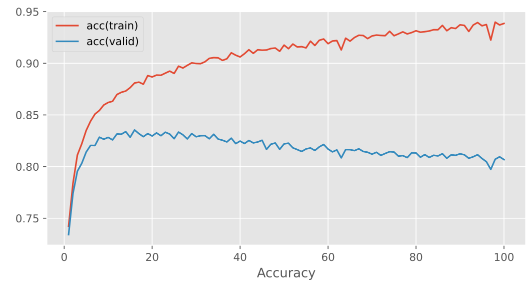
MLP Coursework 1

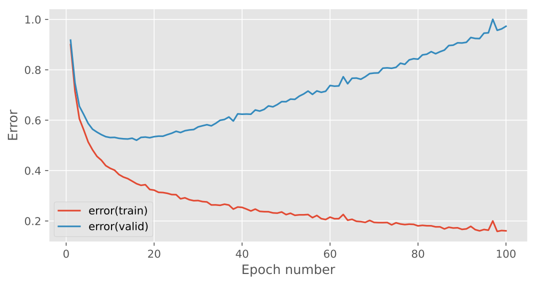
Xiaoyu Jiang

Abstract

In this report we study the problem of overfitting, which is a phenomenon that a machine learning model so flexible that fit very closely to training data, but perform bad on test data . We first analyse the given example and discuss the probable causes of the underlying problem. Then we investigate how the depth and width of a neural network can affect overfitting in a feedforward architecture and observe that increasing width and depth the problem of overfitting becomes more serious . Next we discuss why two standard methods, Dropout and Weight Penalty, can mitigate overfitting, then describe their implementation and use them in our experiments to reduce the overfitting on the EMNIST dataset. Based on our results, we ultimately find that both Dropout and Weight Penalty method have significant contribution in mitigating the severity of overfitting problems, and the combination of L2 penalty and Dropout gives the best performance . Finally, we briefly review another method, Maxout, discuss its strengths and weaknesses, and conclude the report with our observations and future work. Our main findings indicate that wider and deeper neuron network models are more prone to be overfitted, but various approaches have been showed effectiveness, such as Dropout, L1/L2 regularization .



(a) accuracy by epoch



(b) error by epoch

Figure 1. Training and validation curves in terms of classification accuracy (a) and cross-entropy error (b) on the EMNIST dataset for the baseline model.

1. Introduction

In this report we focus on a common and important problem while training machine learning models known as overfitting, or overtraining, which is a phenomenon that a machine learning model so flexible that fit very closely to training data, but perform bad on test data . We first start with analyzing the given problem in Fig. 1, study it in different architectures and then investigate different strategies to mitigate the problem. In particular, Section 2 identifies and discusses the given problem, and investigates the effect of network width and depth in terms of generalization gap (see Ch. 5 in Goodfellow et al. 2016) and generalization performance. Section 3 introduces two regularization techniques to alleviate overfitting: Dropout (Srivastava et al., 2014) and L1/L2 Weight Penalties (see Section 7.1 in Goodfellow et al. 2016). We first explain them in detail and discuss why they are used for alleviating overfitting. In Section 4 we incorporate each of them and their various combinations to a three hidden layer neural network, train it on the EMNIST dataset, which contains 131,600 images of characters and digits, each of size 28x28 which are split into 47 classes, grouping together some difficult to distinguish characters. We evaluate them in terms of generalization gap and performance, and discuss the results and effectiveness of the tested regularization strategies. Our results show that both Dropout and Weight Penalty method have significant contribution in mitigating the severity of overfitting problems, and the combination of L2 penalty and Dropout gives the best performance . In Section 5, we discuss a related work on Maxout Networks and highlight its pros and cons.[1] Finally, we conclude our study in section 6, noting that wider and deeper neuron network models are more prone to be overfitted, but various approaches have been showed effectiveness, such as Dropout, L1/L2 regularization .

2. Problem identification

Overfitting to training data is a very common and important issue that needs to be dealt with when training neural networks or other machine learning models in general (see Ch. 5 in Goodfellow et al. 2016). A model is said to be overfitting when its training error is low, yet a much significant generalization error. Overfitted model has large generalization gap: it has great performance on training dataset, while badly performed on unseen test dataset .

Overfitting occurs when our model is too flexible and the training dataset is relatively small at the same time, thus the model learns too many unnecessary details and noise in the training data to the extent that it negatively impacts the performance on unseen data. The noise and random fluctuation the model learned can not be applied to new data, which decreases the model ability to generalize. One can identify this happens when checking its generalization gap and comparing its performance on training and testing dataset. One important sign of overfitting is the generalization gap is large: model performance on training dataset is much better than on test dataset .

Fig. 1a and 1b show a prototypical example of overfitting. We see in Figure 1a that the accuracy on both training and validation datasets are increasing quickly during first 10 epochs, while after that, model accuracy continues increasing on training set but starts slow decreasing on validation set and the accuracy gap increases during this time. Finally, after 100 epochs, there are a huge gap between accuracy on training and validation set, which are around 0.94 and 0.81 respectively. In figure 1b, We have opposite trend with cross-entropy error as the indicator. The tuning point is still at about 10th epoch, both training error and validation error decrease in rapid pace during first 10 epochs. Training error decreases continuously, but with slower speed, to less than 0.2 while validation error begin to take off, reaching 1.2 after 100 epochs. .

The extent to which our model overfits depends on many factors. For example, the quality and quantity of the training set and the complexity of the model. If we have a lot of varied training samples, or if our model is relatively shallow, it will in general be less prone to overfitting. Any form of regularisation will also limit the extent to which the model overfits.

2.1. Network width

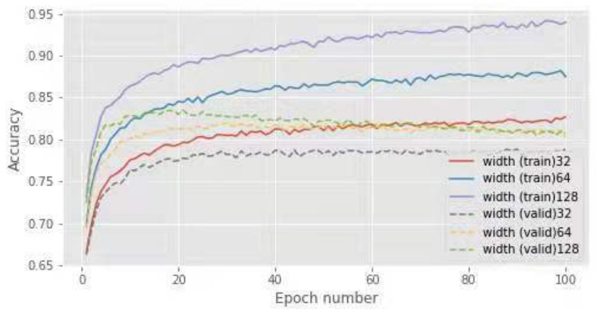
First we investigate the effect of increasing the number of hidden units in a single hidden layer network when training on the EMNIST dataset. The network is trained using the Adam optimizer with a learning rate of 10^{-3} and a batch size of 100, for a total of 100 epochs.

The input layer is of size 784, and output layer consists of 47 units. Three different models were trained, with a single hidden layer of 32, 64 and 128 ReLU hidden units respectively. Figure 2 depicts the error and accuracy curves over 100 epochs for the model with varying number of hidden units. Table 1 reports the final accuracy and generalization gap. We observe that for all three settings, the models perform much better on training set than validation set, in terms of both accuracy and cross-entropy error, which indicating all three models face serious overfitting problems. In addition, we observe that 128 neurons model having largest generalization gap and best performance in training set but worst in validation set, which means its overfitting issue is the most, while generalization gap with 32 neurons model is the smallest, meaning its overfitting problem is less severe. Generally, we find overfitting become increasingly serious as number of neurons increase .

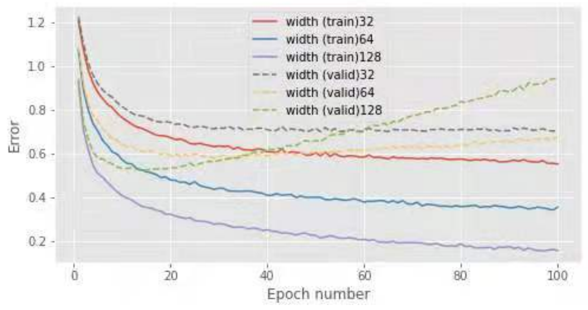
From our observation above, we find the width affects the performance results in a consistent way. More specifically, wider neuron network suffer more severe overfitting problem. This is as what we expected from the model flexibility point of view. The model with more neurons has more

# hidden units	val. acc.	generalization gap
32	0.788	0.151
64	0.802	0.324
128	0.807	0.785

Table 1. Validation accuracy (%) and generalization gap (in terms of cross-entropy error) for varying network widths on the EMNIST dataset.



(a) accuracy by epoch



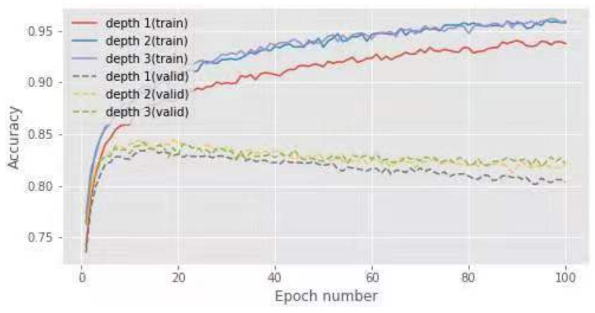
(b) error by epoch

Figure 2. Training and validation curves in terms of classification accuracy (a) and cross-entropy error (b) on the EMNIST dataset for different network widths.

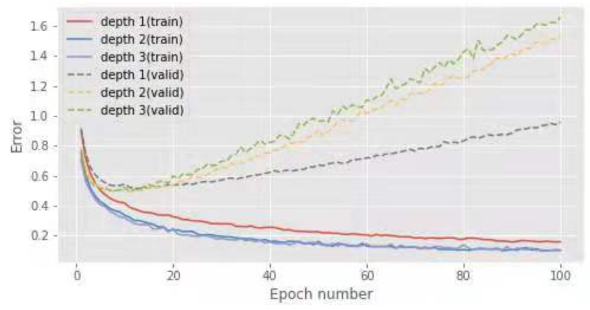
[1]Note: Omitting this for this coursework, but normally you would be more specific and summarise your conclusions about that review here as well.

# hidden layers	val. acc.	generalization gap
1	0.803	0.799
2	0.819	1.425
3	0.821	1.555

Table 2. Validation accuracy (%) and generalization gap (in terms of cross-entropy error) for varying network depths on the EMNIST dataset.



(a) accuracy by epoch



(b) error by epoch

Figure 3. Training and validation curves in terms of classification accuracy (a) and cross-entropy error (b) on the EMNIST dataset for different network depths.

parameters and larger flexibility, which is prone to become overfitting .

2.2. Network depth

Next we investigate the effect of varying the number of hidden layers in the network. Table 2 and Fig. 3 shows the results from training three models with one, two and three hidden layers respectively, each with 128 ReLU hidden units. As with previous experiments, they are trained with the Adam optimizer with a learning rate of 10^{-3} and a batch size of 100.

We observe that all models with three different depth are overfitted. Increasing the number of layers will lead to severer overfitting.From figure 3(a) for instance, it is clear that the model with 3 hidden layers reaches highest accuracy point at roughly 10th epoch in validation set, and then its accuracy starts to shrink slowly. The 3 hidden layers model has largest validation error and one of the lowest validation accuracy, while this is just contrary on training set. The generalization gap for this model is also the largest, indicating this is the most overfitted model. The 1 hidden layer model has less serious overfitting problem and much smaller generalization gap .

From our observation above, we also find the depth affects the performance results in a consistent way which is similar to the width. This is also explainable by analyzing model flexibility. The model with more hidden layers definitely has more neurons, resulting larger complexity and flexibility, which is prone to become overfitting .

From experiments above, we find that increasing width or depth of neuron network will both get model performance improved on training set but decreased on validation set. They both increases generalization gap and lead to overfitting. This can be understood because increasing both width and depth will lead to increase of model complexity(flexibility), resulting to overfitting .

3. Dropout and Weight Penalty

In this section, we investigate three regularization methods to alleviate the overfitting problem, specifically dropout layers and the L1 and L2 weight penalties.

3.1. Dropout

Dropout (Srivastava et al., 2014) is a stochastic method that randomly inactivates neurons in a neural network according to an hyperparameter, the inclusion rate. Dropout is commonly represented by an additional layer inserted between the linear layer and activation function. Its forward propagation during training is defined as follows:

$$mask \sim bernoulli(p) \quad (1)$$
$$y' = mask \odot y \quad (2)$$

where $y, y' \in \mathbb{R}^d$ are the output of the linear layer before and after applying dropout, respectively. $mask \in \mathbb{R}^d$ is a mask vector randomly sampled from the Bernoulli distribution with parameter of inclusion probability p, and $\odot$ denotes the element-wise multiplication.

At inference time, stochasticity is not desired, so no neurons are dropped. To account for the change in expectations of the output values, we scale them down by the inclusion rate p:

$$y' = y * p \quad (3)$$

As there is no nonlinear calculation involved, the backward propagation is just the element-wise product of the gradients with respect to the layer outputs and mask created in the forward calculation. The backward propagation for dropout is therefore formulated as follows:

$$\frac{\partial y'}{\partial y} = mask \tag{4}$$

Dropout is an easy to implement and highly scalable method. It can be implemented as a layer-based calculation unit, and be placed on any layer of the neural network at will. Dropout can reduce the dependence of hidden features between layers so that the neurons of the next layer will not specifically depend on some features from of the previous layer. Instead, it force the network to evenly distribute information among all features. By randomly dropping some neurons in training, dropout makes use of a subset of the whole architecture, so it can also be viewed as bagging different sub networks and averaging their outputs.

3.2. Weight penalty

L1 and L2 regularization (Ng, 2004) are simple but effective methods to mitigate overfitting to training data. The idea of L1/L2 regularization is adding a complexity term to error function, so that the network function gets smoother after training. Both L1 and L2 need hyper-parameters for controlling penalty term. L1 regularization corresponds to adding a term based on summing the absolute values of the weights to the error, similarly, L2 is defined as half of the sum of square of the weight:

$$E = E_{train} + \beta E_{L1} = E_{train} + \beta \sum_i |w_i| \tag{5}$$

$$E_w = E_{L2} = \frac{1}{2} \sum_i w_i^2 \tag{6}$$

.

Training error function with L1/L2 regularization term will discourage the model coefficients become huge, which limits the model flexibility by design and stops being overfitted to training data. Minimising L2 regularized sum of squares error function can be regarded as maximizing a posterior,(Bishop, 2006),(Goodfellow et al., 2016) with Gaussian prior term explicitly introduced to deal with insufficient data problem. L1 regularization used in cost function is equivalent to maximizing a posterior when prior is define as an isotropic Laplace distribution.(Goodfellow et al., 2016).Overfitting problem often occurs when Machine Learning model holds much more parameters than given data.This issue can be mitigated by reducing unnecessary parameters, that is introducing sparsity of the model. This can be done by adding constrains of number of model parameters and L1 regularisation can be regarded as an approximation of that approach.(Rosasco, 2009). L1 and L2 regularization differ in several points:

- Firstly, L1 and L2 have different rate of enforcing weights shrinking to zero, in L2, the weights shrink to 0 at a rate proportional to the size of the weights (βw_i), while in L1, they shrink to 0 at constant rate $\beta sign(w_i)$. Thus, the L1 and L2 regularisation behave differently for small and large weight: when $|w_i|$ is small, L1 penalty shrinks faster, since tiny weight limits the effect of L2 weight penalty; but when $|W_i|$ is large, L2 shrinks faster since the proportionality.

- Secondly, generally speaking, L1 has the property of shrinking less important feature coefficient to zero while leaving a few large important weights, encouraging sparse weight space. This can be understood as feature selection if we have huge amount of features(Goodfellow et al., 2016).

- Thirdly, L2 is less computationally expensive than L1. Optimisation with L2 regularisation leads to have a closed form in many cases, while L1 regularisation is computationally hard since it cannot be accelerated by solving in terms of matrix calculation.

.

4. Balanced EMNIST Experiments

Here we evaluate the effectiveness of the given regularization methods for reducing the overfitting on the EMNIST dataset. We build a baseline architecture with three hidden layers, each with 128 neurons, which suffers from overfitting in EMNIST as shown in section 2.

We start by lowering the learning rate to 10^{-4}, as the previous runs were overfitting in only a handful of epochs. Results for the new baseline (c.f. Table 3) confirm that lowering learning rate helps, so all further experiments are run using it.

Then, we apply the L1 or L2 regularization with dropout to our baseline and search for good hyperparameters on the validation set. We summarize all the experimental results in Table 3. For each method, we plot the relationship between generalisation gap and validation accuracy in Figure 4.

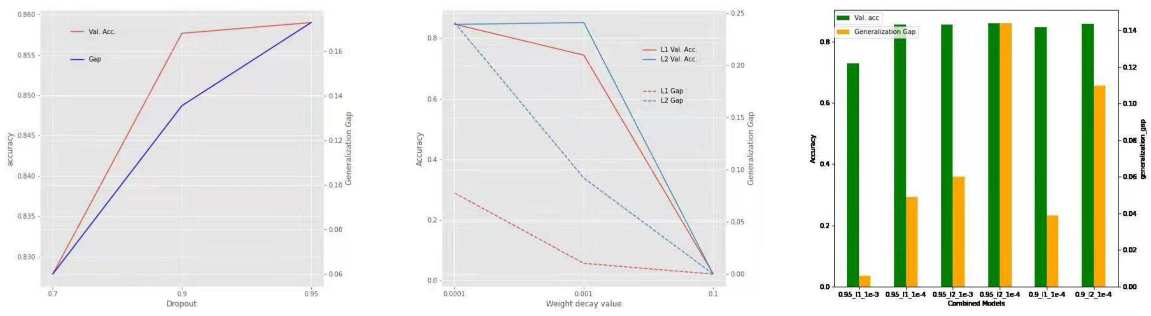
First we analyze three methods separately, train each over a set of hyperparameters and compare their best performing results.

For dropout approach, we have experimented with hyperparameters 0.7,0.9,0.95, and validation accuracy are 0.828,0.858 and 0.859. For the cases of hyperparameter (*inclusive prob*) 0.9 and 0.95, both validation accuracy are higher than baseline performance of 0.836, and generalization gap are narrower than 0.29 of baseline. This suggest the dropout regularisation improves generalization performance and mitigates overfitting problem. Among these hyperparameters, dropout model with *inclusive prob* = 0.95 gets best performance on validation set, and the final accuracy on test set is 0.845.

In experiments with L1/L2 weight penalty, we test several models with weight decay value 10^{-5}, 10^{-4} and 10^{-1}. For L1 penalty, We find model with l1 penalty improved compared with baseline only for hyperparamter=1e-4, which

Model	Hyperparameter value(s)	Validation accuracy	Generalization gap
Baseline	-	0.836	0.290
Dropout	0.7	0.828	0.060
	0.9	0.858	0.136
	0.95	*0.859*	0.173
L1 penalty	1e-4	*0.846*	0.078
	1e-3	0.744	0.010
	1e-1	0.021	-7.740e-9
L2 penalty	1e-4	0.846	0.241
	1e-3	*0.852*	0.092
	1e-1	0.020	7.845e-5
Combined	Dropout 0.95, L1 1e-3	0.729	0.006
	Dropout 0.95, L1 1e-4	0.856	0.049
	Dropout 0.95, L2 1e-4	***0.860***	0.144
	Dropout 0.95, L2 1e-3	0.855	0.060
	Dropout 0.9, L1 1e-4	0.848	0.039
	Dropout 0.9, L2 1e-4	0.858	0.110

Table 3. Results of all hyperparameter search experiments. *italics* indicate the best results per series and **bold** indicate the best overall



(a) Metrics by inclusion rate (b) Metrics by weight penalty (c) Extra experiments

Figure 4. Hyperparameter search for every method and combinations

also have smaller generalization gap, suggesting overfitting problem get mitigated. For 1e-3 and 1e-1, the model performed bad on validation set with small generalization gap. This might because penalty term is so large that the models become underfitting. It is noticeable that for penalty parameter of 1e-1, the generalization gap is slightly negative, which suggests training error become slightly larger than validation set. This phenomenon cannot happen for a well trained model. This observed issue might caused by serious under-training caused by large penalty term, and the model even has not learned well on training set. The best performed l1 penalty model on validation set is also experimented on test set, final accuracy is 0.837.

For L2 penalty with hyperparameter 1e-4 and 1e-3, better validation accuracy and smaller generalization gap are obtained compared to baseline model. L2 weight penalty is also a useful approach to address overfitting. The best performed l2 weight penalty model on validation set is also experimented on test set, final accuracy is 0.844.

For combined models, 6 combinations are experimented of varied dropout rate, L1/L2 penalty weights. Each hyper-parameter in combined experiments is selected to be the optimal in each subgroup. Generally, combined models performed well in validation set compared with baseline. One exception is 0.95, L1 1e-3 model. It performs bad and even worse than each individual experiment. This bad performance might caused by over-regularisation, which makes model become underfitting. Among these, the best model is configured as dropout rate 0.05 and L2 penalty hyper-parameter 1e-4. The final test performance evaluated on test set is 0.847. .

5. Literature Review: Maxout Networks

Summary of Maxout Networks In this section, we briefly discuss another generalization method: Maxout networks (Goodfellow et al., 2013). This paper further explores the dropout method and proposes a new "maxout" layer which can complement dropout. The authors evaluate the performance of Maxout Networks in four standard datasets, namely MNIST, CIFAR-10 and 100, and SVHN. They point out that although dropout has been widely applied in deep models, it should not be regarded as a minor performance enhancement that applicable for arbitrary models. Furthermore, the author argued its performance can be maximized when directly creating a model component which improves the ability of dropout as model averaging technique (Goodfellow et al., 2013) . Following this motivation, they propose the Maxout activation layers. These can be considered learnable activations that work as a universal convex function approximator. The Maxout layer first maps the hidden space to k subspaces through independent affine transformations, then, for each element in output vectors, it takes the maximum value across all subspaces.

Maxout model benefits most from dropout, and highly compatible with it as model averaging techniques. Maxout is designed to be used as activation function which can approximate arbitrary complex convex functions, and dropout can speed up the optimization process. The combination of these two yields state of art performance on four benchmark datasets. The author argued that with several reasons.These can be classified from two aspects: Model Averaging and Optimization, which have been discussed in sction 7 and 8.

- Dropout learns neuron network model averaging techniques more efficiently with maxout activation. Firstly, by training with dropout mechanism, Maxout units to get larger linear region and Maxout is more stable during dropout mask changing.In addition, compared to other activation function, maxout has less curvature, which improves accuracy.Experiment results shows that the dropout is indeed doing better model averaging as expected with the assistance of Maxout, and get greater performance than using Tanh activation.
- Secondly, maxout also improves dropout during training phrase.The author demonstrated that maxout model is easier to optimize than Relu with cross-channel pooling when training with dropout.

.

Strengths and limitations The author proposed a novel neural activation unit that further exploits the dropout technique. The general performance of proposed Maxout model is evaluated on four benchmark datasets, detailed settings including hyperparameter, training epochs are also given.In each dataset, designed model with maxout activations is tested, which obtains the best result compared to other state of art methods. The results that maxout performs best on all four benchmark datasets gives a convincing evidence for their effectiveness of maxout unit. Furthermore, in order to understand how it works in depth, experiments to test Model averaging and Optimization also conducted with designed control group in Section 7 and 8, results further reveal the advantages of Maxout with dropout in two aspects,verifying its validity .

Although the Maxout activation units can maximize the averaging effect of dropout in a deep architecture, we can argue that the Maxout computation is expensive. The advantage of dropout lies in its high scalability and computational advantages. It can be arbitrarily applied to various network structures, and the calculation speed is fast, which is very suitable for heavy computing algorithms such as training and inference of neural networks. In comparison, the design of the Maxout network needs to project the hidden vector into k subspaces. Both the forward algorithm and the backward algorithm of dropout can be calculated in $O(D)$ complexity, but the complexity of Maxout is $O(kD)$. This can lead to increasing the number of training epochs needed to reach convergence. Furthermore, the universal approximation property of Maxout seems powerful, but it would be interesting to verify that it is useful in practice. Specifically, we can design an experiment where we increase the number of subspaces k and see where performances stop improving. In extreme cases, it is even possible that the function learned is too specific to the training data, effectively causing overfitting.

6. Conclusion

In this report, we demonstrate how overfitting problems occurs with different neural network settings and discusses how dropout and L1/L2 weight penalty methods to address it. We find that increasing width or length of neuron network will lead to more severe overfitting problem. Dropout and L1/L2 weight penalty are showed to make contribution to mitigating this issue, and combination both methods has better performance. We observe that validation performance of the best model is with "dropout rate 0.05 and l2 penalty parameter 1e-4". This model has final test accuracy 0.847 on test set. Finally, we briefly review another method, Maxout, proposed in 2013, and discuss how it works and its strengths and weaknesses. Inspired by that paper, a future research direction might be to further explore the potentials of exiting methods by introducing new structures, just like what Goodfellow et al.(Goodfellow et al., 2013) did with Maxout unit .

References

Bishop, Christopher M. *Pattern Recognition and Machine Learning*. Springer, 2006. https://www.microsoft.com/en-us/research/people/cmbishop/prml-book/.

Goodfellow, Ian, Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. In *International conference on machine learning*, pp. 1319–1327. PMLR, 2013.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Ng, Andrew Y. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the twenty-*

first international conference on Machine learning, pp. 78, 2004.

Rosasco, Lorenzo. Sparsity based regularization. https://www.mit.edu/~9.520/spring09/Classes/class11_sparsity.pdf, 2009.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958, 2014.