Vanilla Substitutes

Assignment 2

ENGG 6500 | Intro to Machine Learning

Dr. Graham Taylor

Jesse Knight

2015-11-02

1 Theory

1.1 Model Training Trends

It is important to match the machine learning model to the task at hand; this prevents overfitting or underfitting of a model in supervised learning algorithms. Typically, the complexity of the system is unknown. However, we can develop an intuition by examining the performance of a validation set of data, while training using the training data (Figure 1.1). The complexity at which the validation data error increases is an estimate of the complexity required to represent the system.

Similarly, the performance of the discriminant will be impacted by the number of training cases observed: with very few cases, the model performs well on these cases, but fail to generalize and perform well on the validation data. Increasing the number of training examples will reduce the generalization error – which is the true goal – while losing performance in the training data (Figure 1.2). There is almost always a persistent positive gap between the validation error and training error.

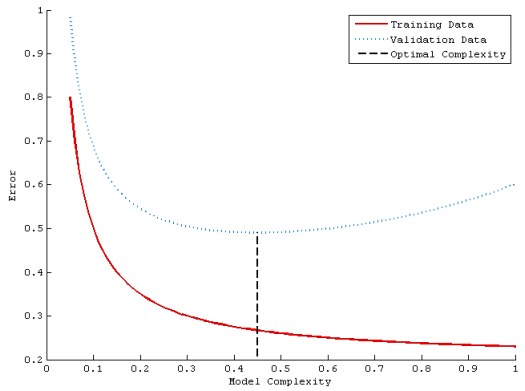

Figure 1.1: Performance of a model as the capacity increases, given a large enough training set

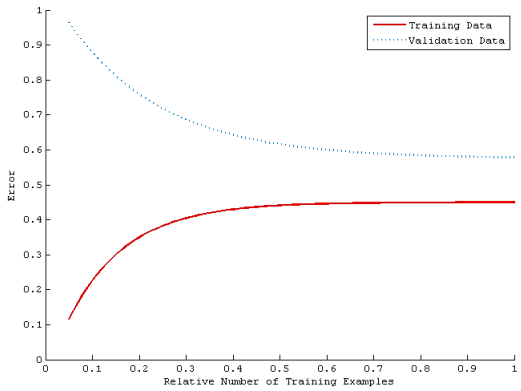

Figure 1.2: Performance of a model as the number of training examples increases, given optimal capacity

1.2 Weighted Least Squares Regression

While least squares regression is not a particularly flexible solution model, it is applicable to certain classes of problems. In some cases we may wish to normalize the estimated solution with respect to the variance or some other factor in our observations. To do this, we introduce a weight term for each data point. The least squares solution is then computed as follows:

$$E(w,b) = \sum_{i=1}^{N} r_i\left(y_i - (wx_i + b)\right)^2 \tag{1.1}$$

$$= \sum_{i=1}^{N} r_i(y_i - wx_i - b)\,(y_i - wx_i - b)$$

$$= \sum_{i=1}^{N} r_i({y_i}^2 + w^2x_i^2 + b^2 - 2y_ib - 2y_iwx_i + 2wx_ib)$$

$$\frac{\partial E}{\partial w} = \sum_{i=1}^{N} r_i(2wx_i^2 - 2y_ix_i + 2x_ib) \qquad \frac{\partial E}{\partial b} = \sum_{i=1}^{N} r_i(2b - 2y_i + 2wx_i)$$

$$0 = 2\sum_{i=1}^{N} r_i(w{x_i}^2 - y_ix_i + x_ib) \qquad 0 = 2\sum_{i=1}^{N} r_i(b - y_i + wx_i)$$

$$0 = b\sum_{i=1}^{N} r_ix_i + w\sum_{i=1}^{N} r_i\,{x_i}^2 - \sum_{i=1}^{N} r_iy_ix_i \qquad 0 = b\sum_{i=1}^{N} r_i + w\sum_{i=1}^{N} r_i\,x_i - \sum_{i=1}^{N} r_iy_i$$

$$w = \frac{\sum_{i=1}^{N} r_iy_ix_i - b\sum_{i=1}^{N} r_ix_i}{\sum_{i=1}^{N} r_i\,{x_i}^2} \qquad b = \frac{\sum_{i=1}^{N} r_iy_i - w\sum_{i=1}^{N} r_i\,x_i}{\sum_{i=1}^{N} r_i} \tag{1.2}$$

$$b\sum_{i=1}^{N} r_ix_i + w\sum_{i=1}^{N} r_i\,{x_i}^2 = \sum_{i=1}^{N} r_iy_ix_i \qquad b\sum_{i=1}^{N} r_i + w\sum_{i=1}^{N} r_i\,x_i = \sum_{i=1}^{N} r_iy_i \tag{1.3}$$

At this point it is simple, but unwieldly, to substitute either equation from (1.2) into the other to yield a closed form expression for the solution. Alternatively, we suggest solving the system of 2x2 linear equations presented in (1.3).

When considering optimization problems using the log-likelihood cost function, it should be noted that these models are equivalent. Moreover, in this model, the variance of each measurement is equal to the reciprocal of the weight term we employ here: ${\sigma_i}^2 = \frac{1}{r_i}$.

1.3 Classification Data Separability

A wide variety of classification algorithms exist for machine learning. Selecting the correct classification model for the data is often the most important choice the designer will make. For instance, here we consider two example datasets with two classes, and two features for each data point, Figure 1.3.

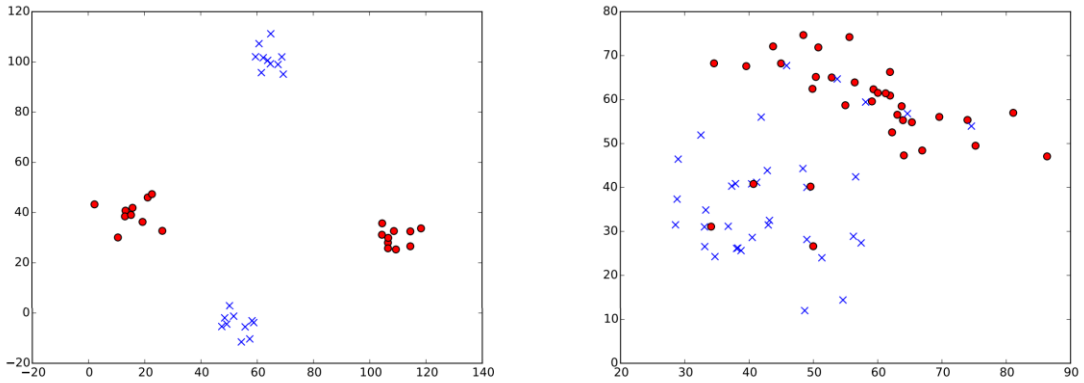

Figure 1.3: Two two-class two-feature datasets: a) Dataset 1; b) Dataset 2

We also consider two learning algorithms, Nearest Neighbour (NN), and Logarithmic Regression (LR). Interestingly, these two algorithms represent the extrema of discriminant nonlinear capacity. That is, LR solutions are always linear – a hyperplane in ND space. Conversely, NN solutions may be arbitrarily nonlinear – for a task of separating two 180° phase-shifted sinusoids, NN will perform well with enough training data. Alternatively, we can say that LR represents extreme nonlinear underfitting, while NN represents overfitting. The performance of the algorithms on these datasets is examined in Table 1.1.

Table 1.1: Comparison of algorithm performance on the datasets: Logarithmic Regression, and K=1 and K=5 NN

	Dataset 1	Dataset 2
LR	LR in 2D represents a linear discriminant, yet these data are not linearly separable (LS). A single line drawn anywhere through this space can achieve at best 75% accuracy, specifically by excluding one cluster. LR is a very poor choice here.	LR would perform well on this data set, as the two classes form roughly one cluster each. Assuming the overlap of the classes is a feature of the data which cannot be overcome, LR will misclassify data in the region where the other class dominates, but is essentially desirable.
1NN	1NN will perform well here because, while the data are not LS, there is no overlap of data along apparent class boundaries. The decision surface will converge on a rough X-shape, and since the data variance appears low, this will almost always classify the data correctly.	Class overlap, which we have here, is the major downfall of 1NN learning. A data point may be deep in 'territory' of one class, but the single nearest neighbour may be an outlier from the other class. This essentially doubles the impact of the outliers, as new outliers are classified incorrectly, and representative data are also misclassified due to the situation described above. 1NN is not a good choice here.
5NN	5NN will perform equally well to 1NN for this data. The class clusters are far enough away from one another, and there are enough data points per cluster, that there will be 5 neighbours of the correct class in every case that there would have been one. The only ambiguity is along the central crosshair, which is the region of highest uncertainty in any case.	Increasing the number of considered neighbours overcomes the overlap problem of 1NN. In fact, the solution here would be very similar to that of LR. The major difference will be bending of the discriminant towards the edges (e.g. the top-left most data from each class are not perpendicular to a LR discriminant). 5NN is a good choice here.

2 Practice

2.1 Data Preparation

Data pre-processing is an important step in algorithm training which is often overlooked. The script used in this project, `prep_data.py` parses a raw form of the data into data python structures, but also performs some important data manipulation. The workflow is roughly:

1. Load the raw data from two `tSV` files – `train.tsv` (training data) and `test.tsv` (test data).
2. Parse the data into a set of data structures for the training features and labels.
3. Augment some of the raw data to sparse ('one-hot'); select a subset of some of the text features.
4. Randomly split the training data into training (7000 cases) and validation sets.
5. Normalize the data: subtract the mean and divide by the standard deviation (feature data only).
6. Save the workspace as a pickle file (`.pkl`) – a unique structure for variable storage in `python`.

It was noted that the random seed for randomizing the ordering of the data is actually hardcoded. Depending on the random generator, this may not be actually providing different results each instance. Similarly, the test set it consistently unique. These settings may be useful during model development for consistency, but should be truly randomized for performance analysis of the final model.

2.2 Latent Dirichlet Allocation

The LDA algorithm is a method for feature extraction for documents; it estimates latent topics based on words in the document and describes each document as a mixture of topics. The topics are a distributed representation of any possible document, and words are a distributed representation of a topic.

Here, we are using LDA with webpage text. However, we note that the topics identified in this type of text are not always consistent with the conventional concept of a topic. For instance, the first output below (`14`) suggest that the concept of a number has been confounded with some other probably correlated words, like `pm` from time specifications and `comment` and `username`, possibly from discussion board pages. Similarly, the next output (`30`) suggests that some markup tags have been clustered into a topic, which is not a conventional notion, but possibly a valid one for our task, which will be discussed in more detail in Section 2.3.

```
14 10, 09, pm, 06, 02, 05, comment, 00, username, 99, 50, 20, 95, 11, 40
30 url, width, background, div, text, height, left, flickr, color, margin, font, position, block, twitter, inner
```

Many topics, however, are very reasonable, such as those shown below, including the university student diet (`12` – although `'dog'` would hopefully be exclusively preceded by `'hot'`):

```
12 bacon, hot, chicken, salad, grilled, pizza, cheese, dog, sauce, pork, meat, fried, grill, dogs, pasta
16 fashion, look, style, like, hair, love, just, dress, shoes, buzz, wear, looks, 01, 03, women
17 dough, baking, oven, minutes, flour, place, bake, sheet, roll, bowl, butter, sugar, make, cup, cut
25 sports, game, just, games, world, time, team, like, best, year, olympics, play, football, sport, athletes
```

The majority of the topics identified were food-related. Some other popular topics were sports, news, and technology. The prevalence of food topics may suggest that the websites investigated are largely blog pages, which might actually be representative of the internet, as user-created content can be created much faster than professional websites which focus on news or technology. However, this may also be a result of the low variance in the types of words in food-related content, relative to more complex topics like politics or academic topics.

Overall, while the topics identified might not fit the paradigm of an English language topic, the other topics like numbers or markups might actually be relevant to the classification problem we will attempt. Additionally, many topics identified were a little 'noisy', in terms of a few strange words joining the list.

2.3 Vanilla Network Optimization

Our machine learning task here is to predict the endurance of webpages – distinguish some sites which are high quality and continually relevant from those which are undesirable or passing trends. Using the LDA results, we have expanded our representation of each website to 66 features from a previous set of 36 numerical features, such as linking word ratios and number or ratio of markup tags. Here, we continue to consider stochastic gradient descent (SGD) learning for training a neural network with a single hidden layer – the Vanilla Network. We now discuss the impact of this augmented representation on the network performance in addition to the impact of some hyper parameters.

2.3.1 Convergence

The first observation we make is that convergence speed of the network has drastically decreased. We might imagine that the addition of the new features has complicated the decision space, and reason that the topic features require more complex non-linear relationships to be useful. However, the error rate which we converge onto improves significantly. In a controlled comparison (holding all other hyperparameters equal) the network quickly saturates with only 36 numerical features (Figure 2.1), while with all 66 features our error continues to decrease well past the this point (Figure 2.2). It was also observed that the additional features also consistently increase the initial error, also attributable to increased model size.

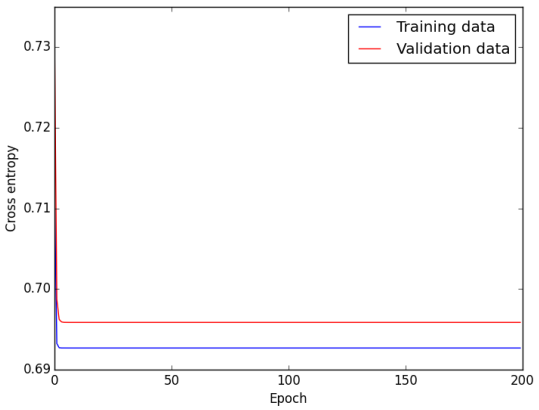

Figure 2.1: Mini-batch SGD error (validation set cross entropy) using only the 36 numerical features.

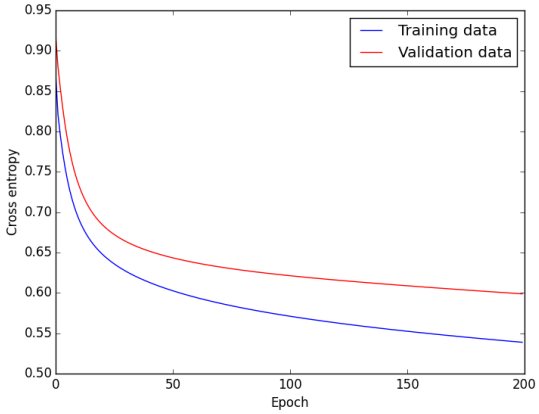

Figure 2.2: Mini-batch SGD error (validation set cross entropy) using all 66 numerical and topic features.

This evidence loosely suggests that increasingly complex feature spaces require increased convergence time, but are desirable for their improved performance. Conveniently, this motivates the employment of convergence-speeding techniques, as will be examined in Section 2.4.

2.3.2 Hyperparameter Optimization

We also observe that, with the new features, the range of hyperparameters which yield an apparently optimal error rate has increased – that is, the model is less sensitive to hyperparameters. For instance, we can significantly increase the learning rate, relative to the 36-input model, and not encounter stability issues until approximately $\alpha \approx 0.5$. Trivially, increasing learning rate (before instability) requires fewer epochs; 100 training epochs with $\alpha = 0.25$ was sufficient to reach steady state in most cases.

Perhaps more interestingly, however, is the performance of the model using very few hidden units. We conducted a very fine grid search across number of hidden units and learning rate, covering as low as a single hidden unit, and found not only no loss in performance, but apparently improved performance for less than 10 hidden units! The results are shown in Figure 2.3. We used 200 epochs and a mini-batch size of 500; only a single training instance was used for each grid point as it was noted that the initialization bore negligible weight (no pun intended) on the optimized performance at 200 epochs.

The rationalization for the small number of hidden units result is that our LDA topic extraction has already computed some highly non-linear features of the webpages. This greatly reduces the capacity

requirements for our discriminant network. In fact, it suggests that a simple linear combination of these new features may be an adequate model for this binary classification problem (i.e. with a single sigmoidal hidden unit). It should be noted, however, that the convergence curve for very few hidden units is 'bumpy' – suggesting our local minima may not always be reliable in these cases. It is for this reason we suggest using at least 10 hidden units for this model.

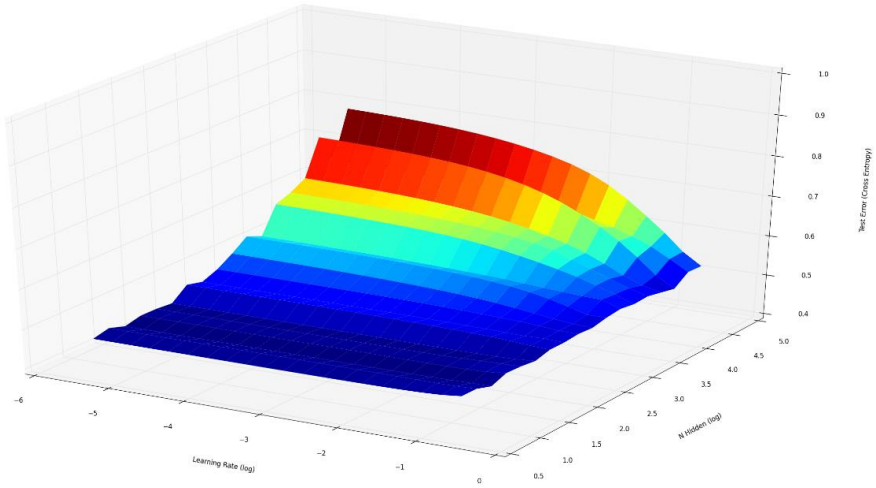

Figure 2.3: Impact of learning rate (log scale) and number of hidden units (log scale) on the test set cross entropy for fixed epoch number (200) and mini-batch size (500)

2.4 Speeding Convergence

As we noted above, the convergence speed of increasingly complex networks is slower. Fortunately, there are a variety of simple techniques used to speed convergence. We will discuss them here.

2.4.1 Early Stopping

A very general technique for ensuring validity of the convergence of a networks is to implement some criteria for stopping training if we suspect overfitting. The simplest criterion to identify the onset of overfitting is an increase in the error of the validation set over the previous two epochs. We note this addition to our algorithm here, as it becomes relevant when optimizing other hyperparameters related to the convergence of the network. That is, some of the other additions to the training method are liable to introduce overfitting.

2.4.2 Momentum

A very simple technique to actually speed convergence is called momentum; it targets plateaus and bouncing in long troughs. In essence, we are that applying a running average to the gradients with respect to each parameter. A common implementation is as follows, analogous to an infinite impulse response low-pass filter in signal processing:

$$\Delta w(t) = \mu\, \Delta w(t-1) - \frac{\partial\, J(t, y, x \mid \theta\,)}{\partial\, w} \tag{2.1}$$

$$w(t) = w(t-1) + \Delta w(t) \tag{2.2}$$

We found momentum to be highly successful in improving the convergence speed of the network, without impacting the convergence 'location' – as inferred by a consistent error value. Varying the momentum term, $\mu \in [0.10, 0.95]$, we found that increasing momentum was essentially always helpful. For 200 epochs, 20 hidden units, and a learning rate of 0.25, our training and validation error results are shown below:

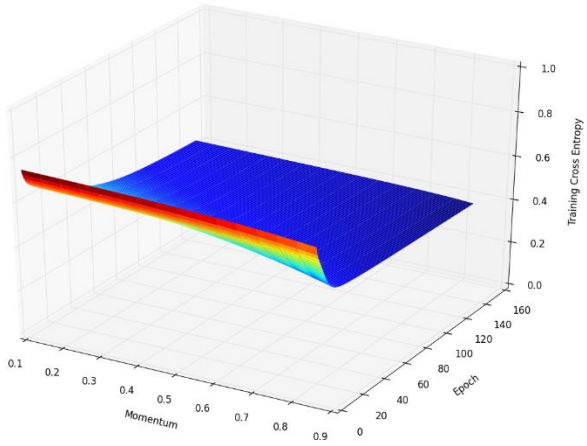

Figure 2.4:Training convergence with momentum

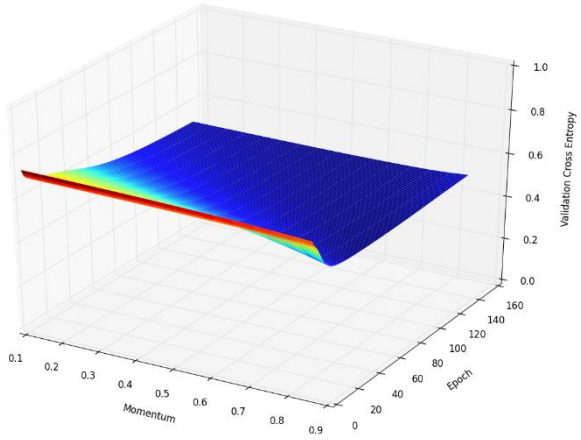

Figure 2.5: Validation convergence with momentum

For completeness, we should note that (2.2) explodes for $\mu \geq 1.0$, the results of which are evident in Figure 2.6. The network accelerates past the apparent solution around the 15th epoch and proceeds to 'bounce around' on in the feature space, unable to decrease speed except after overshooting the mark. This yields the periodic oscillations after the 80th epoch.

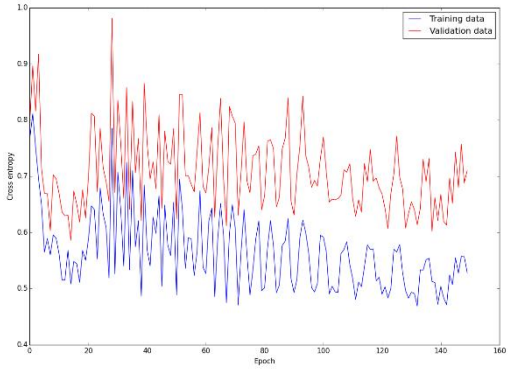

Figure 2.6: Demonstration of instability using momentum ≥ 1.0

As before, we note that exceedingly small numbers of hidden units still perform well, and converge very quickly. However, due to instability concerns, we recommend at least 10 units and $\mu \in [0.80, 0.95]$.

2.4.3 Adaptive Learning Rates

Another potential method for speeding up convergence is the calculation of adaptive learning rates (ALR) for each parameter in the network. Since we wish to learn most rapidly when our gradient consistently points in the same direction, we add a weight term g to the learning rate α. We increase g when the sign of the gradient for this specific parameter has been the same for the last two steps. We decrease g when the sign has changed. We use δ to denote the amount by which we vary g each step:

$$g_{i,j} = \left(1 + \delta \operatorname{sign}\left(\frac{\partial E}{\partial w_{i,j}}(t) \cdot \frac{\partial E}{\partial w_{i,j}}(t-1)\right)\right), \qquad \delta \in (0, 0.1] \tag{2.3}$$

$$\alpha_{i,j}(t) = g_{i,j} \cdot \alpha_{i,j}(t-1) \tag{2.4}$$

The addition of ALR to the model improved the convergence speed of the model in a manner similar to the momentum method. This is because they are both a method of increasing the impact of consistent gradient directions. The ALR method actually converges even faster, and is less sensitive to the choice of δ as momentum is for μ. However, we should note that the both methodologies parallel infinite impulse response filters, which are inherently unstable even for bounded inputs. A suggested improvement would be to clip the maximum and minimum values of α and Δw to prevent instability.

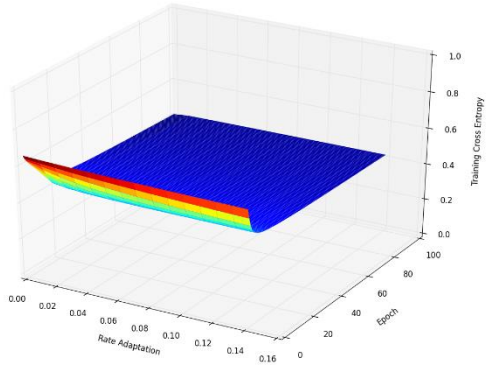

Figure 2.7: Validation data training error convergence acceleration with ALR

2.4.4 Combined Methods

Finally, we combined the two described methods, using high, though not-'optimal' hyper-parameters – this was done to avoid instability associated with the method combination. These are described in Table 2.1. Using this method, we compare the results from parameter matched training, below, as shown in, and note that early stopping is achieved after only 17 epochs.

Table 2.1: Optimal mini-batched stochastic gradient descent parameters using both momentum and ALR.

Parameter	Learning Rate	LR Adaptation	Momentum	N Hidden Units	Epochs
Value	0.25	0.01	0.80	20	50

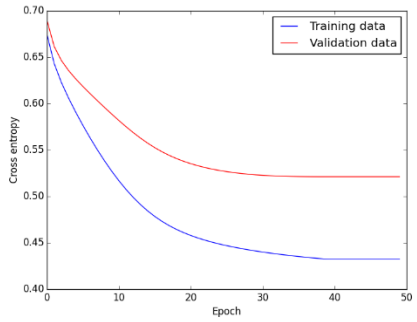

Figure 2.8: Slow error convergence without any SGD acceleration tools.

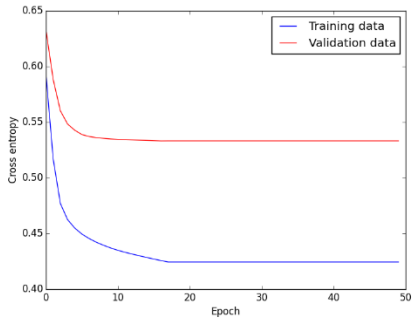

Figure 2.9: Accelerated descent of error using gradient momentum and ALR.