Dynamic Evaluation of Neural Sequence Models

Ben Krause [1] **Emmanuel Kahembwe** [1] **Iain Murray** [1] **Steve Renals** [1]

Abstract

We explore dynamic evaluation, where sequence models are adapted to the recent sequence history using gradient descent, assigning higher probabilities to re-occurring sequential patterns. We develop a dynamic evaluation approach that outperforms existing adaptation approaches in our comparisons. We apply dynamic evaluation to outperform all previous word-level perplexities on the Penn Treebank and WikiText-2 datasets (achieving 51.1 and 44.3 respectively) and all previous character-level cross-entropies on the text8 and Hutter Prize datasets (achieving 1.19 bits/char and 1.08 bits/char respectively).

1. Introduction

Sequence generation and prediction tasks span many modes of data, ranging from audio and language modelling, to more general timeseries prediction tasks. Applications of such models include speech recognition, machine translation, dialogue generation, speech synthesis, forecasting, and music generation. Neural networks can be applied to these tasks by predicting sequence elements one-by-one, conditioning on the history, thus forming an autoregressive model. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs), including long-short term memory (LSTM) networks (Hochreiter & Schmidhuber, 1997) in particular, have achieved many successes at these tasks. However, in their basic form, these models have a limited ability to adapt to recently observed parts of a sequence.

Many sequences contain repetition; a pattern that occurs once is more likely to occur again. For instance, a word that occurs once in a document is much more likely to occur again. A sequence of handwriting will generally stay in the same handwriting style. A sequence of speech will generally stay in the same voice. Although RNNs have a hidden state that can summarize the recent past, they have been shown to have problems learning to reproduce sequence elements (Marcus, 2001; Prickett, 2017). In the case of RNN language modelling, augmenting a model with a simple unigram cache improves perplexity (Grave et al., 2017), demonstrating that RNNs have difficulty using the recent frequency of words in a sequence. Models such as pointer networks (Vinyals et al., 2015), copy nets (Gu et al., 2016), pointer-sentinel RNNs (Merity et al., 2017) and the neural cache method (Grave et al., 2017) allow inputs to be used directly as outputs, thus enabling them to more naturally handle "direct repetitions" in a sequence, where a symbol repeats itself. However, such approaches do not model "indirect repetitions", when a synonym or word related to a recently-occurring word appears. More broadly, it is desirable for an adaptive model to be able to capture deeper regularities such as topic or style.

This paper examines *dynamic evaluation* (Mikolov et al., 2010; Mikolov, 2012; Graves, 2013), which adapts models to recent sequences using gradient descent, as a way to model re-occurring sequential patterns. Previous work using dynamic evaluation did not explore or describe its methodology in depth, and had mixed results. In contrast, our work develops a method and tuning procedure to consistently obtain strong results (Section 5), and uses this approach to outperform previously reported results in word and character-level language modelling (Section 7). Furthermore, we design a method to dramatically reduce the number of adaptation parameters in dynamic evaluation, making it practical in a wider range of settings (Section 6). We analyse the performance of dynamic evaluation over varying time-scales and distribution shifts, and demonstrate that dynamically evaluated models can generate conditional samples that repeat many patterns from the conditioning data (Section 7.4). Finally, we show that dynamic evaluation can generalize to related words, giving it the potential to model indirect repetitions in sequences (Section 7.5).

2. Motivation

Generative models can assign a probability to a sequence $x_{1:T} = \{x_1, \ldots, x_T\}$ by factorizing it as

$$P(x_{1:T}) = P(x_1) \prod_{t=2}^{T} P(x_t | x_{1:t-1}). \quad (1)$$

[1]School of Informatics, University of Edinburgh. Correspondence to: Ben Krause <ben.krause@ed.ac.uk>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

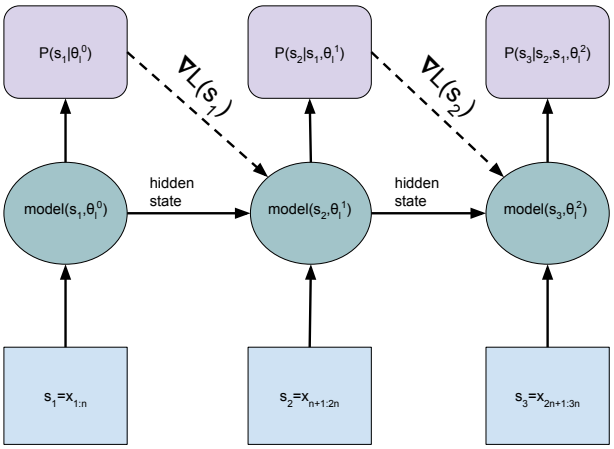


Figure 1. Illustration of dynamic evaluation. The model evaluates the probability of sequence segments s_i. The gradient $\nabla\mathcal{L}(s_i)$ with respect to the log probability of s_i is used to update the model parameters θ_l^{i-1} to θ_l^i before the model progresses to the next sequence segment. Dashed edges are what distinguish dynamic evaluation from static (normal) evaluation.

Methods that apply this factorization either use a fixed context when predicting $P(x_t|x_{1:t-1})$, or use a recurrent hidden state to summarize the context, as in an RNN. However, for longer sequences, the history $x_{1:t-1}$ often contains reoccurring patterns that are difficult to capture using static models with fixed parameters.

In a dataset of sequences $\{x^1_{1:T}, x^2_{1:T}, ..., x^M_{1:T}\}$, each sequence $x^i_{1:T}$ is typically generated from a slightly different distribution $P(x^i_{1:T})$. At any point in time t, the history of a sequence $x^i_{1:t-1}$ contains useful information about the generating distribution for that sequence. Therefore we aim to adapt the global model parameters θ_g learned during training, by inferring a set of local model parameters θ_l from $x^i_{1:t-1}$ that will better approximate $P(x^i_t|x^i_{1:t-1})$.

The generating distribution may change continuously across a single sequence; for instance, a text excerpt may change topic. Furthermore, many machine learning benchmarks do not distinguish between sequence boundaries, and concatenate all sequences into one continuous sequence. Thus, many sequence modelling tasks can be viewed as having a local distribution $P_l(x)$ as well as a global distribution $P_g(x) := \int P(l)P_l(x)\,dl$. When training, the goal is to find the best fixed model possible for $P_g(x)$. However, at evaluation, a model that can infer the current $P_l(x)$ from the recent history has an advantage.

3. Dynamic evaluation

Dynamic evaluation continuously adapts the model parameters learned at training time, θ_g, to parts of a sequence during evaluation. The goal is to learn adapted parameters θ_l that provide a better model of the local sequence distribution, $P_l(x)$. In this work, we apply dynamic evaluation by splitting a long test sequence $x_{1:T}$ into a sequence, $s_{1:M}$, of shorter sequence segments s_i of length n:

$$s_{1:M} = \{s_1 = x_{1:n},\ s_2 = x_{n+1:2n},\ ...,\ s_M\}. \qquad (2)$$

The initial adapted parameters θ_l^0 are set to θ_g, and used to compute the probability of the first segment, $P(s_1|\theta_l^0)$. This probability gives a cross entropy loss $\mathcal{L}(s_1)$, with gradient $\nabla\mathcal{L}(s_1)$, which is computed using truncated backpropagation through time (Werbos, 1990). The gradient $\nabla\mathcal{L}(s_1)$ is used to update the model, resulting in adapted parameters θ_l^1, before evaluating $P(s_2|\theta_l^1)$. The same procedure is then repeated for s_2, and for each s_i (Figure 1). Gradients for each loss $\mathcal{L}(s_i)$ are only backpropagated to the beginning of s_i, so the computation is linear in the sequence length. Each update applies one maximum likelihood training step to approximate the current local distribution $P_l(x)$. The computational cost of dynamic evaluation is thus one forward pass and one gradient computation through the data, with an additional small overhead to apply the update rule for every sequence segment.

As in all autoregressive models, dynamic evaluation only conditions on sequence elements that it has already predicted, and so evaluates a valid log-probability for each sequence. Dynamic evaluation can also be used while generating sequences. In this case, the model generates each sequence segment s_i using fixed weights, and performs a gradient descent based update step on $\mathcal{L}(s_i)$. Applying dynamic evaluation for sequence generation could result in generated sequences with more consistent regularities, meaning that patterns that occur in the generated sequence are more likely to occur again.

4. Related approaches

Adaptive language modelling was first considered for n-grams, adapting to recent history via caching (Kuhn, 1988; Jelinek et al., 1991), and other methods (Bellegarda, 2004). The neural cache approach (Grave et al., 2017) and the related pointer sentinel-LSTM (Merity et al., 2017) have been used for adaptive neural language modelling. Neural caching was recently used to improve the state-of-the-art at word-level language modelling (Merity et al., 2018).

The neural cache model learns a non-parametric output layer on the fly at test time, enabling the network to adapt to recent observations. Each past hidden state h_i is paired with the next input x_{i+1}, and stored as a tuple (h_i, x_{i+1}). When a new hidden state h_t is observed, the output probabilities are adjusted to give a higher weight to words that coincide with

past hidden states with a large inner product ($h_t^T h_i$):

$$P_{\text{cache}}(x_{t+1}|x_{1:t}, h_{1:t}) \propto \sum_{i=1}^{t-1} e^{(x_{i+1})} \exp(\omega h_t^T h_i), \quad (3)$$

where $e^{(x_{i+1})}$ is a one hot encoding of x_{i+1}, and ω is a scale parameter. Test time adaptation is carried out by interpolating the cache probabilities with the base network probabilities.

Both the neural cache and dynamic evaluation can be used to adapt a base model at test time. The main difference is the mechanism used to fit to recent history: the neural cache uses a non-parametric, nearest neighbours-like method, whereas dynamic evaluation uses gradient descent. Both methods rely on an autoregressive factorisation, as they depend on observing sequence elements after they are predicted in order to perform adaptation. Dynamic evaluation and neural caching methods are therefore both applicable to sequence prediction and generation tasks, but not directly to more general supervised learning tasks.

The neural cache has limited ability to capture information that occurs jointly between successive sequence elements, since it cannot adjust the recurrent hidden state dynamics. This capability is critical for adapting to sequences in which each element has very little independent meaning, e.g. character level language modelling. Additionally, the neural cache can only raise the probability of symbols it has previously seen in a test sequence, which could limit its generalization ability in word-level language modelling.

Mikolov et al. (2010) proposed dynamic evaluation of neural language models, with stochastic gradient descent (SGD) updates at every time step, computing the gradient with fully truncated backpropagation through time – equivalent to setting $n=1$ in (2). Dynamic evaluation has since been applied to character- and word-level language models (Graves, 2013; Krause et al., 2017; Ororbia II et al., 2017; Fortunato et al., 2017), although it has largely been considered as an aside and not explored in depth.

Dynamic evaluation as applied at test time, could be considered a form of fast weights (Schmidhuber, 1992; Ba et al., 2016) – recurrent architectures with dynamically changing weight matrices as a function of recent sequence history. In traditional fast weights, the network learns to control changes to the weights during training time, allowing it to be applied to more general sequence problems including sequence labeling. In dynamic evaluation, the procedure to change the weights is automated at test time via gradient descent, making it only directly applicable to autoregressive sequence modelling. As dynamic evaluation leverages gradient descent, it has the potential to generalize better to previously unseen pattern repetitions at test time.

5. Dynamic evaluation methodology

We propose several changes to Mikolov et al. (2010)'s dynamic evaluation update rule with SGD and fully truncated backpropagation, which we refer to as traditional dynamic evaluation. The first modification reduces the update frequency, so that gradients are backpropagated over more timesteps. This change provides more accurate gradient information, and also improves the computational efficiency of dynamic evaluation, since the update rule is applied much less often. We use sequence segments of length 5 for word-level tasks and 20 for character-level tasks.

Next, we add a global decay prior to bias the model towards the parameters θ_g learned during training. Our motivation for dynamic evaluation assumes that the local generating distribution $P_l(x)$ is constantly changing, so it is potentially desirable to weight recent sequence history higher in adaptation. Adding a global decay prior accomplishes this by causing previous adaptation updates to decay exponentially over time. The use of a decay prior for dynamic evaluation relates to an update rule used for fast weights (Ba et al., 2016), which decayed fast weights towards zero. For SGD with a global prior, learning rate η, and decay rate λ we form the update rule

$$\theta_i \leftarrow \theta_{i-1} - \eta \nabla \mathcal{L}(s_i) + \lambda(\theta_g - \theta_l^{i-1}). \quad (4)$$

We then consider using an RMSprop (Tieleman & Hinton, 2012) derived update rule for the learning rule in place of SGD. RMSprop uses a moving average of recent squared gradients to scale learning rates for each weight. In dynamic evaluation, near the start of a test sequence, RMSprop has had very few gradients to average, and therefore may not be able to leverage its updates as effectively. For this reason, we collect mean squared gradients, MS_g, on the training data rather than on recent test data. MS_g is given by

$$MS_g = \frac{1}{N_b} \sum_{k=1}^{N_b} (\nabla \mathcal{L}_k)^2, \quad (5)$$

where N_b is the number of training batches and $\nabla \mathcal{L}_k$ is the gradient on the kth training batch. The mini-batch size for this computation becomes a hyper-parameter, as larger mini-batches will result in smaller mean squared gradients. The update rule, which we call RMS with a global prior in our experiments, is then

$$\theta_l^i \leftarrow \theta_l^{i-1} - \eta \frac{\nabla \mathcal{L}(s_i)}{\sqrt{MS_g} + \epsilon} + \lambda(\theta_g - \theta_l^{i-1}), \quad (6)$$

where ϵ is a stabilization parameter. For the decay step of our update rule, we also scale the decay rate for each parameter proportionally to $\sqrt{MS_g}$, since parameters with a high RMS gradient affect the dynamics of the network

more. RMS_{norm} is $\sqrt{MS_g}$ divided by its mean, resulting in a normalized version of $\sqrt{MS_g}$ with a mean of 1:

$$RMS_{\text{norm}} = \frac{\sqrt{MS_g}}{\text{avg}(\sqrt{MS_g})}. \quad (7)$$

We clip the values of RMS_{norm} to be no greater than $1/\lambda$ to ensure that the decay rate does not exceed 1 for any parameter. Combining the learning component and the regularization component results in the final update equation, which we refer to as RMS with an RMS global prior

$$\theta_l^i \leftarrow \theta_l^{i-1} - \eta \frac{\nabla \mathcal{L}(s_i)}{\sqrt{MS_g + \epsilon}} + \lambda(\theta_g - \theta_l^{i-1}) \odot RMS_{\text{norm}}. \quad (8)$$

Hyper-parameter tuning: Regardless of update rule, we found it was important to properly tune the hyper-parameters of dynamic evaluation. Like in the neural cache model, this hyper-parameter tuning procedure applied a post training step in which the model was dynamically evaluated over different hyper-parameter settings. We found tuning the learning rate was by far the most important, however we also found a small benefit to tuning the decay parameter. Hyper-parameter tuning for dynamic evaluation is much faster than hyper-parameter tuning for general training, because it requires a single pass through the validation set per setting. We also found that it is possible to use a small subset of the validation set to tune hyper-parameters and achieve a similar performance. We suspect that poor hyper-parameter tuning is why past dynamic evaluation results have been mixed. For instance, Sprechmann et al. (2018) reported using dynamic evaluation with optimisation parameters obtained during training, and achieved minimal test time improvements.

6. Sparse dynamic evaluation

Mini-batching over sequences is desirable for some test-time applications because it allows faster processing of multiple sequences in parallel. Dynamic evaluation has a high memory cost for mini-batching because it is necessary to store a different set of parameters for each sequence in the mini-batch. Therefore, we consider a sparse dynamic evaluation variant that updates a smaller number of parameters. We introduce a new adaptation matrix $\mathcal{M}$ which is initialized to zeros. $\mathcal{M}$ multiplies hidden state vector h_t of an RNN at every time-step to get a new hidden state h'_t, via

$$h'_t = h_t + \mathcal{M} h_t. \quad (9)$$

h'_t then replaces h_t and is propagated throughout the network via both recurrent and feed-forward connections. In a stacked RNN, this formulation could be applied to every layer or just one layer. Applying dynamic evaluation to $\mathcal{M}$ avoids the need to apply dynamic evaluation to the original parameters of the network, reduces the number of adaptation parameters, and makes mini-batching less memory intensive. We reduce the number of adaptation parameters further by only using $\mathcal{M}$ to transform an arbitrary subset of H hidden units. This results in $\mathcal{M}$ being an $H \times H$ matrix with $d = H^2$ adaptation parameters. If H is chosen to be much smaller than the number of hidden units, this reduces the number of adaptation parameters dramatically.

7. Experiments

We applied dynamic evaluation to word- and character-level language modelling[1]. After training the base model, we tune hyper-parameters for dynamic evaluation on the validation set, and evaluate both the static and dynamic versions of the model on the test set. We also analyse the sequence lengths for which dynamic evaluation is useful, and investigate how dynamic evaluation can generalize to related words.

7.1. Small scale word-level language modelling

We performed word-level language modelling experiments on the Penn Treebank (PTB, Marcus et al., 1993) and WikiText-2 (Merity et al., 2017) datasets. These experiments compared the performance of static and dynamic evaluation, different dynamic evaluation variants, and the neural cache.

The PTB language modelling dataset, which is derived from Wall Street Journal articles, contains 929k training tokens with a vocabulary limited to 10k words. WikiText-2 is roughly twice the size of PTB, with 2 million training tokens and a vocabulary size of 33k. It features articles in a non-shuffled order, with dependencies across articles that adaptive methods should be able to exploit.

For our baseline model, we use the recent state-of-the-art averaged SGD (ASGD) weight-dropped LSTM (AWD-LSTM, Merity et al., 2018). The AWD-LSTM is a vanilla LSTM that combines the use of drop-connect (Wan et al., 2013) on recurrent weights for regularization, and a variant of ASGD (Polyak & Juditsky, 1992) for optimisation. Our model used 3 layers and tied input and output embeddings (Press & Wolf, 2017; Inan et al., 2017), and was intended to be a direct replication of AWD-LSTM.

We experiment with traditional dynamic evaluation, as well as each modification we make building up to our final update rule as described in Section 5. We also apply our proposed hyper-parameter tuning scheme to all dynamic evaluation methods. Results for PTB are given in Tab. 1, and results for Wikitext-2 are given in Tab. 2. As our final update rule (RMS + RMS global prior) worked best, we use this for all

[1] code available at https://github.com/benkrause/dynamic-evaluation

model	parameters	valid	test
RNN+LDA+kN-5+cache (Mikolov & Zweig, 2012)			92.0
CharCNN (Kim et al., 2016)	19M		78.9
LSTM (Zaremba et al., 2014)	66M	82.2	78.4
Variational LSTM (Gal & Ghahramani, 2016)	66M		73.4
Pointer sentinel-LSTM (Merity et al., 2017)	21M	72.4	70.9
Variational LSTM + augmented loss (Inan et al., 2017)	51M	71.1	68.5
Variational RHN (Zilly et al., 2017)	23M	67.9	65.4
NAS cell (Zoph & Le, 2017)	54M		62.4
Variational LSTM + gradual learning (Aharoni et al., 2017)	105M		61.7
LSTM + BB tuning (Melis et al., 2018)	24M	60.9	58.3
LSTM (Grave et al., 2017)		86.9	82.3
LSTM + neural cache (Grave et al., 2017)		74.6	72.1
AWD-LSTM (Merity et al., 2018)	24M	60.0	57.3
AWD-LSTM + neural cache (Merity et al., 2018)	24M	53.9	52.8
AWD-LSTM (rerun)	**24M**	**59.8**	**57.7**
AWD-LSTM + traditional dynamic eval (sgd, bptt=1)	**24M**	**54.9**	**53.5**
AWD-LSTM + dynamic eval (sgd, bptt=5)	**24M**	**54.7**	**53.3**
AWD-LSTM + dynamic eval (sgd, bptt=5, global prior)	**24M**	**54.0**	**52.4**
AWD-LSTM + dynamic eval (RMS, bptt=5, global prior)	**24M**	**52.7**	**52.0**
AWD-LSTM + dynamic eval (RMS, bptt=5, RMS global prior)	**24M**	**51.6**	**51.1**

Table 1. Penn Treebank perplexities. bptt refers to sequence segment lengths. The bold lines show that dynamic evaluation gives a large improvement over a state-of-the-art static model. Each of our other contributions leads to further improvements.

future experiments and use "dynamic eval" by default to refer to this update rule in tables.

All dynamic evaluation variants give large improvements to both datasets. We demonstrate much larger improvements on PTB even with traditional dynamic evaluation than some past work (Mikolov, 2012), highlighting the importance of using our proposed hyper-parameter tuning scheme. Our most advanced dynamic evaluation variant achieves better final results than the neural cache, improving the state-of-the-art on PTB and WikiText-2. This improvement is especially pronounced on WikiText-2, suggesting that dynamic evaluation is exploiting the rich vocabulary or the non-shuffled order of documents. Since publishing our code, the state-of-the-art on PTB and Wikitext-2 has been further improved by applying our dynamic evaluation implementation on top of a stronger base model (Yang et al., 2018).

7.2. Medium scale word-level language modelling

We benchmarked the performance of dynamic evaluation against static evaluation and the neural cache on the larger text8 dataset. Like WikiText-2, text8 is derived from Wikipedia text. Text8 was introduced for word-level language modelling by Mikolov et al. (2014), who preprocessed the data by mapping rare words to an '<unk>' token, resulting in a vocabulary of 44k and 17M training tokens. We use the same test set as in Mikolov et al. (2014), but also hold out the final 100k training tokens as a validation set to allow for fair hyper-parameter tuning (the original task did not have a validation set). We trained an AWD-LSTM with 52M parameters using the implementation from Merity et al. (2018), and compared the performance of static evaluation, dynamic evaluation, and neural caching at test time.

We used the hyper-parameter settings for dynamic evaluation found on PTB, and only tuned the learning rate (to 2 significant figures). The neural cache uses 3 hyper-parameters: the cache length, a mixing parameter and a flatness parameter. Starting from a cache size of 3000, we used a series of grid searches to find optimal values for the mixing parameter and flatness parameter (to 2 significant figures). We found that the affect of varying the cache size in the range 2000–4000 was negligible, so we kept the cache size at 3000. Results are given in table 3, with the results from Grave et al. (2017) that used the same test set given for context.

Dynamic evaluation soundly outperforms static evaluation and the neural cache method, demonstrating that the benefits of dynamic evaluation are maintained when using a stronger model with more training data.

7.3. Character-level language modelling

We consider dynamic evaluation for character-level language modelling using the text8 and Hutter Prize datasets.

model	parameters	valid	test
Byte mLSTM (Krause et al., 2016)	46M	92.8	88.8
Variational LSTM (Inan et al., 2017)	28M	91.5	87.0
Pointer sentinel-LSTM (Merity et al., 2017)		84.8	80.8
LSTM + BB tuning (Melis et al., 2018)	24M	69.1	65.9
LSTM (Grave et al., 2017)		104.2	99.3
LSTM + neural cache (Grave et al., 2017)		72.1	68.9
AWD-LSTM (Merity et al., 2018)	33M	68.6	65.8
AWD-LSTM + neural cache (Merity et al., 2018)	33M	53.8	52.0
AWD-LSTM (rerun)	**33M**	**68.9**	**66.1**
AWD-LSTM + traditional dynamic eval (sgd, bptt=1)	**33M**	**51.6**	**49.0**
AWD-LSTM + dynamic eval (sgd, bptt=5)	**33M**	**51.5**	**48.8**
AWD-LSTM + dynamic eval (sgd, bptt=5, global prior)	**33M**	**49.8**	**47.4**
AWD-LSTM + dynamic eval (RMS, bptt=5, global prior)	**33M**	**46.9**	**44.7**
AWD-LSTM + dynamic eval (RMS, bptt=5, RMS global prior)	**33M**	**46.4**	**44.3**

Table 2. WikiText-2 perplexities.

model	valid	test
LSTM (Grave et al., 2017)		121.8
LSTM + neural cache (Grave et al., 2017)		99.9
AWD-LSTM	**80.0**	**87.5**
AWD-LSTM + neural cache	**67.5**	**75.1**
AWD-LSTM + dynamic eval	**63.3**	**70.3**

Table 3. text8 (word-level) perplexities

The Hutter Prize dataset (Hutter, 2006) is comprised of Wikipedia text, including XML and characters from non-Latin languages. It is 100 million UTF-8 bytes long and contains 205 unique bytes. Similarly to other reported results, we used a 90:5:5 split for training, validation, and testing. The text8 dataset is derived from the Hutter Prize dataset, with all XML removed, and lower cased to 26 characters of English text plus spaces. As with Hutter Prize, we used the standard 90:5:5 split for training, validation, and testing for text8. We used a multiplicative LSTM (mLSTM) (Krause et al., 2016) as our base model for both datasets. The mLSTMs for both tasks used 2800 hidden units, an embedding layer of 400 units, weight normalization (Salimans & Kingma, 2016), variational dropout (Gal & Ghahramani, 2016), and ADAM (Kingma & Ba, 2014) for training.

We also used sparse dynamic evaluation (Section 6) on the Hutter Prize dataset. In this case, we adapted a subset of 500 hidden units, resulting in a 500×500 adaptation matrix and 250k adaptation parameters. Our mLSTM only contained one recurrent layer, so only one adaptation matrix was needed. All of our dynamic evaluation results in this section use the final update rule given in Section 5. Results for Hutter Prize are given in Table 4, and results for text8 are given in Table 5.

Dynamic evaluation achieves large improvements to our base models and state-of-the-art results on both datasets. Sparse dynamic evaluation also achieves significant improvements on Hutter Prize using only 0.5% of the adaptation parameters of regular dynamic evaluation.

7.4. Time-scales of dynamic evaluation

Starting from the model trained on Hutter Prize, we measure the time-scales at which dynamic evaluation is useful by plotting the performance of static and dynamic evaluation against the number of characters processed on sequences from the Hutter Prize test set, and sequences in Spanish from the European Parliament dataset (Koehn, 2005).

The Hutter Prize data experiments show the timescales at which dynamic evaluation gained the advantage observed in Table 4. We divided the Hutter Prize test set into 500 sequences of length 10000, and applied static and dynamic evaluation to these sequences using the same model and methodology used to obtain results in Table 4. Losses were averaged across these 500 sequences to obtain average losses at each time step. Plots of the average cross-entropy errors against the number of Hutter characters sequenced are given in Figure 2a.

The Spanish experiments measure how dynamic evaluation handles large distribution shifts between training and test time, as Hutter Prize contains very little Spanish. We used the first 5 million characters of the Spanish European Parliament data in place of the Hutter Prize test set. The Spanish experiments used the same base model and dynamic evaluation settings as Hutter Prize. Plots of the average cross-entropy errors against the number of Spanish characters

model	parameters	test
Stacked LSTM (Graves, 2013)	21M	1.67
Stacked LSTM + traditional dynamic eval (Graves, 2013)	21M	1.33
Multiplicative integration LSTM (Wu et al., 2016)	17M	1.44
HyperLSTM (Ha et al., 2017)	27M	1.34
Hierarchical multiscale LSTM (Chung et al., 2017)		1.32
Bytenet decoder (Kalchbrenner et al., 2016)		1.31
LSTM + BB tuning (Melis et al., 2018)	46M	1.30
Recurrent highway networks (Zilly et al., 2017)	46M	1.27
Fast-slow LSTM (Mujika et al., 2017)	47M	1.25
mLSTM (Krause et al., 2016)	46M	1.24
mLSTM + sparse dynamic eval ($d = 250k$)	**46M**	**1.13**
mLSTM + dynamic eval	**46M**	**1.08**

Table 4. Hutter Prize test set error in bits/char.

model	parameters	test
Multiplicative RNN (Mikolov et al., 2012)	5M	1.54
Multiplicative integration LSTM (Wu et al., 2016)	4M	1.44
LSTM (Cooijmans et al., 2017)		1.43
Batch normalised LSTM (Cooijmans et al., 2017)		1.36
Hierarchical multiscale LSTM (Chung et al., 2017)		1.29
Recurrent highway networks (Zilly et al., 2017)	45M	1.27
mLSTM (Krause et al., 2016)	45M	1.27
mLSTM + dynamic eval	**45M**	**1.19**

Table 5. text8 (char-level) test set error in bits/char.

sequenced are given in Figure 2b.

Dynamic evaluation gave a very noticeable advantage after a few hundred characters. For Spanish this advantage continued to grow as more of the sequence was processed. For Hutter, this advantage was maximized after viewing around 2–3k characters, demonstrating that the adaptation effect was local rather than global.

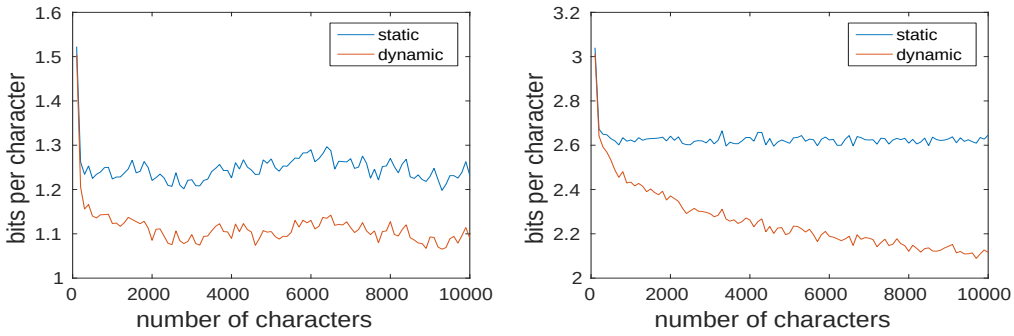
We drew 300 character conditional samples from the static and dynamic models after viewing 10k characters of Spanish. For the dynamic model, we continued to apply dynamic evaluation during sampling as well. The conditional samples are given in the appendix. The static samples quickly switched to English that resembled Hutter Prize data. The dynamic model generated data with some Spanish words and a number of made up words with characteristics of Spanish words for the entirety of the sample.

7.5. Generalizing to unseen words

Mikolov et al. (2010) hypothesized that dynamic evaluation updates generalize not only to the direct re-occurrence of words, but also to the re-occurrence of related words. The change to probabilities of symbols other than the observed symbols can be measured by doing a second forward pass after each dynamic evaluation update. We generally found that dynamic evaluation can increase the probability of related words, and we demonstrate this for a specific point in the WikiText-2 test set. We analyse the output log probabilities before and after applying dynamic evaluation to the word "production", which occurred in the following context:

"He appeared on a 2006 episode of the television series , Doctors, followed by a role in the 2007 theatre **production**"

After applying a dynamic evaluation update to the sequence segment containing the word "production", we recomputed the output probabilities at this time step with the updated network weights. We measure the change in log probabilities to words with a similar word embedding to "production". The results of this experiment is given in Table 6. Updating on the observation of the word "production" also increases the log probability of words with similar word embeddings. If a similar context were to occur again, the model would likely assign a higher probability to the word "production" as well the related words in Table 6. Words with a similar (output) embedding to a target word would also result in



(a) Hutter data

(b) Spanish data

Figure 2. Average losses in bits/char of dynamic and static evaluation plotted against number of characters processed; on sequences from the Hutter Prize test set (left) and European Parliament dataset in Spanish (right), averaged over 500 trials for each. Losses at each data point are averaged over sequence segments of length 100, and are not cumulative. Note the different y-axis scales in the two plots.

Word	emb. cos distance	Δ log prob
"production"	1.00	+2.42
"development"	0.55	+1.14
"construction"	0.53	+1.29
"filming"	0.52	+0.80
"recording"	0.50	+1.27
"photography"	0.46	+0.15
"release"	0.45	+1.25
"performance"	0.45	+1.01
"design"	0.44	+0.90
"work"	0.44	+1.23
"productions"	0.44	+0.50
median	−0.02	−0.66

Table 6. Effect of dynamic evaluation on probabilities of related words. We measure how updating on the target word "production" changes the probabilities of words most related to production, measured by cosine distance of word embeddings learned during training time (note that input and output embeddings are tied). The median values across the entire vocabulary are given in the bottom entry.

similar hidden state gradients to the target word, so updating on the target word also increases the probability of related words. This gives dynamic evaluation a greater ability to generalize to indirect repetitions in sequences.

8. Conclusion

This work explores and develops methodology for applying dynamic evaluation to sequence modelling tasks. Experiments show that the proposed dynamic evaluation methodology gives large test time improvements across character and word level language modelling. Our improvements to language modelling have applications to speech recognition and machine translation over longer contexts, as past improvements in adaptive language modelling have lead to improvements in these tasks (Jelinek et al., 1991; Tu et al., 2017; Kuang et al., 2017). Overall, dynamic evaluation is shown to be an effective method for exploiting pattern re-occurrence in sequences.

A. Appendix

300 character samples generated from the static and dynamic versions of the model trained on Hutter Prize, conditioned on 10k of Spanish characters. The final sentence fragment of the 10k conditioning characters is given to the reader, with the generated text given in **bold**.

STATIC:
*Tiene importancia este compromiso en la medida en que la Comisión es un organismo que tiene el mon**de,***
***
There is a secret act in the world except Cape Town, seen in now flat comalo and ball market and has seen the closure of the eagle as imprints in a dallas within the country." Is a topic for an increasingly small contract saying Allan Roth acquired the government in [[1916]].***

===

DYNAMIC:
*Tiene importancia este compromiso en la medida en que la Comisión es un organismo que tiene el mon**tembre tas procedíns la conscriptione se ha Tesalo del Pómienda que et hanemos que Pe la Siemina.***
De la Pedrera Orden es Señora Presidente civil, Orden de siemin presente relevante frónmida que esculdad pludiore e formidad President de la Presidenta Antidorne Adamirmidad i ciemano de el 200'. Fo

Acknowledgements

This work was partially supported by the H2020 project SUMMA, under grant agreement 688139.

References

Aharoni, Z., Rattner, G., and Permuter, H. Gradual learning of deep recurrent neural networks. *arXiv:1708.08863*, 2017.

Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z., and Ionescu, C. Using fast weights to attend to the recent past. In *NIPS*, pp. 4331–4339, 2016.

Bellegarda, J. R. Statistical language model adaptation: review and perspectives. *Speech Communication*, 42(1): 93–108, 2004.

Chung, J., Ahn, S., and Bengio, Y. Hierarchical multiscale recurrent neural networks. *ICLR*, 2017.

Cooijmans, T., Ballas, N., Laurent, C., and Courville, A. Recurrent batch normalization. *ICLR*, 2017.

Fortunato, M., Blundell, C., and Vinyals, O. Bayesian recurrent neural networks. *arXiv:1704.02798*, 2017.

Gal, Y. and Ghahramani, Z. A theoretically grounded application of dropout in recurrent neural networks. In *NIPS*, pp. 1019–1027, 2016.

Grave, E., Joulin, A., and Usunier, N. Improving neural language models with a continuous cache. *ICLR*, 2017.

Graves, A. Generating sequences with recurrent neural networks. *arXiv:1308.0850*, 2013.

Gu, J., Lu, Z., Li, H., and Li, V. O. K. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv:1603.06393*, 2016.

Ha, D., Dai, A., and Lee, Q. Hypernetworks. *ICLR*, 2017.

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

Hutter, M. The human knowledge compression prize. *URL http://prize.hutter1.net*, 2006.

Inan, H., Khosravi, K., and Socher, R. Tying word vectors and word classifiers: A loss framework for language modeling. *ICLR*, 2017.

Jelinek, F., Merialdo, B., Roukos, S., and Strauss, M. A dynamic language model for speech recognition. In *HLT*, volume 91, pp. 293–295, 1991.

Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, A., Graves, A., and Kavukcuoglu, K. Neural machine translation in linear time. *arXiv:1610.10099*, 2016.

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

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

Koehn, P. Europarl: A parallel corpus for statistical machine translation. In *MT Summit*, volume 5, pp. 79–86, 2005.

Krause, B., Lu, L., Murray, I., and Renals, S. Multiplicative LSTM for sequence modelling. *arXiv:1609.07959*, 2016.

Krause, B., Murray, I., Renals, S., and Lu, L. Multiplicative LSTM for sequence modelling. *ICLR Workshop track*, 2017. URL https://openreview.net/forum?id=SJCS5rXFl.

Kuang, S., Xiong, D., Luo, W., and Zhou, G. Cache-based document-level neural machine translation. *arXiv:1711.11221*, 2017.

Kuhn, R. Speech recognition and the frequency of recently used words: A modified Markov model for natural language. In *Proceedings of the 12th conference on Computational linguistics-Volume 1*, pp. 348–350. Association for Computational Linguistics, 1988.

Marcus, G. The algebraic mind, 2001.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):313–330, 1993.

Melis, G., Dyer, C., and Blunsom, P. On the state of the art of evaluation in neural language models. *ICLR*, 2018.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *ICLR*, 2017.

Merity, S., Keskar, N. S., and Socher, R. Regularizing and optimizing LSTM language models. *ICLR*, 2018.

Mikolov, T. *Statistical language models based on neural networks*. PhD thesis, Brno University of Technology, 2012.

Mikolov, T. and Zweig, G. Context dependent recurrent neural network language model. *SLT*, 12:234–239, 2012.

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. Recurrent neural network based language model. In *Interspeech*, volume 2, pp. 3, 2010.

Mikolov, T., Sutskever, I., Deoras, A., Le, H., Kombrink, S., and Cernocky, J. Subword language modeling with neural networks. *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)*, 2012.

Mikolov, T., Joulin, A., Chopra, S., Mathieu, M., and Ranzato, M. Learning longer memory in recurrent neural networks. *arXiv:1412.7753*, 2014.

Mujika, A., Meier, F., and Steger, A. Fast-slow recurrent neural networks. *arXiv:1705.08639*, 2017.

Ororbia II, A. G., Mikolov, T., and Reitter, D. Learning simpler language models with the differential state framework. *Neural Computation*, 2017.

Polyak, B. T. and Juditsky, A. B. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

Press, O. and Wolf, L. Using the output embedding to improve language models. *EACL 2017*, pp. 157, 2017.

Prickett, B. Vanilla sequence-to-sequence neural nets cannot model reduplication. 2017. URL https://scholarworks.umass.edu/ics_owplinguist/2/.

Salimans, T. and Kingma, D. P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NIPS*, pp. 901–909, 2016.

Schmidhuber, J. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Sprechmann, P., Jayakumar, S., Rae, J., Pritzel, A., Badia, A. P., Uria, B., Vinyals, O., Hassabis, D., Pascanu, R., and Blundell, C. Memory-based parameter adaptation. *ICLR*, 2018.

Tieleman, T. and Hinton, G. E. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4 (2), 2012.

Tu, Z., Liu, Y., Shi, S., and Zhang, T. Learning to remember translation history with a continuous cache. *arXiv:1711.09367*, 2017.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In *NIPS*, pp. 2692–2700, 2015.

Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., and Fergus, R. Regularization of neural networks using dropconnect. In *ICML*, pp. 1058–1066, 2013.

Werbos, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78:1550–1560, 1990.

Wu, Y., Zhang, S., Zhang, Y., Bengio, Y., and Salakhutdinov, R. On multiplicative integration with recurrent neural networks. In *NIPS*, 2016.

Yang, Z., Dai, Z., Salakhutdinov, R., and Cohen, W. W. Breaking the softmax bottleneck: a high-rank RNN language model. *ICLR*, 2018.

Zaremba, W., Sutskever, I., and Vinyals, O. Recurrent neural network regularization. *arXiv:1409.2329*, 2014.

Zilly, J. G., Srivastava, R. K., Koutník, J., and Schmidhuber, J. Recurrent highway networks. *ICLR*, 2017.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *ICLR*, 2017.