Machine Learning Practical: Coursework 2

Christopher Sipola (s1667278)

24th November 2016

Contents

1 The combination of L1 and L2 regularization

This section will investigate whether combining L1 and L2 regularization offers any advantage over using either individually when trained on the MNIST data set using a three-layer model. The experiment will apply L1 and L2 regularization to *different* layers within the model to determine the effect of penalty order within the forward propagation process. Note that this differs notably from the more conventional approach of using both regularization methods on the same layer.[1]

1.1 Model architecture, hyperparameters and initialization settings

The model architecture for this experiment had three affine layers, separated by ReLU layers, with 100 features in the hidden layers. Each affine layer was penalized separately (see next section).

[1] An example of this is "elastic net," where both penalization terms are included in one cost function. See MLPR lecture notes http://www.inf.ed.ac.uk/teaching/courses/mlpr/2016/notes/w10a_sparsity_and_L1.html

Many of the initialization settings were taken from Exercise 3 of lab 5 ("Training with regularisation") because they produced desirable results,[2] which is discussed briefly in the next section. The experiment used a learning rate of 0.01, a momentum learning rate with a coefficient of 0.9, a uniform Glorot initialization with a multiplicative factor of 0.5 to set the initial weights, a constant initialization to set the initial bias, multi-class cross entropy error with Softmax applied to outputs, and a batch size of 50, and 100 epochs.

1.2 Description of experiment

To avoid overcomplicating the experiment, only the best performing L1 and L2 weight penalties were chosen from the results of lab 5 to test on the three affine layers. These two weight penalties, along with no weight penalty, gave three values to test:

- no penalty
- L1 penalty with regularization constant 10^{-5}
- L2 penalty with regularization constant 10^{-4}

Every combination of these weight penalties was tested on the three layers, for 27 experiments in total.

The set with no regularization was considered the baseline model (figure 1). The three affine layers for this model were not regularized. The training and validation errors have the hallmark shape of overfitting described in the last paragraph.

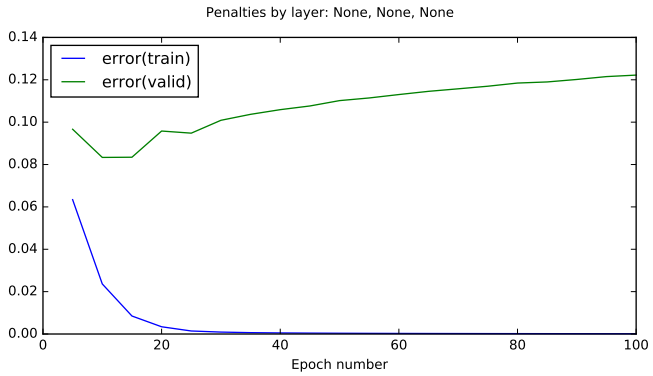


Figure 1: Baseline model.

[2]This paper assumes familiarity with the `mlp` module and the course labs.

1.3 Results and comments

The model with L2 regularization for all three layers outperformed all other models when using validation error as the metric for model performance (table 1). Unlike the baseline model, the validation error continued to decrease after epoch 10 (figure 2).

Layer 1	Layer 2	Layer 3	Validation error
L2	L2	L2	0.0757
L1	L2	L2	0.0776
L1	L2	L1	0.0785
L2	L1	L2	0.0795
L2	L2	L1	0.0796
L1	L2	None	0.0805
L1	None	L2	0.0812
L2	None	L2	0.0820
L2	L1	None	0.0823
L2	L2	None	0.0823
L1	L1	L1	0.0826
L1	L1	L2	0.0836
L1	L1	None	0.0842
L2	L1	L1	0.0843
L2	None	L1	0.0862
None	L2	L2	0.0865
L2	None	None	0.0884
L1	None	None	0.0901
None	L1	L2	0.0905
None	L2	L1	0.0917
None	L2	None	0.0964
None	None	L2	0.0992
None	L1	None	0.1019
None	L1	L1	0.1027
L1	None	L1	0.1135
None	None	L1	0.1149
None	None	None	0.1222

Table 1: The regularization models are ranked and ordered by their validation error.

It seems that the order of the models when sorted by validation error is not as "interesting" as was hoped for—at least not consistently across the results. That is, it seems that there is a preference for L2 over L1 regularization, and a preference for L1 over no regularization, with no consistent exception to this pattern. A possible way to explore the interaction effect between weight penalties of two different layers would be through a statistical analysis that tests whether the interaction effects of L1 and L2 regularization between these layers on validation error is higher or lower than the interaction effects of L2 and L2 (or L1 and L1) interaction effects.

However, it does seem plausible that the first layer has disproportionate influence on the validation

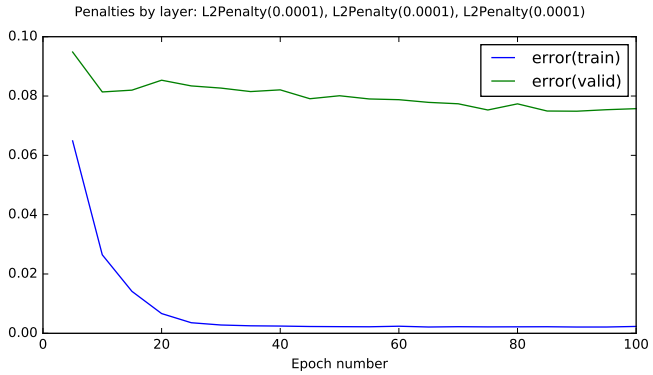


Figure 2: Best-performing penalty model.

error given the results. Notably, many of the models that performed worst had no regularization in the first layer. In other words, when the second and third layers were regularized, the error did not seem to decrease as much when the first layer was regularized. Furthermore, when comparing models where L2 regularization was applied to each layer separately, we see that the model where the first layer was regularized had the lowest final validation error due to greater regularization effects (figure 3).

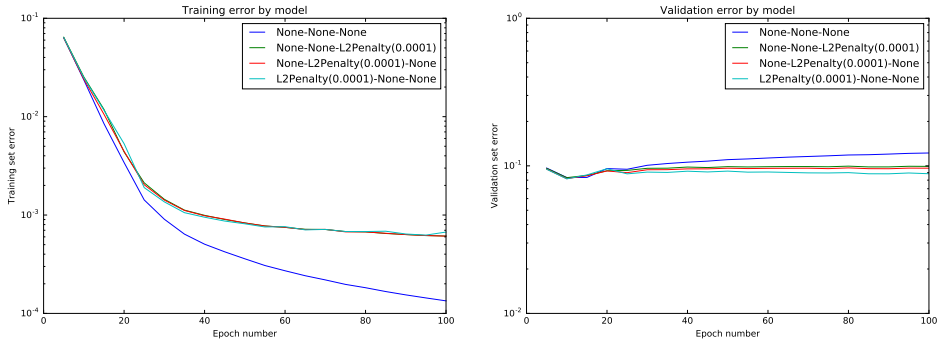


Figure 3: The influence of L2 regularization on each layer separately. The models are labeled using the format: *[layer 1 penalty]-[layer 2 penalty]-[layer 3 penalty]*. Note that *None-None-None* represents the baseline model.

It should be stated that these results may be specific to the particular architecture of the models. Additionally, the relationship between the penalties applied to each layer and the validation error may be complex and not easily demonstrated using the results and metrics presented in this paper.

2 Data augmentation: Gaussian filter

When a 2-dimensional Gaussian filter

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

is applied to the pixel data of an image, it produces a smoothing or blurring effect.[3] This section will explore whether this is an effective form of data augmentation when modeling the MNIST data. It will also explore how the following hyperparameters influence validation error: (1) the standard deviation σ for the Gaussian kernel;[4] and (2) the proportion of input images π_{aug} affected by the filter.

To simulate varying degrees of blurriness, the Gaussian filter function chooses σ from a uniform distribution $\sigma \sim \text{Uniform}(0, \sigma_{max})$, where σ_{max} is the maximum possible value or upper bound on the uniform distribution.[5] The motivation behind varying π_{aug} is that varying σ_{max} actually adjusts two implicit hyperparameters that we may want to tune separately: (1) the proportion of images that are sufficiently blurred and (2) the maximum blurriness. For example, when testing whether it is useful to blur a high proportion of images, we increase σ_{max}—which in turn may result in some images that are so blurry that they are ineffective at training our model and offset any potential benefits from increasing the proportion of blurred images. Therefore, introducing π_{aug} as another hyperparameter affords flexibility and helps to avoid this trade-off.

2.1 Model architecture, hyperparameters and initialization settings

The model architecture for this experiment had three affine layers, separated by ReLU layers, with 100 features in the hidden layers.

Similarly to the last section on L1 and L2 regularization, many of the initialization settings for this section were taken from lab 5. Specifically, the experiment used a learning rate of 0.01, a momentum learning rate with a coefficient of 0.9, a uniform Glorot initialization with a multiplicative factor of 0.5 to set the initial weights, a constant initialization to set the initial bias, multi-class cross entropy error with Softmax applied to outputs, a batch size of 100, and 100 epochs.

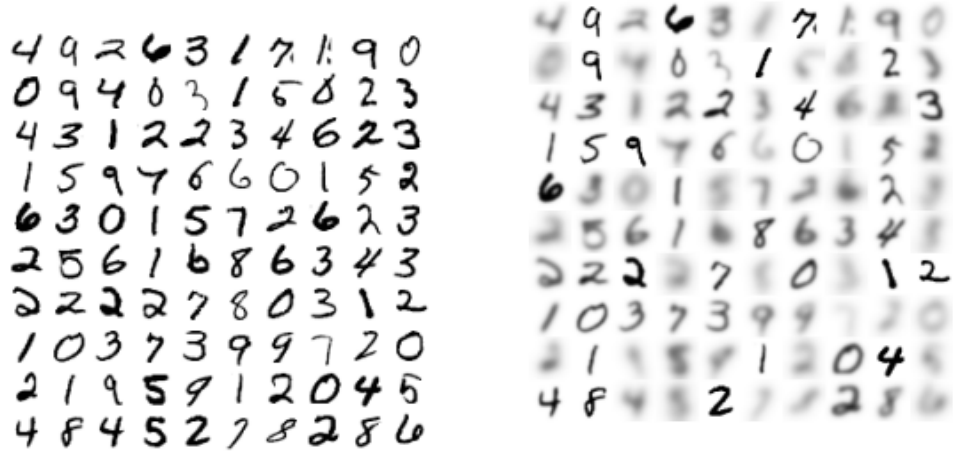
2.2 Description of experiment

During pilot runs, models with values $\pi_{aug} > 0.50$ took longer to compute, so the decision was made to exclude these—unless the results showed that the best performance was at $\pi_{aug} = 0.50$ in which case the search would be expanded. The blurring parameter σ produced noticeable blurring at around $\sigma = \frac{1}{2}$ and blurring beyond recognition at around $\sigma = 4$ (figure 4b).

[3] http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

[4] More accurately, this experiment will vary the upper bound on a distribution that dictates how σ will be distributed across altered images. This is discussed in detail later.

[5] Note that $\sigma = 0$ results in no blurring.



(a) MNIST targets without Gaussian filter.

(b) MNIST targets with Gaussian filter. Here, blurring parameter σ was chosen randomly for the 100 digits as $\sigma \sim \text{Uniform}(0, 4)$.

Figure 4: Blurring effects of the Gaussian filter.

With this in mind, the values $\sigma_{max} = \{\frac{1}{2}, 1, 2, 4, 8\}$ and $\pi_{aug} = \{0.00, 0.17, 0.25, 0.33, 0.50\}$ were chosen for the grid search. Values of π_{aug} were chosen to be close to 0.25 because this value produced reasonable results in the lab. All models where exactly one of σ_{max} and π_{aug} were equal to zero were not trained, since these would have produced no data augmentation and therefore would have been redundant to the baseline model (where both parameters were set to zero).

2.3 Results and comments

The model with $\sigma_{max} = 1$ and $\pi_{aug} = \frac{1}{3}$ achieved the lowest final validation error of 0.0929 (figure 5). This means a small to moderate amount of blurring on a significant portion of the data was helpful to training the model.

However, this model—and other models that performed well—did not seem to achieve this performance through greatly increased robustness and regularization. Evidence of this is that the validation error curves were shifted downward rather than flattened (figure 6).

Unsurprisingly, when the Gaussian filter has too strong a blurring effect ($\sigma_{max} \geq 4$), the model performs poorly. But more interestingly, the final validation error does not vary smoothly across values of π_{aug} when holding σ_{max} constant. For example, when increasing values of π_{aug} while setting $\sigma_{max} = 1$, the validation error starts low at 0.0970, increases to 0.1087, decreases to 0.0929, then increases to 0.1039. This pattern holds for all cuts of σ_{max} except for one.

Validation error

σ_{max} \ π_{aug}	0.00	0.17	0.25	0.33	0.50
0.0	0.1077				
0.5		0.1038	0.1071	0.1046	0.1071
1.0		0.0970	0.1087	0.0929	0.1039
2.0		0.0946	0.1045	0.0999	0.1034
4.0		0.1189	0.1260	0.1316	0.1302
8.0		0.1380	0.1483	0.1444	0.1469

Figure 5: Validation error of Gaussian filter models.

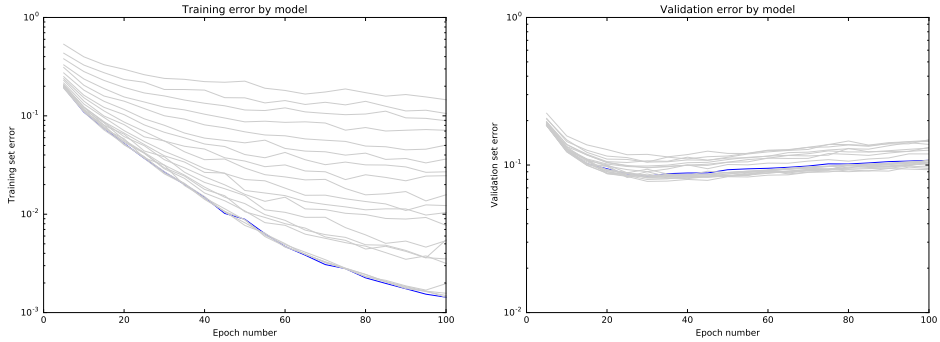


Figure 6: All models using Gaussian filters (gray) when compared against the baseline model (blue).

3 Batch normalization

Batch normalization is a technique that fixes the means and variance of layer inputs in order to decrease internal covariate shift [1]. This experiment will test some of the claimed benefits of using batch normalization, including regularization and improved speed of convergence, by using a three-layer model on the MNIST data set.

Additionally, it will also explore the effects of tweaking the parameters β and γ of the batch normalization function

$$z_i = \text{batchNorm}(u_i) = \gamma_i \hat{u}_i + \beta_i$$

where $\hat{u}_i$ is the normalized[6] activation for the layer.

[6]Here, normalized means having a mean of zero and a variance of 1. This is achieved for each parameter by subtracting off the mean and dividing by the standard deviation.

3.1 Model architecture, hyperparameters and initialization settings

The model architecture for this experiment had three affine layers, separated by ReLU layers, with 100 features in the hidden layers. For all models that are not the baseline model, a batch normalization layer was placed after the ReLU layer. More clearly, the layers in order were: affine, batch normalization, ReLU, affine, batch normalization, ReLU, and affine.

The experiment used a learning rate of 0.001, a momentum learning rate with a coefficient of 0.9, a uniform Glorot initialization with a multiplicative factor of 0.5 to set the initial weights, a constant initialization to set the initial bias, multi-class cross entropy error with Softmax applied to outputs, a batch size of 50, and 100 epochs.

3.2 Description of experiment

A slow learning rate of 0.001 was chosen for this experiment because this caused the validation error curve in the baseline model to decrease steadily across the 100 epochs (figure 7)—as opposed to reaching a minimum and then increasing due to overfitting (see baseline from the section on L1 and L2 regularization). This allows us view the effects of any increase in the speed of convergence in the early epochs.

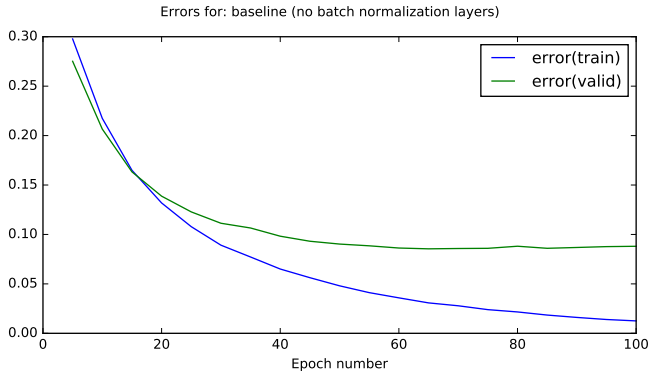


Figure 7: Baseline model.

The recommended initial values of the batch normalization parameters are $\gamma = 1$ and $\beta = 0$.[7] Not wanting to stray too far from zero, this experiment uses combinations of the values $\gamma = \{\frac{1}{9}, \frac{1}{3}, 1\}$ and $\beta = \{0, \frac{1}{9}, \frac{1}{3}, 1\}$ in the grid search.[8] $\gamma = 0$ was excluded as this would set weights to zero. Note that the goal of this "search" is not to optimize any particular value; rather, it is just to produce a reasonable assortment of models for comparing the aspects mentioned earlier.

[7] http://cthorey.github.io./backpropagation/

[8] Searching by factors of 3 is recommended by Dr Andrew Ng's Coursera course on machine learning: https://www.coursera.org/learn/machine-learning

3.3 Results and comments

Convergence is much faster for most of the batch normalization models (figure 8). The model with $\gamma = 1$ and $\beta = 0$ performed particularly well, reaching a validation error of 0.088 at epoch 20. In contrast, the baseline model reaches an error rate of 0.086 at epoch 65. This is a small increase in error for a large decrease in computation time. Regularization effects are also quite impressive in that the validation errors do not increase dramatically after hitting their lowest validation error value.

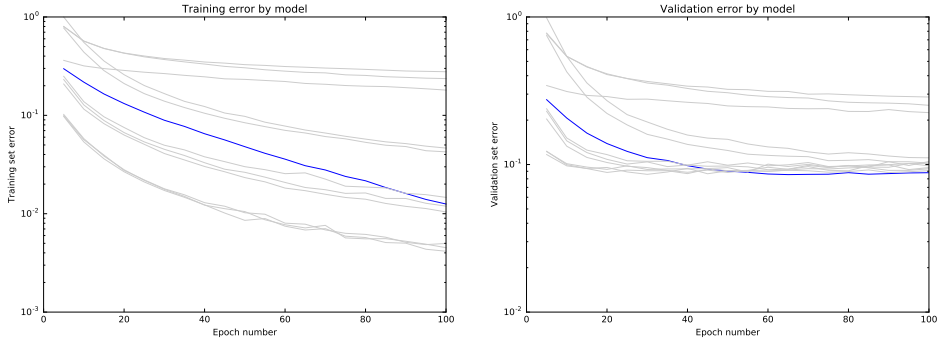


Figure 8: All models with batch normalization (gray) when compared against the baseline (blue).

We see little change when varying β across the values we chose (figure 9). The error curves for each choice of γ, however, were quite different, with some models performing better and some performing worse than our baseline model (figure 10).

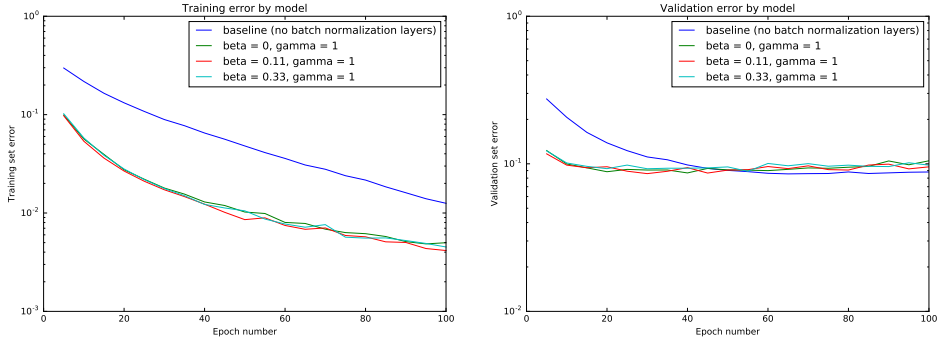


Figure 9: Training and validation errors when varying β and setting γ to its best-performing value.

Overall, batch normalization seems to produce large benefits in decreased computation time at the cost of just slight increases in error. Given more time, higher and more varied values of β would have been tested, as well as larger values of γ.

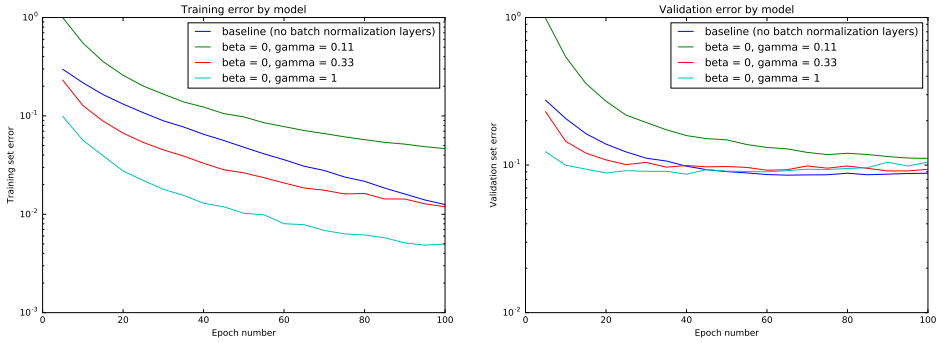


Figure 10: Training and validation errors when varying γ and setting β to its best-performing value.

References

[1] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.