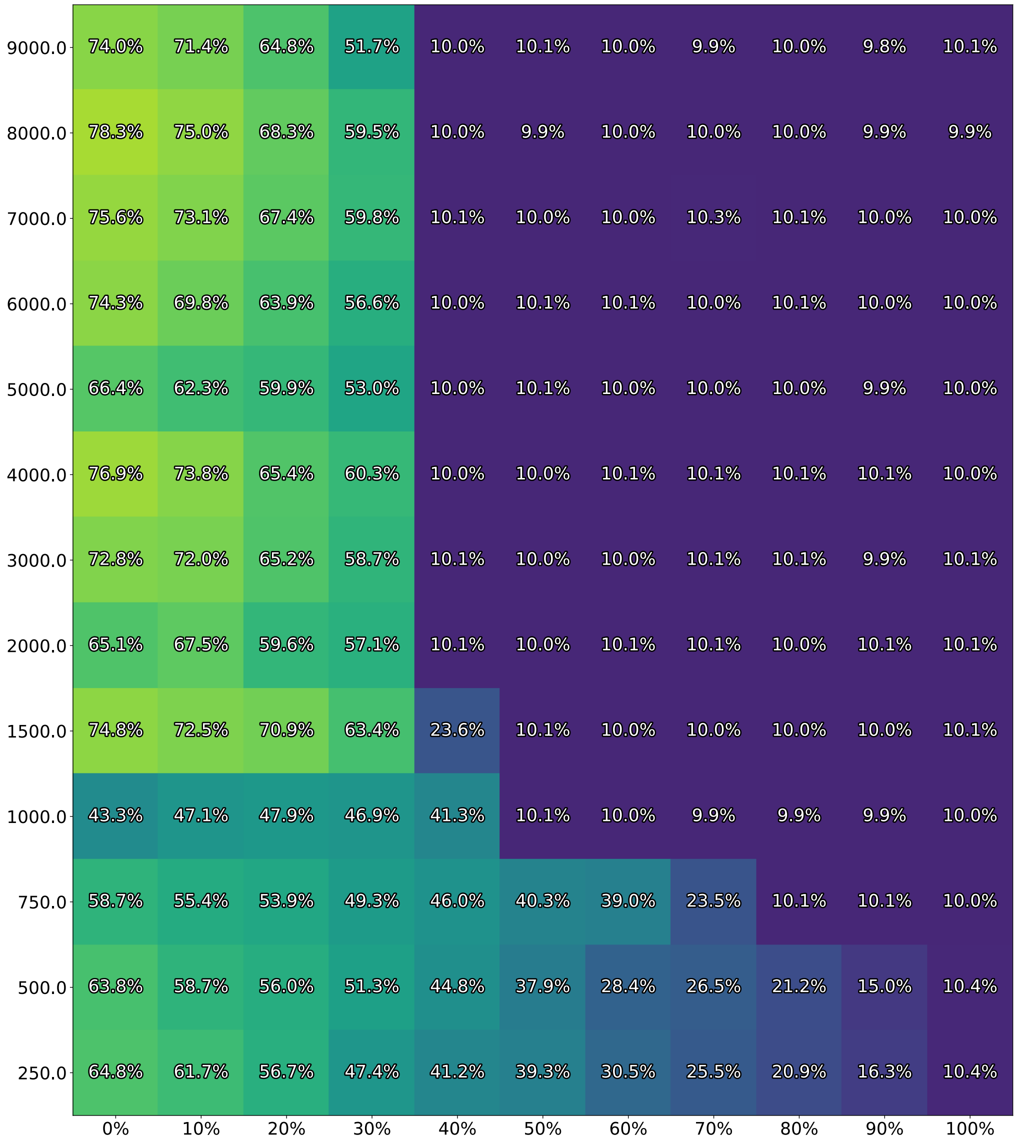
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
9000.0	96.2%	96.1%	96.0%	95.4%	94.9%	94.2%	93.3%	91.1%	86.7%	59.6%	9.3%
8000.0	96.3%	96.0%	95.8%	95.4%	94.6%	94.0%	93.2%	90.4%	87.0%	55.4%	9.1%
7000.0	96.1%	95.8%	95.7%	95.4%	94.1%	93.7%	92.1%	90.2%	85.3%	60.5%	9.1%
6000.0	96.1%	95.6%	95.4%	94.9%	94.1%	93.3%	91.2%	89.1%	83.8%	54.2%	9.5%
5000.0	95.8%	95.3%	95.1%	93.8%	93.1%	91.9%	90.1%	87.5%	80.8%	54.1%	10.2%
4000.0	95.7%	95.2%	94.6%	94.0%	92.7%	91.3%	90.3%	86.9%	78.0%	49.7%	10.6%
3000.0	95.2%	94.6%	94.1%	93.1%	92.1%	89.9%	88.4%	84.3%	74.5%	46.1%	11.0%
2000.0	94.7%	93.6%	92.8%	91.2%	89.1%	87.1%	84.3%	78.8%	65.7%	42.4%	9.8%
1500.0	94.2%	93.0%	92.1%	89.8%	88.8%	85.6%	81.9%	76.0%	64.8%	40.5%	7.8%
1000.0	92.9%	91.3%	89.5%	85.1%	83.4%	79.7%	76.7%	65.2%	53.0%	30.6%	9.1%
750.0	91.9%	90.1%	88.4%	84.6%	82.5%	78.1%	70.9%	62.9%	47.3%	28.3%	10.0%
500.0	89.8%	87.9%	85.3%	83.5%	78.7%	73.0%	64.3%	55.0%	44.4%	28.6%	9.6%
250.0	84.9%	82.6%	79.4%	73.5%	69.2%	64.2%	51.8%	42.3%	34.5%	22.9%	10.0%

Figure A.8: **LeNet-5 regularised with MGS penalty (trace)**. Average final test accuracy with label noise on the x-axis and training size on the y-axis.



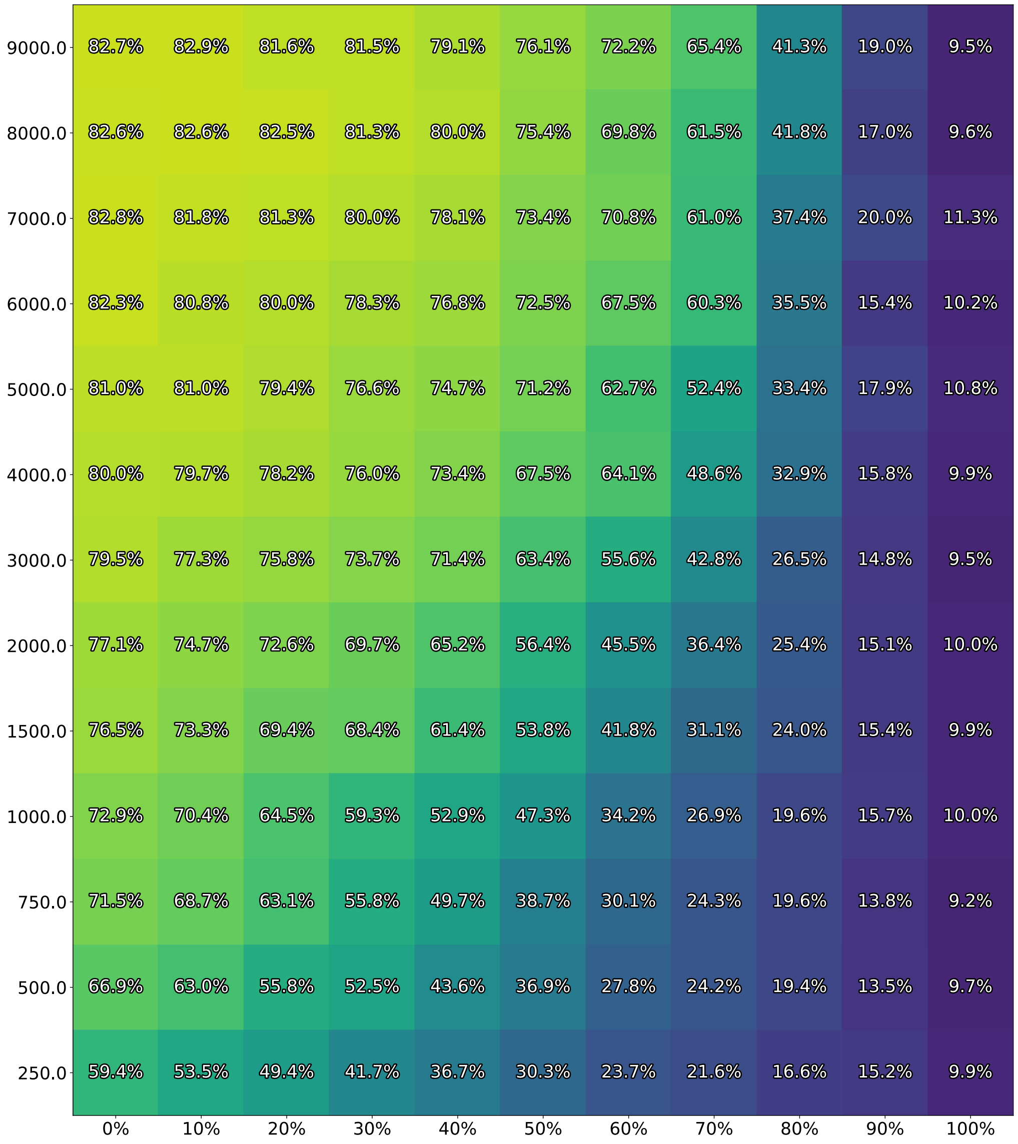
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
9000.0	87.0%	77.7%	61.4%	56.9%	43.5%	38.6%	35.0%	25.6%	22.1%	16.9%	10.1%
8000.0	87.4%	78.0%	68.8%	60.9%	53.2%	44.8%	37.1%	30.6%	22.7%	17.9%	11.4%
7000.0	87.1%	76.1%	69.9%	59.4%	51.6%	44.1%	37.0%	30.1%	21.9%	17.0%	11.1%
6000.0	85.8%	68.5%	67.2%	60.1%	47.2%	44.3%	35.2%	31.0%	21.8%	16.7%	10.1%
5000.0	77.0%	67.6%	61.7%	50.6%	44.8%	35.2%	31.9%	27.3%	22.2%	15.8%	10.4%
4000.0	83.8%	75.5%	67.4%	59.6%	54.0%	44.0%	38.3%	30.3%	23.5%	16.7%	10.2%
3000.0	82.1%	73.5%	66.0%	59.5%	48.8%	42.3%	36.9%	32.4%	24.8%	16.9%	11.0%
2000.0	79.9%	71.5%	65.8%	52.1%	48.4%	41.3%	33.6%	29.7%	21.8%	18.0%	10.2%
1500.0	78.4%	72.5%	64.4%	57.0%	51.2%	45.0%	34.7%	31.3%	24.0%	18.2%	11.4%
1000.0	74.9%	58.9%	53.0%	52.8%	39.9%	40.7%	30.0%	24.4%	19.6%	15.6%	10.5%
750.0	72.8%	66.0%	58.7%	47.8%	47.1%	37.2%	34.2%	26.9%	21.1%	15.8%	10.4%
500.0	70.0%	60.6%	57.1%	52.9%	46.2%	39.1%	30.4%	26.8%	20.8%	15.9%	9.3%
250.0	67.9%	61.7%	60.9%	48.9%	44.7%	37.4%	28.9%	26.7%	21.8%	17.5%	10.6%

Figure A.9: **Unregularised FCN**. Average final test accuracy for MNIST. Label noise on the x-axis and training size on the y-axis.



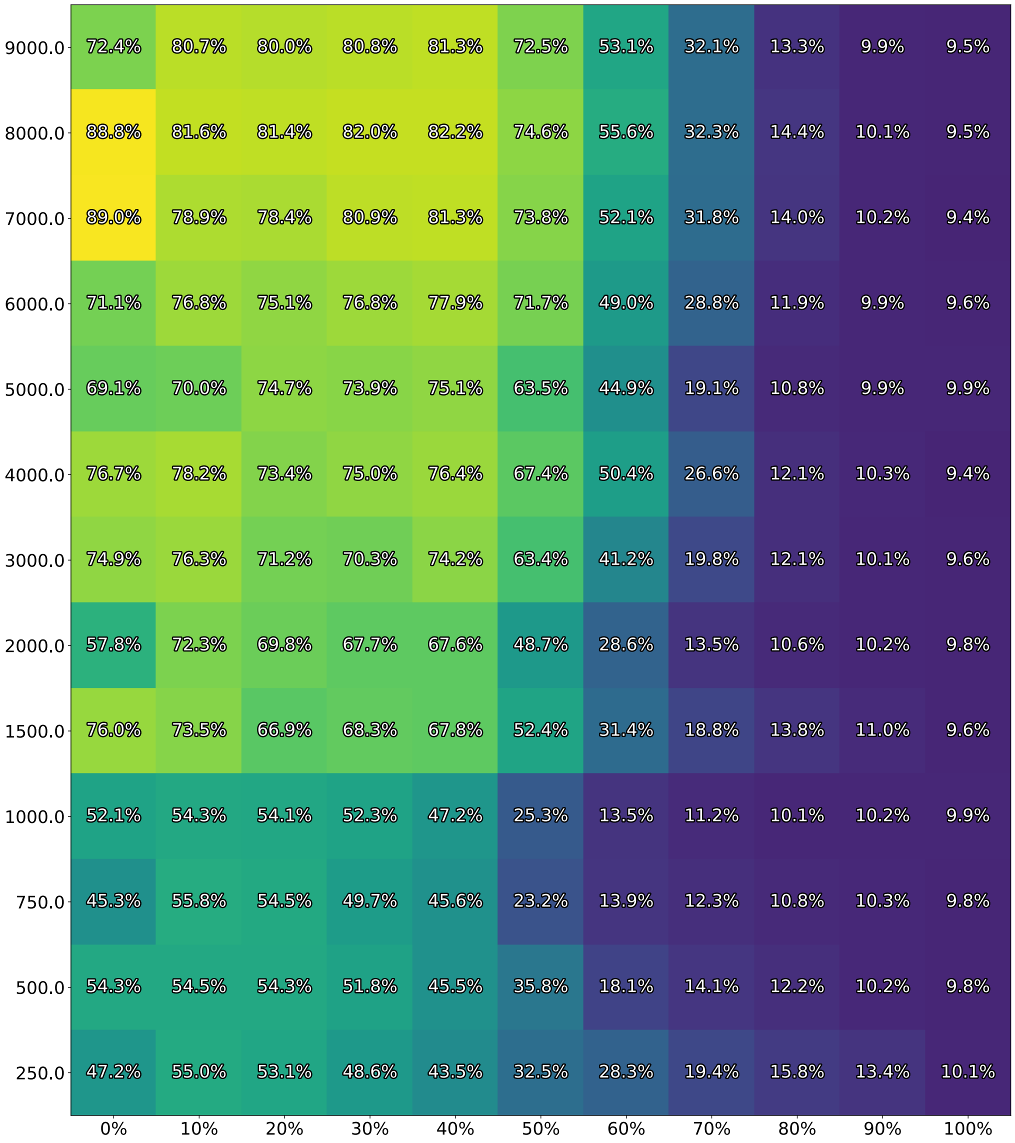
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
9000.0	74.0%	71.4%	64.8%	51.7%	10.0%	10.1%	10.0%	9.9%	10.0%	9.8%	10.1%
8000.0	78.3%	75.0%	68.3%	59.5%	10.0%	9.9%	10.0%	10.0%	10.0%	9.9%	9.9%
7000.0	75.6%	73.1%	67.4%	59.8%	10.1%	10.0%	10.0%	10.3%	10.1%	10.0%	10.0%
6000.0	74.3%	69.8%	63.9%	56.6%	10.0%	10.1%	10.1%	10.0%	10.1%	10.0%	10.0%
5000.0	66.4%	62.3%	59.9%	53.0%	10.0%	10.1%	10.0%	10.0%	10.0%	9.9%	10.0%
4000.0	76.9%	73.8%	65.4%	60.3%	10.0%	10.0%	10.1%	10.1%	10.1%	10.1%	10.0%
3000.0	72.8%	72.0%	65.2%	58.7%	10.1%	10.0%	10.0%	10.1%	10.1%	9.9%	10.1%
2000.0	65.1%	67.5%	59.6%	57.1%	10.1%	10.0%	10.1%	10.1%	10.0%	10.1%	10.1%
1500.0	74.8%	72.5%	70.9%	63.4%	23.6%	10.1%	10.0%	10.0%	10.0%	10.0%	10.1%
1000.0	43.3%	47.1%	47.9%	46.9%	41.3%	10.1%	10.0%	9.9%	9.9%	9.9%	10.0%
750.0	58.7%	55.4%	53.9%	49.3%	46.0%	40.3%	39.0%	23.5%	10.1%	10.1%	10.0%
500.0	63.8%	58.7%	56.0%	51.3%	44.8%	37.9%	28.4%	26.5%	21.2%	15.0%	10.4%
250.0	64.8%	61.7%	56.7%	47.4%	41.2%	39.3%	30.5%	25.5%	20.9%	16.3%	10.4%

Figure A.10: **FCN regularised with weight penalty**. Average final test accuracy with label noise on the x-axis and training size on the y-axis.



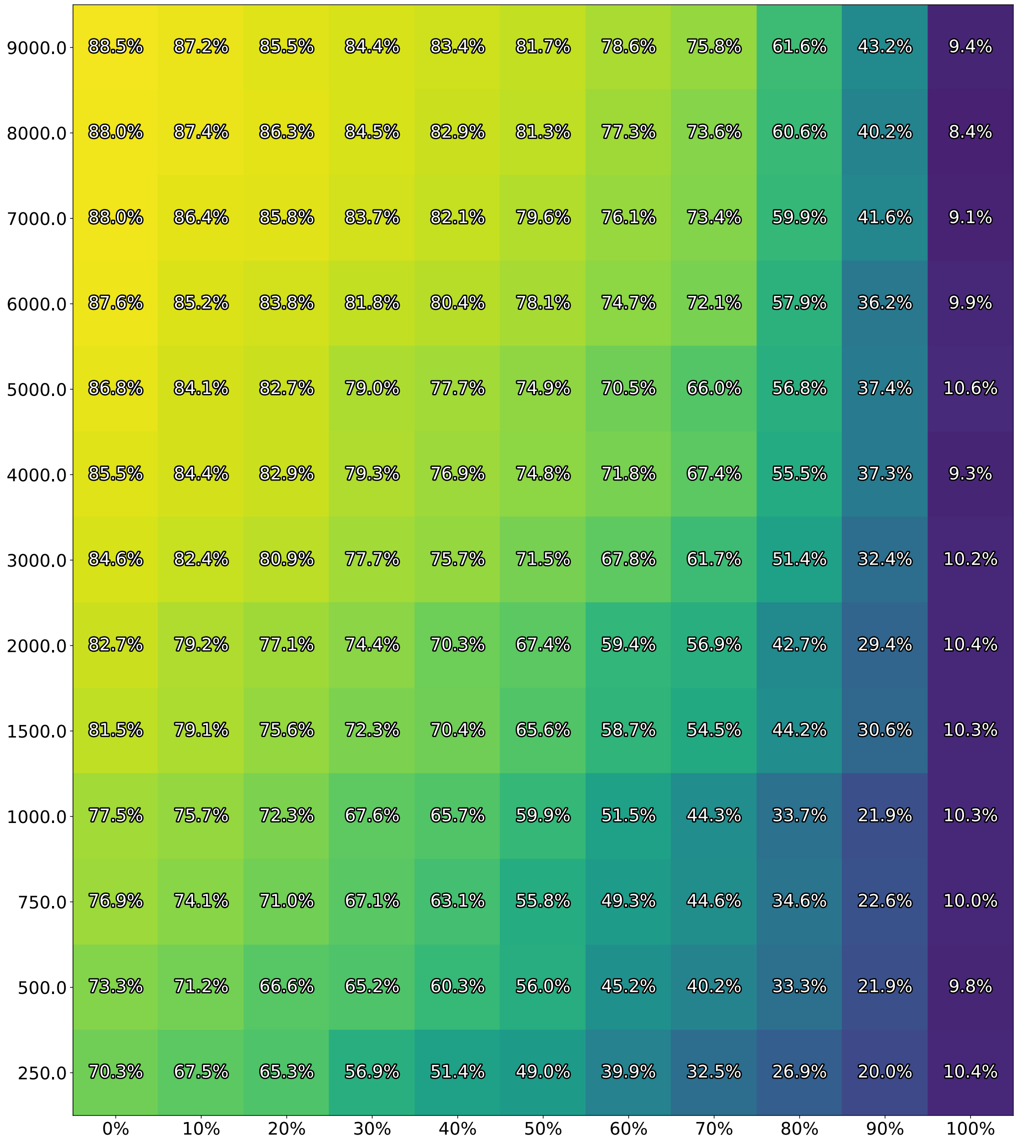
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
9000.0	82.7%	82.9%	81.6%	81.5%	79.1%	76.1%	72.2%	65.4%	41.3%	19.0%	9.5%
8000.0	82.6%	82.6%	82.5%	81.3%	80.0%	75.4%	69.8%	61.5%	41.8%	17.0%	9.6%
7000.0	82.8%	81.8%	81.3%	80.0%	78.1%	73.4%	70.8%	61.0%	37.4%	20.0%	11.3%
6000.0	82.3%	80.8%	80.0%	78.3%	76.8%	72.5%	67.5%	60.3%	35.5%	15.4%	10.2%
5000.0	81.0%	81.0%	79.4%	76.6%	74.7%	71.2%	62.7%	52.4%	33.4%	17.9%	10.8%
4000.0	80.0%	79.7%	78.2%	76.0%	73.4%	67.5%	64.1%	48.6%	32.9%	15.8%	9.9%
3000.0	79.5%	77.3%	75.8%	73.7%	71.4%	63.4%	55.6%	42.8%	26.5%	14.8%	9.5%
2000.0	77.1%	74.7%	72.6%	69.7%	65.2%	56.4%	45.5%	36.4%	25.4%	15.1%	10.0%
1500.0	76.5%	73.3%	69.4%	68.4%	61.4%	53.8%	41.8%	31.1%	24.0%	15.4%	9.9%
1000.0	72.9%	70.4%	64.5%	59.3%	52.9%	47.3%	34.2%	26.9%	19.6%	15.7%	10.0%
750.0	71.5%	68.7%	63.1%	55.8%	49.7%	38.7%	30.1%	24.3%	19.6%	13.8%	9.2%
500.0	66.9%	63.0%	55.8%	52.5%	43.6%	36.9%	27.8%	24.2%	19.4%	13.5%	9.7%
250.0	59.4%	53.5%	49.4%	41.7%	36.7%	30.3%	23.7%	21.6%	16.6%	15.2%	9.9%

Figure A.11: **FCN regularised with dropout**. Average final test accuracy with label noise on the x-axis and training size on the y-axis.



	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
9000.0	72.4%	80.7%	80.0%	80.8%	81.3%	72.5%	53.1%	32.1%	13.3%	9.9%	9.5%
8000.0	88.8%	81.6%	81.4%	82.0%	82.2%	74.6%	55.6%	32.3%	14.4%	10.1%	9.5%
7000.0	89.0%	78.9%	78.4%	80.9%	81.3%	73.8%	52.1%	31.8%	14.0%	10.2%	9.4%
6000.0	71.1%	76.8%	75.1%	76.8%	77.9%	71.7%	49.0%	28.8%	11.9%	9.9%	9.6%
5000.0	69.1%	70.0%	74.7%	73.9%	75.1%	63.5%	44.9%	19.1%	10.8%	9.9%	9.9%
4000.0	76.7%	78.2%	73.4%	75.0%	76.4%	67.4%	50.4%	26.6%	12.1%	10.3%	9.4%
3000.0	74.9%	76.3%	71.2%	70.3%	74.2%	63.4%	41.2%	19.8%	12.1%	10.1%	9.6%
2000.0	57.8%	72.3%	69.8%	67.7%	67.6%	48.7%	28.6%	13.5%	10.6%	10.2%	9.8%
1500.0	76.0%	73.5%	66.9%	68.3%	67.8%	52.4%	31.4%	18.8%	13.8%	11.0%	9.6%
1000.0	52.1%	54.3%	54.1%	52.3%	47.2%	25.3%	13.5%	11.2%	10.1%	10.2%	9.9%
750.0	45.3%	55.8%	54.5%	49.7%	45.6%	23.2%	13.9%	12.3%	10.8%	10.3%	9.8%
500.0	54.3%	54.5%	54.3%	51.8%	45.5%	35.8%	18.1%	14.1%	12.2%	10.2%	9.8%
250.0	47.2%	55.0%	53.1%	48.6%	43.5%	32.5%	28.3%	19.4%	15.8%	13.4%	10.1%

Figure A.12: **FCN regularised with loss-gradient penalty (parameter)**. Average final test accuracy with label noise on the x-axis and training size on the y-axis.



	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
9000.0	88.5%	87.2%	85.5%	84.4%	83.4%	81.7%	78.6%	75.8%	61.6%	43.2%	9.4%
8000.0	88.0%	87.4%	86.3%	84.5%	82.9%	81.3%	77.3%	73.6%	60.6%	40.2%	8.4%
7000.0	88.0%	86.4%	85.8%	83.7%	82.1%	79.6%	76.1%	73.4%	59.9%	41.6%	9.1%
6000.0	87.6%	85.2%	83.8%	81.8%	80.4%	78.1%	74.7%	72.1%	57.9%	36.2%	9.9%
5000.0	86.8%	84.1%	82.7%	79.0%	77.7%	74.9%	70.5%	66.0%	56.8%	37.4%	10.6%
4000.0	85.5%	84.4%	82.9%	79.3%	76.9%	74.8%	71.8%	67.4%	55.5%	37.3%	9.3%
3000.0	84.6%	82.4%	80.9%	77.7%	75.7%	71.5%	67.8%	61.7%	51.4%	32.4%	10.2%
2000.0	82.7%	79.2%	77.1%	74.4%	70.3%	67.4%	59.4%	56.9%	42.7%	29.4%	10.4%
1500.0	81.5%	79.1%	75.6%	72.3%	70.4%	65.6%	58.7%	54.5%	44.2%	30.6%	10.3%
1000.0	77.5%	75.7%	72.3%	67.6%	65.7%	59.9%	51.5%	44.3%	33.7%	21.9%	10.3%
750.0	76.9%	74.1%	71.0%	67.1%	63.1%	55.8%	49.3%	44.6%	34.6%	22.6%	10.0%
500.0	73.3%	71.2%	66.6%	65.2%	60.3%	56.0%	45.2%	40.2%	33.3%	21.9%	9.8%
250.0	70.3%	67.5%	65.3%	56.9%	51.4%	49.0%	39.9%	32.5%	26.9%	20.0%	10.4%

Figure A.13: **FCN regularised with MGS penalty (trace)**. Average final test accuracy with label noise on the x-axis and training size on the y-axis.

G Calculation of K_θ

G.1 Vector output models

In section 2 K_θ is defined for a scalar output model and therefore is a matrix of size $n \times n$. In the case of a vector output model K_θ becomes a tensor of size $n \times n \times q \times q$ where q is the number of outputs. The reasoning about MGS still holds. However, the calculations become more involved as one not only has to consider the model update for a single output (given by the diagonal matrices $K_\theta^{i,j}, i = j$) but also how an update for one output affects other outputs (given by the off diagonal matrices $K_\theta^{i,j}, i \neq j$). Therefore, it is easier to instead combine all the outputs by concatenating their model gradients and then calculate a single kernel K_θ of size $np \times np$, as is common in the NTK literature [Lee et al., 2019]:

$$\nabla_\theta f = [\nabla_\theta f_1; \cdots ; \nabla_\theta f_q].$$

G.2 Handling of mini-batches and large data sets

When using mini-batches, K_θ is no longer square and instead will be of size $n \times m$, where m is the number of samples in the current batch.

Both this matrix and the $n \times n$ matrix from standard gradient descent can be infeasible to calculate for large data sets. Instead, an approximation is calculated using only the data from the current batch which will be of size $m \times m$. Throughout the entire article, K_θ is always calculated for the current batch and not the full dataset.

H Relation between spectrum of MGS kernel and gradient sample covariance matrix

Proposition 1 *The matrices X^TX (Gram matrix) and XX^T (scatter matrix) share the same non-zero eigenvalues.*

Proof: Suppose that λ is a non-zero eigenvalue of X^TX with the associated eigenvector u.

Then:

$$\begin{aligned} X^TXu &= \lambda u \\ XX^TXu &= X\lambda u \\ XX^T(Xu) &= \lambda(Xu) \\ XX^T\tilde{u} &= \lambda\tilde{u}. \end{aligned}$$

Thus, λ is an eigenvalue of XX^T, with the associated eigenvector $\tilde{u} = XU$.■

Theorem 1 *Separation theorem [Takane and Shibayama, 1991].*

Let M be a d-by-n matrix. Let two orthogonal projection matrices be P_{left} and P_{right} of size d-by-d and n-by-n respectively.

Then:

$$\sigma_{j+t}(M) \leq \sigma_j(P_{\mathit{left}} M P_{\mathit{right}}) \leq \sigma_j(M),$$

where $\sigma_j(\cdot)$ denotes the j-th largest singular value of the matrix, while $t = d - r(P_{\mathit{left}}) + n - r(P_{\mathit{right}})$ and $r(\cdot)$ is the rank of the matrix.

Theorem 2 *Let C and $\bar{C}$ be the scatter matrix and its centred counterpart, then their eigenvalues are interlaced, such that [Honeine, 2014]:*

$$\lambda_{j+1} \leq \bar{\lambda}_j \leq \lambda_j.$$

Proof: Let $M = X$, P_{left} being the d-by-d identity matrix and $P_{\mathit{right}} = (I - \frac{1}{n}11^T)$ which is the n-by-n centering matrix.

With $r(P_{\text{left}}) = d$ and $r(P_{\text{right}}) = n - 1$, it follows from the Separation theorem 1 that:

$$\sigma_{j+1}(X) \leq \sigma_j(X - (I - \frac{1}{n}11^T)) \leq \sigma_j(X)$$

Furthermore it is well known that the eigenvalues of C are equal to the square roots of the singular values of X.■

By using proposition 1 and theorem 2 we see that the eigenvalues of the MGS kernel, which is the Gram matrix for the model-gradients, and the centred scatter matrix (sample-covariance matrix) will be interlaced.