Fast Parametric Learning with Activation Memorization

Jack W Rae [1 2] **Chris Dyer** [1] **Peter Dayan** [3] **Timothy P Lillicrap** [1 2]

Abstract

Neural networks trained with backpropagation often struggle to identify classes that have been observed a small number of times. In applications where most class labels are rare, such as language modelling, this can become a performance bottleneck. One potential remedy is to augment the network with a fast-learning non-parametric model which stores recent activations and class labels into an external memory. We explore a simplified architecture where we treat a subset of the model parameters as fast memory stores. This can help retain information over longer time intervals than a traditional memory, and does not require additional space or compute. In the case of image classification, we display faster binding of novel classes on an Omniglot image curriculum task. We also show improved performance for word-based language models on news reports (GigaWord), books (Project Gutenberg) and Wikipedia articles (WikiText-103) — the latter achieving a state-of-the-art perplexity of 29.2.

1. Introduction

Neural networks can be trained to classify discrete outputs by appending a softmax output layer. This is a linear map projecting the d-dimensional hidden output of the network to m outputs, where m is the number of distinct classes. A softmax operator (Bridle, 1990) is then applied to produce a probability distribution over classes. The parameters in this softmax layer are typically optimized with the network's parameters by gradient descent.

We can think of the weights in the softmax layer $\theta \in \mathbb{R}^{m \times d}$ as a set of m vectors $\theta[i];\ i = 1, \ldots, m$ that each correspond to a given class. When trained with a supervised loss, such as cross-entropy, each step of gradient descent pulls the parameter $\theta[y]$, corresponding to the class label y, towards having a greater inner product with the network output h, and pushes all other parameters $\theta[j]$, $j \neq y$ towards having a smaller inner product with h.

One shortcoming of neural network classifiers trained with backpropagation is that they require many input examples for a given class in order to predict it with reasonable accuracy. That is, many positive class examples and optimization steps are required to pull $\theta[i]$ towards a point in space where class i can then be recognized. While the learner will have many opportunities to organize $\theta[i]$ parameters associated with frequent classes, infrequent class parameters will be poorly estimated. In domains where new classes are frequently introduced, or large-scale classification problems where some classes are very infrequently observed, this estimation problem is potentially quite serious.

One approach to speed up learning, which has received revived interest, is meta-learning. Here, meta-learning refers to algorithms which learn to produce or manipulate learning algorithms (Thrun, 1998; Hochreiter et al., 2001), and it operates by learning over a distribution of tasks or datasets. A meta-learner applies knowledge from the global distribution of tasks to produce or optimize algorithms which specialize to a given task instance. Meta-learning of neural networks has seen promising results for applications such as parameter optimization (Andrychowicz et al., 2016; Ravi & Larochelle, 2016; Finn et al., 2017) and classification (Santoro et al., 2016; Vinyals et al., 2016; Zhou et al., 2018). For classification, the networks are augmented with a differentiable external memory, and are trained with many rounds of data — with class labels permuted between episodes.

Meta-learning can be very powerful for few-shot learning in cases where there is a set of similar prior data to meta-learn over, however it may not be practical for standalone datasets. For example, if one wants to model the grammar of computer code, it is unclear that a meta-learning system trained over natural language will be useful. Also memory-based meta-learning requires backpropagating from the read time to the original write time, which is not well suited to applications where writes and reads are separated by many time steps. In the case of modelling language, for example,

[1] DeepMind, London, UK [2] CoMPLEX, Computer Science, University College London, London, UK [3] Gatsby Computational Neuroscience Unit, University College London, UK. Correspondence to: Jack W Rae <jwrae@google.com>, Timothy P Lillicrap <countzero@google.com>.

Proceedings of the 35^{th} *International Conference on Machine Learning*, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

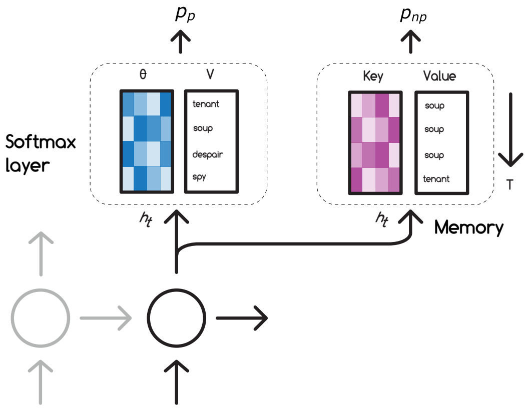

Figure 1. Mixture model of parametric and non-parametric classifiers connected to a recurrent language model. The non-parametric model (right hand side) stores a history of past activations and associated labels as key, value pairs. The parametric model (left hand side) contains learnable parameters θ for each class in the output vocabulary V. We can view both components as key, value memories — one slow-moving, optimized with gradient descent, and one rapidly updating but ephemeral.

infrequent words will not occur for large time intervals — rendering memory-based meta-learning challenging.

The task of statistical language modelling itself is interesting to investigate issues of binding new or infrequent classes, because most classes (words) are infrequent (Zipf, 1935) and new classes naturally emerge over time. Recent approaches to improve neural language models have involved augmenting the network with a non-parametric cache, which stores past hidden activations $h_{t-n}, \dots, h_{t-1}$ and corresponding labels, $y_{t-n}, \dots, y_{t-1}$ (Vinyals et al., 2015; Merity et al., 2016; Grave et al., 2016b; Kawakami et al., 2017; Grave et al., 2017). Attention over this cache provides better modelling of infrequent words that occur in a recent context, including previously unknown words (Gulcehre et al., 2016). However there is a diminishing return to increasing the cache size (Grave et al., 2016b), and once rare words fall outside the recent context the boost in predictive performance expires.

Motivated from these memory systems, we explore a very simple optimization procedure where the network accumulates activations h_t directly into the softmax layer weights $\theta[y_t]$ when a class y_t has been seen a small number of times, and uses gradient descent otherwise. Accumulating or smoothing network activations into the weights actually corresponds to the well-known Hebbian learning update rule $W[i,j] \leftarrow \frac{1}{n}\sum_{t=1}^{n} x_t^i x_t^j$ (Hebb, 1949) in the special case of classification on the output layer, where W, x_t^i, x_t^j correspond to θ, h_t, y_t respectively. We see that mixing the two rules provides better initial representations and can also preserve these representations for much longer time spans. This is because memorized activations for one class are not competing for space with activations from other (more frequent, say) classes — unlike a conventional external memory. In this sense, the parameters become an instance of a quickly updated compressed memory, we explore this idea in Section 3.2

We demonstrate this model adapts quickly to novel classes in a simple image classification task using handwritten characters from Omniglot (Lake et al., 2015). We then show it improves overall test perplexity for two medium-scale language modelling corpora, WikiText103 (wikipedia articles) from Merity et al. (2016) and Project Gutenberg[1] (books), alongside a large-scale corpus GigaWord v5 (news articles) from Parker et al. (2011). By splitting accuracy over word frequency buckets, we see improved perplexity for less frequent words.

2. Background

2.1. Memory

There has been recent interest in models which store past hidden activations through time $h_1, h_2, \dots, h_{t-1}$ into a memory matrix and query the contents with a differentiable attention mechanism. This has been applied to machine translation (Bahdanau et al., 2014), program induction (Graves et al., 2014; 2016), and question answering (Sukhbaatar et al., 2015). Memory-augmented neural networks have also been successfully applied to language modelling (Mikolov et al., 2010; Vinyals et al., 2015; Kawakami et al., 2017; Merity et al., 2016; Grave et al., 2016b; 2017) to facilitate the learning of unknown words, capture the tendency for globally rare words to be repeated in close proximity, and to quickly adapt the network to contextually relevant prior text (Sprechmann et al., 2018).

There are many variants of how to read from memory and mix this information with the network's activations. One approach is to retrieve hidden activations and mix these with network activations in latent space (Gulcehre et al., 2016). Another approach is a classic mixture model, as shown in Figure 1; the output probability distribution can be obtained by interpolating the probabilities p_p, p_{np} from the parametric model and memory respectively.

For intuition we briefly explain a particular architecture, the *Neural Cache* (Grave et al., 2016b), whose operation is related to our model. The cache is a store of the last n hidden activations along with their corresponding target outputs (next words) from a trained parametric language model, such as the Long Short Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997). The conditional probability of a

[1] Project Gutenberg. (n.d.). Retrieved January 2, 2018, from www.gutenberg.org

word w occurring is proportional to the sum over kernalized inner product similarities between the current hidden state h_t and past hidden states when word w occurred.

$$p_c(w \mid h_t) \propto \sum_{i=t-n}^{t-1} e^{h_t^T h_i}\, \mathbb{I}\{y_i = w\} \qquad (1)$$

Where $\mathbb{I}\{p\} = 1$ if p is true, 0 otherwise. This is then interpolated with the parametric language model using a fixed hyper-parameter, swept over during validation. Although the cache is of fixed size n, it can be defined to be very large with sparse attention and efficient data-structures (Rae et al., 2016; Kaiser et al., 2017; Grave et al., 2017).

2.2. Language modelling

We can model a sequence of text as the product of conditional word probabilities,

$$p(w_1, w_2, \ldots, w_t) = \prod_{i=1}^{t} p(w_i \mid w_1, w_2, \ldots, w_{i-1})$$

which are estimated separately. Traditional n-gram models take frequency-based estimates of these conditional probabilities with truncated contexts $p_n = p(w_i \mid w_{i-n}, \ldots, w_{i-1})$ and smooth between them to estimate the full conditional probability, $p(w_i \mid w_1, \ldots, w_{i-1}) = \sum_{j=1}^{n} \lambda_j p_j$. A popular approach is Kneser-Ney smoothing (Kneser & Ney, 1995). More recently, neural language models such as LSTMs and convolutional neural networks directly model the conditional probabilities through sequence-to-sequence training and achieve state-of-the-art performance in many established benchmarks (Collobert & Weston, 2008; Sundermeyer et al., 2012; Kalchbrenner et al., 2014; Jozefowicz et al., 2016; Dauphin et al., 2016; Melis et al., 2017).

3. Model

We propose the Hebbian Softmax, a modification of the traditional softmax layer with an updated learning rule. The Hebbian Softmax contains the same linear map from the hidden state to the output vocabulary, but learns by smoothing hidden activations into the weight parameters for novel classes whilst concurrently applying gradient descent. This is to facilitate faster binding of novel classes, and improve learning of infrequent classes. We note this corresponds to a learning rule that transitions from Hebbian learning to gradient descent, and we will show that the combination of the two learning rules works better than either one in isolation.

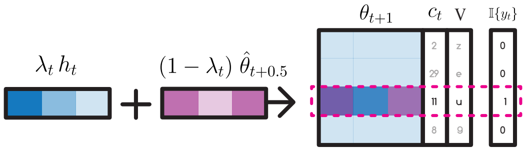

Figure 2. Update rule. Here the vector $\hat{\theta}_{t+0.5}$ denotes the parameters $\theta_t[y_t]$ of the final layer softmax corresponding to the active class y_t after one step of gradient descent. This is interpolated with the hidden activation at the time of class occurrence, h_t. The remaining parameters are optimized with gradient descent. Here, $\mathbb{I}\{y_t\}$ is the one-hot target vector, V denotes the vocabulary of classes, and c_t is defined to be a counter of class occurrences during training — which is used to anneal λ_t as described in (4).

Many of the features of the Hebbian Softmax are motivated from memory systems, and the theory of complementary learning systems in the brain (McClelland et al., 1995). During training, the weights corresponding to a given class will initially correspond to a compressed[2] episodic memory store — with new activations memorized and older activations eventually forgotten.

The parameters of the softmax layer are treated both as regular slow-adapting network parameters through which gradients flow to the rest of the network, and fast-adapting memory slots which are updated sparsely without altering the rest of the network. In comparison to an external memory, the advantage of Hebbian Softmax is that it is simple to implement and requires almost no additional space or computation.

We will describe the learning rule in detail, and contrast the conditional probabilities from Hebbian Softmax to those generated by a non-parametric cache. We also generalize the memorization procedure in Section 3.3 as an instance of a secondary fast-learning overfitting procedure with respect to a euclidean objective, and explore several promising variant objective functions.

3.1. Update Rule

Given the weights of a linear projection $\theta \in \mathbb{R}^{d \times m}$ in the final softmax layer of a network, we calculate the gradient descent update with respect to a cross-entropy loss,

$$\hat{\theta}_{t+0.5}[i] \leftarrow \begin{cases} \theta_t[i] - \alpha\,(p_i - 1)\,h_t & i = y_t \\ \theta_t[i] - \alpha\, p_i\, h_t & i \neq y_t \end{cases} \qquad (2)$$

where $p_i = e^{h_t^T \theta_i} / \sum_{j=1}^{n} e^{h_t^T \theta_j}$ is the probability output from the softmax, and α is the learning rate. In practice the gradient descent update $\hat{\theta}_{t+0.5}$ can be calculated with adaptive optimizers, such as RMSProp (Tieleman & Hinton, 2012). This is interpolated with the previous layer's hidden

[2] The memory is denoted 'compressed' because multiple activations corresponding to the same class are smoothed into one vector, instead of being stored separately.

activation h_t for the active class y_t,

$$\theta_{t+1}[i] \leftarrow \begin{cases} \lambda_t \, h_t + (1-\lambda_t) \, \hat{\theta}_{t+0.5}[i] & i = y_t \\ \hat{\theta}_{t+0.5}[i] & i \neq y_t \ , \end{cases} \tag{3}$$

as illustrated in Figure 2. When $\lambda_t = 1$ this corresponds to the rule $\theta_{t+1} \leftarrow h_t \cdot \mathbb{I}\{y_t\}$ where $\mathbb{I}\{y_t\} \in [0,1]^m$ is a one-hot target vector. In this case Hebbian update rule, $W_{ij} \leftarrow x_i x_j$ for $x_i = h_t$ the hidden output and $x_j = \mathbb{I}\{y_t\}$ the target. Naturally when $\lambda = 0$ this is gradient descent, and so we see Hebbian Softmax is mixture of the two learning rules. All remaining parameters in the model are optimized with gradient descent as usual.

When mixing the two learning rules, we would like to benefit from fast initial learning of classes that have not been seen many times, along with stable consolidation of frequently seen classes. As such we do not want λ_t to be constant, but instead something that is eventually annealed to zero. We add an additional counter array $\mathbf{c} \in \mathbb{Z}^m$ which counts class occurrences, and propose an annealing function of

$$\lambda_t = \max(1 \, / \, \mathbf{c}[y_t], \ \gamma) \ \cdot \ \mathbb{I}\{\mathbf{c}[y_t] < T\} \tag{4}$$

where γ, T are tuning parameters. T is the number of class occurrences before switching completely to gradient descent and γ is the minimum activation mixing parameter. Although heuristic, we found this worked well in practice vs. a constant λ or pure annealing $\lambda_t = 1/c[y_t]$. If training from scratch, we suggest setting $\gamma = 1/N_{min}$ and $T = N_{min} \times$ (# epochs until convergence) where N_{min} is the minimum number of occurrences of any class in a training epoch. This is to ensure we smooth over many class examples in a given epoch, and the memorization of activations continues until the representation of h_t stabilizes. We describe the full algorithm in Algorithm 1, including details for training with minibatches.

The final layer trains with a two-speed dynamic. For some training steps the full network will be optimized slowly via gradient descent as usual (when frequently-encountered classes are observed), and for other time steps a sparse subset of parameters will rapidly change. The remaining network parameters are optimized with gradient descent.

It is worth noting that simply increasing the learning rate of the softmax layer, or running multiple steps of optimization on rare class inputs, would not achieve the same effect. The value $\theta[y_t]$ would indeed be pulled towards a large inner product with h_t, however neighbouring parameters $\theta[i]$; $i \neq y_t$ would be pushed towards a large negative inner product with h_t and this could lead to catastrophic forgetting of previously consolidated classes. Instead we allow gradient descent to slowly push neighbouring parameters away, and thus disambiguate similar classes in a gradual fashion.

Algorithm 1 Hebbian Softmax batched update

```
— At iteration 0
$\gamma \leftarrow$ min. discount (hyper-parameter)
$T \leftarrow$ smoothing limit (hyper-parameter)
$M \leftarrow$ num. classes
$B \leftarrow$ batch size
$\mathbf{c}_0[i] \leftarrow 0; \quad i = 1, \ldots, M$
— At iteration $t$
$\mathbf{h}_{t,1:B} \leftarrow$ softmax inputs
$\mathbf{p}_{t,1:B} \leftarrow$ softmax outputs
$\mathbf{y}_{t,1:B} \leftarrow$ target labels
$\hat{\theta}_{t+0.5} \leftarrow \mathtt{SGD}(\theta_t, \mathbf{h}_{t,1:B}, \mathbf{p}_{t,1:B}, \mathbf{y}_{1:B})$
for $i = 1, \ldots, M$ do
  $n_{t,i} \leftarrow \sum_{j=1}^{B} \mathbb{I}\{y_{t,j} = i\}$
  if $n_{t,i} > 0$ then
    $\lambda_{t,i} \leftarrow \max(1/\mathbf{c}_t[i], \gamma) \, \mathbb{I}\{\mathbf{c}_t[i] < T\}$
    $\bar{h}_{t,i} \leftarrow \frac{1}{n_{t,i}} \sum_{j=1}^{B} h_{t,j} \mathbb{I}\{y_{t,j} = i\}$
    $\theta_{t+1} \leftarrow \lambda_{t,i} \bar{h}_{t,i} + (1 - \lambda_{t,i}) \hat{\theta}_{t+0.5}[i]$
  else
    $\theta_{t+1} \leftarrow \hat{\theta}_{t+0.5}[i]$
  end if
  $\mathbf{c}_{t+1}[i] \leftarrow \mathbf{c}_t[i] + n_{t,i}$
end for
```

3.2. Relation to cache models

We can consider the weights constructed from the above optimization procedure as a compressed memory, storing historic activations. We contrast the output probabilities of Hebbian Softmax with those produced from a non-parametric cache model.

Recall the conditional probability of a class, w, given a cache of previous activations (1). If we set $I_w(j)$ to be the time step of j-th most recent occurrence of w, then we can re-write the cache probability,

$$\begin{aligned} p_c(w \mid h_t) &\propto \sum_{i=t-n}^{t-1} e^{h_t^T h_i} \mathbb{I}\{y_i = w\} \\ &= \sum_{j=1}^{N_w} e^{g(j) \, h_t^T h_{I_w(j)}} \end{aligned} \tag{5}$$

where $g(j) = -\infty$ if $j < t-n$ and 1 otherwise, is a weighting function which places uniform weight to the attention over classes in the past n time steps. However if we wish to characterize infrequent classes, we may want a weighting scheme with a larger time horizon that has a smooth decay.

If we modified the cache to have infinite memory capacity and used a geometric weighting scheme to decay the contribution of the j-th most recent activation corresponding to the given class, e.g. $g(j) = \lambda \, (1 - \lambda)^{j-1}$, then the resulting conditional probability is,

$$\tilde{p}_c(w \mid h_t) \propto \sum_{j=1}^{N_w} e^{\lambda\,(1-\lambda)^{j-1}\, h_t^T h_{I_w(j)}} \tag{6}$$

where N_w is the total number of occurrences of class w. Let us now consider the conditional probability from Hebbian Softmax for class w, where w has been observed less than T times. If θ has not received large gradients from the occurrence of nearby neighboring classes, and we fix $\lambda_t = \lambda$ over time, then (3) gives

$$\theta_i \approx \sum_{j=1}^{N_w} \lambda\,(1-\lambda)^{j-1} h_{I_w(j)}\ ,$$

plugging this into our softmax conditional probability,

$$\begin{aligned} p_\theta(w \mid h_t) \propto e^{h_t^T \theta_w} &\approx e^{h_t^T \sum_{j=1}^{N_w} \lambda\,(1-\lambda)^{j-1} h_{I_w(j)}} \\ &= \prod_{j=1}^{N_w} e^{\lambda\,(1-\lambda)^{j-1} h_t^T h_{I_w(j)}}\ . \end{aligned}$$

we see the parametric Hebbian Softmax actually becomes a proxy for the conditional probability output by the non-parametric infinite cache model $\tilde{p}_c$. Past activations now have a geometric contribution to the probability, versus the cache's arithmetic reduction (6). This form is useful because we can compute p_{sm} much more efficiently than $\tilde{p}_c$ and it does not require storing the entire history of past activations.

3.3. Alternate Objective Functions

We briefly discuss a generalization of the Hebbian Softmax update by casting it as an overfitting procedure to an inner objective function. Recall equation (3) for parameters corresponding to the active class,

$$\theta_{t+1}[i] \leftarrow \lambda_t\, h_t + (1-\lambda_t)\, \hat{\theta}_{t+0.5}[i].$$

We can re-phrase this as smoothing $\hat{\theta}_{t+0.5}[i]$ with the trivial solution to a euclidean objective function, which we overfit to.

$$\begin{aligned} &\theta_{t+1}[i] \leftarrow \lambda w^* + (1-\lambda)\, \hat{\theta}_{t+0.5}[i] \\ &w^* \leftarrow \arg\max_w -||w - h_t||_2 \end{aligned}$$

From this perspective we are performing a two-level optimization procedure. The outer optimization loop is the mixture of gradient descent and exponential smoothing, and the inner optimization loop determines a good value for w^* based on the activation h_t and the current parameters.

We consider several other objective functions that are more expensive to compute, but may be preferable to a simple Euclidean distance. Notably, switching to inner product similarity (`IP`), and also incorporating a cost to parameter similarity (`SVM, Smax`) to push w^* towards h_t but away from neighbouring parameters — to avoid confusion or interference with other classes. As we keep neighbouring parameters fixed, we hope to avoid the catastrophic forgetting typically associated with model overfitting. We list the set of objectives considered,

$$w^* \leftarrow \arg\max_w\ g(w)$$

$$g_{\text{L2}}(w) = -||w - h_t||_2 \tag{7}$$

$$g_{\text{IP}}(w) = w^T h_t \tag{8}$$

$$g_{\text{SVM}}(w) = w^T h_t - \sum_{\theta_j \in \mathcal{N}_k(h_t)} \xi\, w^T \theta_j\ \cdot \mathbb{I}(w^T \theta_j > \epsilon) \tag{9}$$

$$g_{\text{Smax}}(w) = e^{w^T h_t} / \sum_{\theta_j \in \mathcal{N}_k(h_t)} e^{w^T \theta_j} \tag{10}$$

where $\mathcal{N}_k(h_t)$ refers to the k nearest parameters to the activation h_t that do not correspond to y_t, the class label. Including all M parameters in θ_t would make the inner optimization loop very slow, so we choose a sparse subset $k \ll M$. These are all optimized under the hard norm constraint $||w||_2 < 10$ with gradient descent for multiple steps, typically 20, at a given point in training.

4. Results

4.1. Image Curriculum

We apply Hebbian Softmax to the problem of image classification. We create a simple curriculum task using Omniglot data (Lake et al., 2015), where a subset of classes (30) are initially provided, and 5 new classes are added when test performance exceeds a threshold (60%). Although this is a toy setup, it allows us to investigate the basic properties of fast class binding without other confounding factors, found in real-world problems.

Omniglot contains handwritten characters from 50 alphabets, totalling 1623 unique character classes. There are 20 examples per class. We partition the first 5 examples per class to a test set, and assign the rest for training.

We use the same architectural setup as *Matching Networks* (Vinyals et al., 2016) where the images are re-sized to 28×28 and a 4 layer convolutional neural network is used. Each layer has 64 filters, 3×3 convolutions, batch normalization, ReLU activations, and 2×2 max pooling. Each channel maps the input to a scalar, so the resulting hidden size is 64. All weight parameter in the softmax are initialized with Glorot initialization (Glorot & Bengio, 2010). Models were trained with 20% dropout on the final layer and a small amount of data augmentation was applied to training examples (rotation $\in [-30, 30]$, translation) to avoid overfitting. Otherwise the models quickly plateau on a low level.

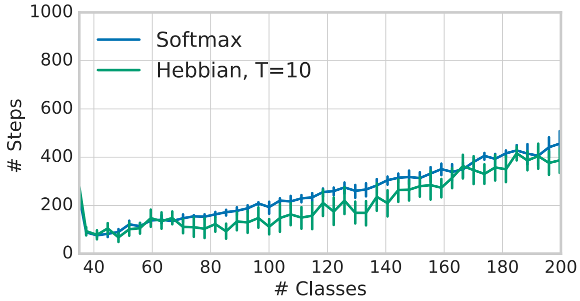

Figure 3. Number of training steps taken to complete each level on the Omniglot curriculum task. Comparisons between the Hebbian Softmax and softmax baseline are averaged over 10 independent seeds. As classes are sampled uniformly, we expect the number of steps taken to level completion to rise linearly with the number of classes.

For the Hebbian Softmax update, we store the pristine hidden activation pre-dropout. Unlike many one-shot Omniglot papers, we do not train in a meta-learning setup — namely, labels are not shuffled between episodes.

We trained the convnet classifier with RMSProp (Tieleman & Hinton, 2012), Adam (Kingma & Ba, 2014), and AdaGrad (Duchi et al., 2011). We swept over learning rates to find the fastest-learning baseline softmax model (see Figure 11 in Appendix B). We then compared the regular softmax layer with the Hebbian Softmax, both placed on top of the convnet encoder.

If we inspect the number of steps spent on each level averaged over 10 seeds, focusing on RMSProp for simplicity, we see in Figure 3 that the model is noticeably more data efficient after 80 total classes. In Appendix B, Figure 10 we see this faster curriculum progression is consistent across RMSProp, Adam, and AdaGrad. Although the models are far from one-shot, there is a $1 - 2X$ data efficiency gain on average.

4.2. Language Modelling

We would like to evaluate the Hebbian Softmax in the context of a large-scale classification task, where some classes are infrequently observed. Word-level language modelling is an ideal fit because it satisfies both criteria, and there are established performance benchmarks. Some large-scale language modelling corpora require the use of efficient softmax approximations, such as the adaptive softmax (Grave et al., 2016a) or hierarchical softmax (Goodman, 2001) due to the very large vocabulary size. To reduce confounding factors, we restrict ourselves to applications where the full softmax can be used. We investigate two medium-sized corpora, WikiText-103 which contains just over $100M$ tokens derived from Wikipedia articles (Merity et al., 2016), and Gutenberg which contains a subset of open-access texts from Project Gutenberg listed in Appendix A.3. The idea is that Wikipedia articles should cover factual information, where the style of writing is somewhat consistent and named entities may appear across many articles; whereas books should be more self-contained (unique named entities) and stylistically different. We also consider a very large corpus, GigaWord v5, which is a collection of articles from eight press associations exceeding a decade's worth of global news.

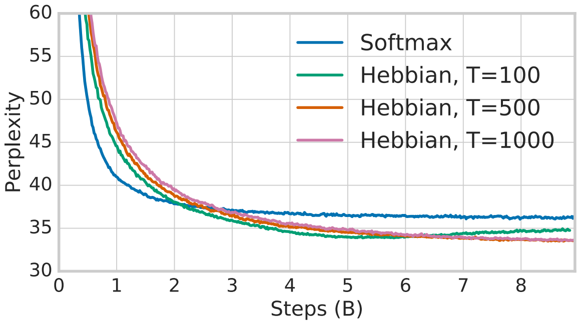

Figure 4. Validation perplexity for WikiText-103 over 9 billion words of training ($\approx$ 90 epochs). The LSTM drops to a perplexity of 36.4 with a regular softmax layer, and 34.3 with the Hebbian Softmax , $T = 500$, when representations from the LSTM begin to settle. For tuning parameter T; $T = 100$ converges quicker, but begins to overfit after 5.5B training words (coinciding when all classes have been observed at least 100 times).

We selected the baseline model to be a single-layer LSTM with 2048 units, tied input/output embedding parameters, and an embedding dropout rate of 0.3. These were selected from a baseline sweep on WikiText-103. Hyper-parameters and further training details are described in Appendix A.1.

4.2.1. WikiText-103

The WikiText-103 corpus contains $267,735$ unique words and each word occurs at least three times in the training set. We take the best LSTM parameter configuration (described above) as a baseline, and compare it to an identical model where the final layer is replaced with Hebbian Softmax . We swept over the insertion limit parameter $T \in \{100, 500, 1000\}$ and discount factor $\gamma \in \{0.05, 0.1, 0.25\}$ using the validation set. We found $T = 500$, $\gamma = 0.25$ worked best, achieving a test perplexity of 34.3 on this dataset (Table 1). Inspecting the validation curves in Figure 4 we see the Hebbian Softmax initially hampers validation performance, until around 2–3B training tokens have been consumed. This makes sense, as storing activations from prior layers of the network is only an effective strategy once the network has rich intermediate representations of its inputs. Table 2 shows the test perplexity broken down by word frequency, we see the gain in overall

Table 1. Validation and test perplexities on WikiText-103.

	Valid.	Test
LSTM (Grave et al., 2016b)	-	48.7
Temporal CNN (Bai et al., 2018)	-	45.2
Gated CNN (Dauphin et al., 2016)	-	37.2
LSTM (ours)	36.0	36.4
LSTM + Cache	34.5	34.8
LSTM + Hebbian	34.1	34.3
LSTM + Hebbian + Cache	29.7	29.9
LSTM + Hebbian + Cache + MbPA	**29.0**	**29.2**

Table 2. Test perplexity versus training word frequency. Hebbian Softmax models less frequent words with better accuracy. Note the training set size of WikiText is smaller than Gutenberg, which is itself much smaller than GigaWord; so the > 10K bucket includes an increasing number of unique words. This explains GigaWord's larger perplexity in this bucket. Furthermore there were no words observed $<$ 100 times within the GigaWord 250K vocabulary. A random model would have a perplexity of $|V| \approx 2.5e5$ for all frequency buckets.

	> 10K	1K-10K	100-1K	< 100	ALL
WIKITEXT-103					
SOFTMAX	12.1	2.2E2	1.2E3	9.7E3	36.4
HEBBIAN SOFTMAX	12.1	1.8E2	7.6E2	5.2E3	34.3
GUTENBERG					
SOFTMAX	19.0	9.8E2	6.9E3	8.6E4	47.9
HEBBIAN SOFTMAX	18.1	9.4E2	6.6E3	5.9E4	45.5
GIGAWORD					
SOFTMAX	39.4	6.5E3	3.7E4	-	53.5
HEBBIAN SOFTMAX	33.2	3.2E3	1.6E4	-	43.7

performance is obtained from less frequent vocabulary.

We also investigate the model evaluated dynamically on the test using (a) a Neural Cache (Grave et al., 2016b) and (b) Memory-based Parameter Adaptation (MbPA) (Sprechmann et al., 2018). Hyper-parameter details for these models are detailed in Appendix A.2. The cache reduces the test perplexity by 1.6 for the LSTM and 4.4 for LSTM + Hebbian Softmax . The addition of MbPA reaches a test perplexity of 29.2 which is, to the authors' knowledge, state-of-the-art at time of writing.

4.2.2. GUTENBERG

Books provide several different linguistic challenges to articles. The style of writing is intentionally varied between authors, and named entities can be wholly fictional — confined to a single text. We extract a subset of English-language books from the corpus, strip the Gutenberg headers and tokenize the text (Appendix A.3.2). We select a dataset of comparable size to WikiText-103; 2042 books in total with 2017 training books ($175,181,505$ tokens), 12 validation books ($609,545$ tokens), and 13 test books ($526,646$ tokens) — see Appendix A.3 for full details. We select all words that occur at least five times in the training set, a total vocabulary of $242,621$ and map the remainder to an unk token.

We use the same LSTM hyper-parameters as those chosen from the wikipedia sweep, and compare against Hebbian Softmax with $T = 100,\ T = 500$ and $\gamma = 0.1$. Figure 5 in Appendix A.3 shows the validation performance after 15B steps of training, equating to roughly 80 epochs and 6 days of training with 8 P100s training synchronously. After approximately 4B steps of training the softmax performance is surpassed, and this gap widens even up to 15B steps to a gap of 2-3 points in perplexity. Similar to WikiText-103, we see in Table 2 the gain in perplexity is more pronounced over less frequent words.

4.2.3. GIGAWORD V5

We evaluate Hebbian Softmax on a large-scale language modelling corpus. GigaWord is interesting because it is a vast collection of news articles, and there is a natural temporal order. We pre-process the dataset (Appendix A.3.2), select all articles from 2000-2009 for the training set, and test on all articles from 2010. The total number of training tokens is 4.0B and the total number of test tokens is 260M. The total unique tokens (after pre-processing) for the training set reaches 6M, however for parity with the other experiments we choose a vocabulary size of 250K. We use the same LSTM hyper-parameters and Hebbian Softmax hyper-parameters, and train the model for 6B steps, after which the models plateau in evaluation performance. We observe a 9.8-point drop in perplexity, from 53.5 to 43.7, illustrated in Table 2.

4.3. Softmax Approximations

So far we have always used the full softmax as a baseline. This is to make experimental comparisons straightforward, however in many applications the full softmax is too expensive to compute. We now consider the interaction between the Hebbian Softmax update rule and computationally efficient softmax approximations, namely the *sampled softmax* (Jean et al., 2014). When the baseline language model is trained on WikiText-103 with a sampled softmax (using 8192 samples) we see in Appendix A.5.2, Figure 8 that the learning update from Hebbian Softmax improves upon the sampled softmax by approximately 2 perplexity points, however both models plateau $2-3$ perplexity points higher than the exact softmax models from Section 4.2.1.

4.4. Alternate Objective Functions

We test out some of the alternate inner objective functions described in (7) from Section 3.3. The inner objective functions include *Euclidean, Inner Product, SVM, (sparse) Softmax*. These could be applied to any of the described experiments, we chose the WikiText-103 language modelling task because it is more comparable to prior work.

Although more expressive objective functions appear promising, in practice we find that validation performance is roughly equivalent between all inner objective functions (Figure 9 in Appendix A.5.3). This suggests the network activation h_t naturally do not land too close to other class parameters, and the norm of activations is not too large or small, in comparison to the model parameters θ. The latter may be due to the use of layer normalization from the LSTM.

5. Related Work

Few-shot classification has been investigated in a meta-learning setup with a mixture model of a parametric neural network and a non-parametric memory (Santoro et al., 2016; Vinyals et al., 2016). Here, a subset of classes are used with permuted labels per episode, activations are stored to memory, and gradients are passed through the memory. This allows the network to shape its activations to be conducive to accurate retrieval and classification. In this study we do not meta-learn the activations stored into network parameters and instead rely on their representation being rich enough from regular training. We do this to avoid backpropagating through time to the point of memory write, which is impractical when memories are stored millions of time steps ago, such as in the case of modelling rare words.

In natural language processing memory-augmented models have been shown to improve the modelling of unknown words and adaptation to new domains (Grave et al., 2016b; Merity et al., 2016; Kawakami et al., 2017). However in these works the memory is typically small and models the recent past. During evaluation the test activations and corresponding labels are stored in memory, and the model is evaluated dynamically — adapting to the test data on the fly. Whilst dynamic evaluation provides insights into domain transfer, it is limited in applicability as the model may not receive ground-truth labels when launched into production.

More recent work has investigated methods of memorizing and searching over the training set to enhance performance (Kaiser et al., 2017; Grave et al., 2017; Gu et al., 2017). These approaches typically require complex engineering to efficiently index this memory store. Part of the benefit of the Hebbian Softmax is implementation simplicity.

Prior literature on the softmax operator for language modelling computational efficiency (Chen et al., 2015; Grave et al., 2016a) or tricks such as smoothing across many softmax layers (Yang et al., 2017). However these do not focus on increasing the data-efficiency or faster learning of infrequent classes.

Other architectures have been considered for fast learning, such as the 'fast weights' auto-associative memory (Ba et al., 2016a). This focuses on fast adaptation to recent information that persists over a short window of time. The LEABRA architecture (O'Reilly, 1996a) contains a mixture of contrastive Hebbian learning (GENEREC) (O'Reilly, 1996b) and gradient descent for fast and slow learning, however this cognitively-inspired model has not been shown to scale to large-scale classification problems.

6. Discussion

This paper explores one way in which we can achieve fast parametric learning in neural networks, and preserve this knowledge over time. We show that activation memorization is useful for vision in the binding of newly introduced classes, beating well tuned adaptive learning rate optimizers, RMSProp and AdaGrad.

For language we show improvement in the modelling of text with an extensive vocabulary. In the latter we show the model beats a very strong LSTM benchmark on three stylistically different corpora, and achieves state of the art on WikiText-103. This is achieved with effectively no additional compute or memory resources. Breaking down perplexity over word frequency bucket, we see that less frequent words are better modelled, as hypothesized. We suggest that the Hebbian Softmax could be applied to any classification domain with infrequent classes, or non-stationary data. It may also be useful in quickly adapting a pre-trained classifier to a new task / set of classes — however this is beyond the scope of our initial investigation.

It would also be interesting to explore activation memorization deeper within the network, and thus in more general scenarios to classification. In this case, there is no direct feedback from a ground-truth class label and the update rule would not necessarily be an instance of Hebbian learning. A natural first step would be to generalize the ideas to large-scale softmax operators that are internal to the network — such as attention over a large memory.

Acknowledgements

The authors would like to thank Gabor Melis, Greg Wayne, Oriol Vinyals, Pablo Sprechmann, Siddhant Jayakumar, Charles Blundell, Koray Kavukcuoglu, Shakir Mohamed, Adam Santoro, Phil Blunsom, Felix Hill, Angeliki Lazaridou, James Martens, Ozlem Aslan, Guillaume Desjardins, and Chloe Hillier for their comments and assistance during the course of this project. Peter Dayan is currently on a leave of absence at Uber Technologies; Uber was not involved in this study.

References

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., and de Freitas, N. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pp. 3981–3989, 2016.

Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z., and Ionescu, C. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*, pp. 4331–4339, 2016a.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016b.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Bai, S., Kolter, J. Z., and Koltun, V. Convolutional sequence modeling revisited, 2018. URL https://openreview.net/forum?id=rk8wKk-R-.

Bridle, J. S. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *Advances in neural information processing systems*, pp. 211–217, 1990.

Chen, W., Grangier, D., and Auli, M. Strategies for training large vocabulary neural language models. *arXiv preprint arXiv:1512.04906*, 2015.

Collobert, R. and Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 160–167. ACM, 2008.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. Language modeling with gated convolutional networks. *arXiv preprint arXiv:1612.08083*, 2016.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

Goodman, J. Classes for fast maximum entropy training. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, volume 1, pp. 561–564. IEEE, 2001.

Grave, E., Joulin, A., Cissé, M., Grangier, D., and Jégou, H. Efficient softmax approximation for gpus. *arXiv preprint arXiv:1609.04309*, 2016a.

Grave, E., Joulin, A., and Usunier, N. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*, 2016b.

Grave, E., Cisse, M. M., and Joulin, A. Unbounded cache model for online language modeling with open vocabulary. In *Advances in Neural Information Processing Systems*, pp. 6044–6054, 2017.

Graves, A., Wayne, G., and Danihelka, I. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471, 2016.

Gu, J., Wang, Y., Cho, K., and Li, V. O. Search engine guided non-parametric neural machine translation. *arXiv preprint arXiv:1705.07267*, 2017.

Gulcehre, C., Ahn, S., Nallapati, R., Zhou, B., and Bengio, Y. Pointing the unknown words. *arXiv preprint arXiv:1603.08148*, 2016.

Hebb, D. O. The organization of behavior: A neurophysiological approach, 1949.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hochreiter, S., Younger, A. S., and Conwell, P. R. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pp. 87–94. Springer, 2001.

Jean, S., Cho, K., Memisevic, R., and Bengio, Y. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*, 2014.

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Kaiser, L., Nachum, O., Roy, A., and Bengio, S. Learning to remember rare events. *International Conference on Learning Representations*, 2017.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

Kawakami, K., Dyer, C., and Blunsom, P. Learning to create and reuse words in open-vocabulary neural language modeling. *arXiv preprint arXiv:1704.06986*, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kneser, R. and Ney, H. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pp. 181–184. IEEE, 1995.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

McClelland, J. L., McNaughton, B. L., and O'reilly, R. C. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

Melis, G., Dyer, C., and Blunsom, P. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

O'Reilly, R. C. Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural computation*, 8(5):895–938, 1996b.

O'Reilly, R. C. *The Leabra model of neural interactions and learning in the neocortex*. PhD thesis, PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1996a.

Parker, R., Graff, D., Kong, J., Chen, K., and Maeda, K. English gigaword fifth edition ldc2011t07. dvd. *Philadelphia: Linguistic Data Consortium*, 2011.

Rae, J., Hunt, J. J., Danihelka, I., Harley, T., Senior, A. W., Wayne, G., Graves, A., and Lillicrap, T. Scaling memory-augmented neural networks with sparse reads and writes. In *Advances in Neural Information Processing Systems*, pp. 3621–3629, 2016.

Ravi, S. and Larochelle, H. Optimization as a model for few-shot learning. *International Conference on Learning Representations*, 2016.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016.

Sprechmann, P., Jayakumar, S., Rae, W. J., Pritzel, A., Puigdomenech, A. B., Uria, B., Vinyals, O., Hassabis, D., Pascanu, R., and Blundell, C. Memory-based parameter adaptation. *International Conference on Learning Representations*, 2018.

Sukhbaatar, S., Weston, J., Fergus, R., et al. End-to-end memory networks. In *Advances in neural information processing systems*, pp. 2440–2448, 2015.

Sundermeyer, M., Schlüter, R., and Ney, H. Lstm neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

Thrun, S. Lifelong learning algorithms. In *Learning to learn*, pp. 181–209. Springer, 1998.

Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4 (2):26–31, 2012.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In *Advances in Neural Information Processing Systems*, pp. 2692–2700, 2015.

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pp. 3630–3638, 2016.

Yang, Z., Dai, Z., Salakhutdinov, R., and Cohen, W. W. Breaking the softmax bottleneck: a high-rank rnn language model. *arXiv preprint arXiv:1711.03953*, 2017.

Zhou, F., Wu, B., and Li, Z. Deep meta-learning: Learning to learn in the concept space. *arXiv preprint arXiv:1802.03596*, 2018.

Zipf, G. K. The psychology of language. *NY Houghton-Mifflin*, 1935.