Generalizing and Hybridizing Count-based and Neural Language Models

Graham Neubig[†] and **Chris Dyer**[‡]
[†]Carnegie Mellon University, USA
[‡]Google DeepMind, United Kingdom

Abstract

Language models (LMs) are statistical models that calculate probabilities over sequences of words or other discrete symbols. Currently two major paradigms for language modeling exist: count-based n-gram models, which have advantages of scalability and test-time speed, and neural LMs, which often achieve superior modeling performance. We demonstrate how both varieties of models can be unified in a single modeling framework that defines a set of probability distributions over the vocabulary of words, and then dynamically calculates mixture weights over these distributions. This formulation allows us to create novel hybrid models that combine the desirable features of count-based and neural LMs, and experiments demonstrate the advantages of these approaches.[1]

1 Introduction

Language models (LMs) are statistical models that, given a sentence $w_1^I := w_1, \dots, w_I$, calculate its probability $P(w_1^I)$. LMs are widely used in applications such as machine translation and speech recognition, and because of their broad applicability they have also been widely studied in the literature. The most traditional and broadly used language modeling paradigm is that of count-based LMs, usually smoothed n-grams (Witten and Bell, 1991; Chen and Goodman, 1996). Recently, there has been a focus on LMs based on neural networks (Nakamura et al., 1990; Bengio et al., 2006; Mikolov et al., 2010), which have shown impressive improvements in performance over count-based LMs. On the other hand, these neural LMs also come at the cost of increased computational complexity at both training and test time, and even the largest reported neural LMs (Chen et al., 2015; Williams et al., 2015) are trained on a fraction of the data of their count-based counterparts (Brants et al., 2007).

In this paper we focus on a class of LMs, which we will call *mixture of distributions LMs* (MODLMs; §2). Specifically, we define MODLMs as all LMs that take the following form, calculating the probabilities of the next word in a sentence w_i given preceding context $\boldsymbol{c}$ according to a mixture of several component probability distributions $P_k(w_i|\boldsymbol{c})$:

$$P(w_i|\boldsymbol{c}) = \sum_{k=1}^{K} \lambda_k(\boldsymbol{c}) P_k(w_i|\boldsymbol{c}). \tag{1}$$

Here, $\lambda_k(\boldsymbol{c})$ is a function that defines the mixture weights, with the constraint that $\sum_{k=1}^{K} \lambda_k(\boldsymbol{c}) = 1$ for all $\boldsymbol{c}$. This form is not new in itself, and widely used both in the calculation of smoothing coefficients for n-gram LMs (Chen and Goodman, 1996), and interpolation of LMs of various varieties (Jelinek and Mercer, 1980).

The main contribution of this paper is to demonstrate that depending on our definition of $\boldsymbol{c}$, $\lambda_k(\boldsymbol{c})$, and $P_k(w_i|\boldsymbol{c})$, Eq. 1 can be used to describe not only n-gram models, but also feed-forward (Nakamura et al., 1990; Bengio et al., 2006; Schwenk, 2007) and

[1]Work was performed while GN was at the Nara Institute of Science and Technology and CD was at Carnegie Mellon University. Code and data to reproduce experiments is available at `http://github.com/neubig/modlm`

recurrent (Mikolov et al., 2010; Sundermeyer et al., 2012) neural network LMs (§3). This observation is useful theoretically, as it provides a single mathematical framework that encompasses several widely used classes of LMs. It is also useful practically, in that this new view of these traditional models allows us to create new models that combine the desirable features of n-gram and neural models, such as:

neurally interpolated n-gram LMs (§4.1), which learn the interpolation weights of n-gram models using neural networks, and

neural/n-gram hybrid LMs (§4.2), which add a count-based n-gram component to neural models, allowing for flexibility to add large-scale external data sources to neural LMs.

We discuss learning methods for these models (§5) including a novel method of randomly dropping out more easy-to-learn distributions to prevent the parameters from falling into sub-optimal local minima.

Experiments on language modeling benchmarks (§6) find that these models outperform baselines in terms of performance and convergence speed.

2 Mixture of Distributions LMs

As mentioned above, MODLMs are LMs that take the form of Eq. 1. This can be re-framed as the following matrix-vector multiplication:

$$\boldsymbol{p}_{\boldsymbol{c}}^{\intercal} = D_{\boldsymbol{c}} \boldsymbol{\lambda}_{\boldsymbol{c}}^{\intercal},$$

where $\boldsymbol{p}_{\boldsymbol{c}}$ is a vector with length equal to vocabulary size, in which the jth element $p_{\boldsymbol{c},j}$ corresponds to $P(w_i = j|\boldsymbol{c})$, $\boldsymbol{\lambda}_{\boldsymbol{c}}$ is a size K vector that contains the mixture weights for the distributions, and $D_{\boldsymbol{c}}$ is a J-by-K matrix, where element $d_{\boldsymbol{c},j,k}$ is equivalent to the probability $P_k(w_i = j|\boldsymbol{c})$.[2] An example of this formulation is shown in Fig. 1.

Note that all columns in D represent probability distributions, and thus must sum to one over the J words in the vocabulary, and that all $\boldsymbol{\lambda}$ must sum to 1 over the K distributions. Under this condition, the vector $\boldsymbol{p}$ will represent a well-formed probability distribution as well. This conveniently allows us to

$$\overbrace{\begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_J \end{bmatrix}}^{\text{Probabilities } \boldsymbol{p}^{\intercal}} = \underbrace{\begin{bmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,K} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ d_{J,1} & d_{J,2} & \cdots & d_{J,K} \end{bmatrix}}_{\text{Distribution matrix } D} \overbrace{\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_K \end{bmatrix}}^{\text{Coefficients } \boldsymbol{\lambda}^{\intercal}}$$

Figure 1: MODLMs as linear equations

calculate the probability of a single word $w_i = j$ by calculating the product of the jth row of $D_{\boldsymbol{c}}$ and $\boldsymbol{\lambda}_{\boldsymbol{c}}^{\intercal}$

$$P_k(w_i = j|\boldsymbol{c}) = \boldsymbol{d}_{\boldsymbol{c},j} \boldsymbol{\lambda}_{\boldsymbol{c}}^{\intercal}.$$

In the sequel we show how this formulation can be used to describe several existing LMs (§3) as well as several novel model structures that are more powerful and general than these existing models (§4).

3 Existing LMs as Linear Mixtures

3.1 n-gram LMs as Mixtures of Distributions

First, we discuss how count-based interpolated n-gram LMs fit within the MODLM framework.

Maximum likelihood estimation: n-gram models predict the next word based on the previous N-1 words. In other words, we set $\boldsymbol{c} = w_{i-N+1}^{i-1}$ and calculate $P(w_i|w_{i-N+1}^{i-1})$. The maximum-likelihood (ML) estimate for this probability is

$$P_{ML}(w_i|w_{i-N+1}^{i-1}) = c(w_{i-N+1}^{i})/c(w_{i-N+1}^{i-1}),$$

where $c(\cdot)$ counts frequency in the training corpus.

Interpolation: Because ML estimation assigns zero probability to word sequences where $c(w_{i-N+1}^{i}) = 0$, n-gram models often interpolate the ML distributions for sequences of length 1 to N. The simplest form is static interpolation

$$P(w_i|w_{i-n+1}^{i-1}) = \sum_{n=1}^{N} \lambda_{S,n} P_{ML}(w_i|w_{i-n+1}^{i-1}). \quad (2)$$

$\boldsymbol{\lambda}_S$ is a vector where $\lambda_{S,n}$ represents the weight put on the distribution $P_{ML}(w_i|w_{i-n+1}^{i-1})$. This can be expressed as linear equations (Fig. 2a) by setting the nth column of D to the ML distribution $P_{ML}(w_i|w_{i-n+1}^{i-1})$, and $\boldsymbol{\lambda}(\boldsymbol{c})$ equal to $\boldsymbol{\lambda}_S$.

[2]We omit the subscript $\boldsymbol{c}$ when appropriate.

$$\overbrace{\begin{bmatrix} p_1\\ p_2\\ \vdots\\ p_J\end{bmatrix}}^{\text{Probabilities } \boldsymbol{p}^\intercal} = \underbrace{\begin{bmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,N}\\ d_{2,1} & d_{2,2} & \cdots & d_{2,N}\\ \vdots & \vdots & \ddots & \vdots\\ d_{J,1} & d_{J,2} & \cdots & d_{J,N}\end{bmatrix}}_{\text{Count-based probabilities } P_C(w_i = j|w_{i-n+1}^{i-1})} \overbrace{\begin{bmatrix}\lambda_1\\ \lambda_2\\ \vdots\\ \lambda_N\end{bmatrix}}^{\text{Heuristic interp. coefficients } \boldsymbol{\lambda}^\intercal}$$

(a) Interpolated n-grams as MODLMs

$$\overbrace{\begin{bmatrix} p_1\\ p_2\\ \vdots\\ p_J\end{bmatrix}}^{\text{Probabilities } \boldsymbol{p}^\intercal} = \underbrace{\begin{bmatrix} 1 & 0 & \cdots & 0\\ 0 & 1 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 1\end{bmatrix}}_{\text{J-by-J identity matrix } I} \overbrace{\begin{bmatrix}\lambda_1\\ \lambda_2\\ \vdots\\ \lambda_J\end{bmatrix}}^{\text{Result of softmax(NN}(\boldsymbol{c}))}$$

(b) Neural LMs as MODLMs

Figure 2: Interpretations of existing models as mixtures of distributions

Static interpolation can be improved by calculating $\boldsymbol{\lambda}(\boldsymbol{c})$ dynamically, using heuristics based on the frequency counts of the context (Good, 1953; Katz, 1987; Witten and Bell, 1991). These methods define a context-sensitive fallback probability $\alpha(w_{i-n+1}^{i-1})$ for order n models, and recursively calculate the probability of the higher order models from the lower order models:

$$\begin{aligned} P(w_i|w_{i-n+1}^{i-1}) = \alpha(w_{i-n+1}^{i-1}) P(w_i|w_{i-n+2}^{i-1}) + \\ (1-\alpha(w_{i-n+1}^{i-1})) P_{ML}(w_i|w_{i-n+1}^{i-1}). \end{aligned} \quad (3)$$

To express this as a linear mixture, we convert $\alpha(w_{i-n+1}^{i-1})$ into the appropriate value for $\lambda_n(w_{i-N+1}^{i-1})$. Specifically, the probability assigned to each $P_{ML}(w_i|w_{i-n+1}^{i-1})$ is set to the product of the fallbacks α for all higher orders and the probability of not falling back $(1-\alpha)$ at the current level:

$$\lambda_n(w_{i-N+1}^{i-1}) = (1-\alpha(w_{i-n+1}^{i-1})) \prod_{\tilde{n}=n+1}^{N} \alpha(w_{i-\tilde{n}+1}^{i-1}).$$

Discounting: The widely used technique of discounting (Ney et al., 1994) defines a fixed discount d and subtracts it from the count of each word before calculating probabilities:

$$P_D(w_i|w_{i-n+1}^{i-1}) = (c(w_{i-n+1}^{i}) - d)/c(w_{i-n+1}^{i-1}).$$

Discounted LMs then assign the remaining probability mass after discounting as the fallback probability

$$\begin{aligned} \beta_D(w_{i-n+1}^{i-1}) =& 1 - \sum_{j=1}^{J} P_D(w_i = j|w_{i-n+1}^{i-1}), \\ P(w_i|w_{i-n+1}^{i-1}) =& \beta_D(w_{i-n+1}^{i-1}) P(w_i|w_{i-n+2}^{i-1}) + \\ & P_D(w_i|w_{i-n+1}^{i-1}). \end{aligned} \quad (4)$$

In this case, $P_D(\cdot)$ does not add to one, and thus violates the conditions for MODLMs stated in §2, but it is easy to turn discounted LMs into interpolated LMs by normalizing the discounted distribution:

$$P_{ND}(w_i|w_{i-n+1}^{i-1}) = \frac{P_D(w_i|w_{i-n+1}^{i-1})}{\sum_{j=1}^{J} P_D(w_i = j|w_{i-n+1}^{i-1})},$$

which allows us to replace $\beta(\cdot)$ for $\alpha(\cdot)$ and $P_{ND}(\cdot)$ for $P_{ML}(\cdot)$ in Eq. 3, and proceed as normal.

Kneser–Ney (KN; Kneser and Ney (1995)) and Modified KN (Chen and Goodman, 1996) smoothing further improve discounted LMs by adjusting the counts of lower-order distributions to more closely match their expectations as fallbacks for higher order distributions. Modified KN is currently the defacto standard in n-gram LMs despite occasional improvements (Teh, 2006; Durrett and Klein, 2011), and we will express it as $P_{KN}(\cdot)$.

3.2 Neural LMs as Mixtures of Distributions

In this section we demonstrate how neural network LMs can also be viewed as an instantiation of the MODLM framework.

Feed-forward neural network LMs: Feed-forward LMs (Bengio et al., 2006; Schwenk, 2007) are LMs that, like n-grams, calculate the probability of the next word based on the previous words. Given context w_{i-N+1}^{i-1}, these words are converted into real-valued word representation vectors $\boldsymbol{r}_{i-N+1}^{i-1}$, which are concatenated into an overall representation vector $\boldsymbol{q} = \oplus(\boldsymbol{r}_{i-N+1}^{i-1})$, where $\oplus(\cdot)$ is the vector concatenation function. $\boldsymbol{q}$ is then run through a series of affine transforms and non-linearities defined as function NN$(\boldsymbol{q})$ to obtain a vector $\boldsymbol{h}$. For example, for a one-layer neural net-

$$\overbrace{\begin{bmatrix}p_1\\p_2\\\vdots\\p_J\end{bmatrix}}^{\text{Probabilities } \boldsymbol{p}^\intercal} = \underbrace{\begin{bmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,N}\\ d_{1,2} & d_{2,2} & \cdots & d_{2,N}\\ \vdots&\vdots&\ddots&\vdots\\ d_{J,1}&d_{J,2}&\cdots&d_{J,N}\end{bmatrix}}_{\text{Count-based probabilities } P_C(w_i=j|w_{i-n+1}^{i-1})} \overbrace{\begin{bmatrix}\lambda_1\\\lambda_2\\\vdots\\\lambda_N\end{bmatrix}}^{\text{Result of softmax(NN}(\boldsymbol{c}))}$$

(a) Neurally interpolated n-gram LMs

$$\overbrace{\begin{bmatrix}p_1\\p_2\\\vdots\\p_J\end{bmatrix}}^{\text{Probabilities } \boldsymbol{p}^\intercal} = \underbrace{\begin{bmatrix} d_{1,1} & \cdots & d_{1,N} & 1 & \cdots & 0\\ d_{2,1} & \cdots & d_{2,N} & 0 & \cdots & 0\\ \vdots&\ddots&\vdots&\vdots&\ddots&\vdots\\ d_{J,1}&\cdots&d_{J,N}&0&\cdots&1\end{bmatrix}}_{\text{Count-based probabilities \underline{and} } J\text{-by-}J \text{ identity matrix}} \overbrace{\begin{bmatrix}\lambda_1\\\lambda_2\\\vdots\\\lambda_{J+N}\end{bmatrix}}^{\text{Result of softmax(NN}(\boldsymbol{c}))}$$

(b) Neural/n-gram hybrid LMs

Figure 3: Two new expansions to n-gram and neural LMs made possible in the MODLM framework

work with a tanh non-linearity we can define

$$\text{NN}(\boldsymbol{q}) := \tanh(\boldsymbol{q}W_q + \boldsymbol{b}_q), \tag{5}$$

where W_q and $\boldsymbol{b}_q$ are weight matrix and bias vector parameters respectively. Finally, the probability vector $\boldsymbol{p}$ is calculated using the softmax function $\boldsymbol{p} = \text{softmax}(\boldsymbol{h}W_s + \boldsymbol{b}_s)$, similarly parameterized.

As these models are directly predicting $\boldsymbol{p}$ with no concept of mixture weights $\boldsymbol{\lambda}$, they cannot be interpreted as MODLMs as-is. However, we can perform a trick shown in Fig. 2b, not calculating $\boldsymbol{p}$ directly, but instead calculating mixture weights $\boldsymbol{\lambda} = \text{softmax}(\boldsymbol{h}W_s + \boldsymbol{b}_s)$, and defining the MODLM's distribution matrix D as a J-by-J identity matrix. This is equivalent to defining a linear mixture of J Kronecker δ_j distributions, the jth of which assigns a probability of 1 to word j and zero to everything else, and estimating the mixture weights with a neural network. While it may not be clear why it is useful to define neural LMs in this somewhat roundabout way, we describe in §4 how this opens up possibilities for novel expansions to standard models.

Recurrent neural network LMs: LMs using recurrent neural networks (RNNs) (Mikolov et al., 2010) consider not the previous few words, but also maintain a hidden state summarizing the sentence up until this point by re-defining the net in Eq. 5 as

$$\text{RNN}(\boldsymbol{q}_i) := \tanh(\boldsymbol{q}_i W_q + \boldsymbol{h}_{i-1} W_h + \boldsymbol{b}_q),$$

where $\boldsymbol{q}_i$ is the current input vector and $\boldsymbol{h}_{i-1}$ is the hidden vector at the previous time step. This allows for consideration of long-distance dependencies beyond the scope of standard n-grams, and LMs using RNNs or long short-term memory (LSTM) networks (Sundermeyer et al., 2012) have posted large improvements over standard n-grams and feed-forward models. Like feed-forward LMs, LMs using RNNs can be expressed as MODLMs by predicting $\boldsymbol{\lambda}$ instead of predicting $\boldsymbol{p}$ directly.

4 Novel Applications of MODLMs

This section describes how we can use this framework of MODLMs to design new varieties of LMs that combine the advantages of both n-gram and neural network LMs.

4.1 Neurally Interpolated n-gram Models

The first novel instantiation of MODLMs that we propose is *neurally interpolated n-gram models*, shown in Fig. 3a. In these models, we set D to be the same matrix used in n-gram LMs, but calculate $\boldsymbol{\lambda}(\boldsymbol{c})$ using a neural network model. As $\boldsymbol{\lambda}(\boldsymbol{c})$ is learned from data, this framework has the potential to allow us to learn more intelligent interpolation functions than the heuristics described in §3.1. In addition, because the neural network only has to calculate a softmax over N distributions instead of J vocabulary words, training and test efficiency of these models can be expected to be much greater than that of standard neural network LMs.

Within this framework, there are several design decisions. First, how we decide D: do we use the maximum likelihood estimate P_{ML} or KN estimated distributions P_{KN}? Second, what do we provide as input to the neural network to calculate the mixture weights? To provide the neural net with the same information used by interpolation heuristics used in traditional LMs, we first calculate three features for each of the N contexts w_{i-n+1}^{i-1}: a binary feature indicating whether the context has been observed in the training corpus ($c(w_{i-n+1}^{i-1}) > 0$), the log frequency of the context counts ($\log(c(w_{i-n+1}^{i-1}))$ or zero for unobserved contexts), and the log frequency of the number of unique words following the context ($\log(u(w_{i-n+1}^{i-1}))$ or likewise zero). When using discounted distributions, we also use the log of the sum of the discounted counts as a feature. We can also optionally use the word representation vector $\boldsymbol{q}$ used in neural LMs, allowing for richer representation of the input, but this may or may not be necessary in the face of the already informative count-based features.

4.2 Neural/n-gram Hybrid Models

Our second novel model enabled by MODLMs is *neural/n-gram hybrid models*, shown in Fig. 3b. These models are similar to neurally interpolated n-grams, but D is augmented with J additional columns representing the Kronecker δ_j distributions used in the standard neural LMs. In this construction, $\boldsymbol{\lambda}$ is still a stochastic vector, but its contents are both the mixture coefficients for the count-based models and direct predictions of the probabilities of words. Thus, the learned LM can use count-based models when they are deemed accurate, and deviate from them when deemed necessary.

This model is attractive conceptually for several reasons. First, it has access to all information used by both neural and n-gram LMs, and should be able to perform as well or better than both models. Second, the efficiently calculated n-gram counts are likely sufficient to capture many phenomena necessary for language modeling, allowing the neural component to focus on learning only the phenomena that are not well modeled by n-grams, requiring fewer parameters and less training time. Third, it is possible to train n-grams from much larger amounts of data, and use these massive models to bootstrap learning of neural nets on smaller datasets.

5 Learning Mixtures of Distributions

While the MODLM formulations of standard heuristic n-gram LMs do not require learning, the remaining models are parameterized. This section discusses the details of learning these parameters.

5.1 Learning MODLMs

The first step in learning parameters is defining our training objective. Like most previous work on LMs (Bengio et al., 2006), we use a negative log-likelihood loss summed over words w_i in every sentence $\boldsymbol{w}$ in corpus $\mathcal{W}$

$$L(\mathcal{W}) = -\sum_{\boldsymbol{w}\in\mathcal{W}}\sum_{w_i\in\boldsymbol{w}} \log P(w_i|\boldsymbol{c}),$$

where $\boldsymbol{c}$ represents all words preceding w_i in $\boldsymbol{w}$ that are used in the probability calculation. As noted in Eq. 2, $P(w_i = j|\boldsymbol{c})$ can be calculated efficiently from the distribution matrix $D_{\boldsymbol{c}}$ and mixture function output $\boldsymbol{\lambda}_{\boldsymbol{c}}$.

Given that we can calculate the log likelihood, the remaining parts of training are similar to training for standard neural network LMs. As usual, we perform forward propagation to calculate the probabilities of all the words in the sentence, back-propagate the gradients through the computation graph, and perform some variant of stochastic gradient descent (SGD) to update the parameters.

5.2 Block Dropout for Hybrid Models

While the training method described in the previous section is similar to that of other neural network models, we make one important modification to the training process specifically tailored to the hybrid models of §4.2.

This is motivated by our observation (detailed in §6.3) that the hybrid models, despite being strictly more expressive than the corresponding neural network LMs, were falling into poor local minima with higher training error than neural network LMs. This is because at the very beginning of training, the count-based elements of the distribution matrix in Fig. 3b are already good approximations of the target distribution, while the weights of the single-word δ_j distributions are not yet able to provide accurate probabilities. Thus, the model learns to set the mixture proportions of the δ elements to near zero and rely mainly on the count-based n-gram distributions.

To encourage the model to use the δ mixture components, we adopt a method called *block dropout* (Ammar et al., 2016). In contrast to standard dropout (Srivastava et al., 2014), which drops out single nodes or connections, block dropout randomly drops out entire subsets of network nodes. In our case, we want to prevent the network from overusing the count-based n-gram distributions, so for a randomly selected portion of the training examples (here, 50%) we disable all n-gram distributions and force the model to rely on only the δ distributions. To do so, we zero out all elements in $\boldsymbol{\lambda}(\boldsymbol{c})$ that correspond to n-gram distributions, and re-normalize over the rest of the elements so they sum to one.

5.3 Network and Training Details

Finally, we note design details that were determined based on preliminary experiments.

Network structures: We used both feed-forward networks with tanh non-linearities and LSTM (Hochreiter and Schmidhuber, 1997) networks. Most experiments used single-layer 200-node networks, and 400-node networks were used for experiments with larger training data. Word representations were the same size as the hidden layer. Larger and multi-layer networks did not yield improvements.

Training: We used ADAM (Kingma and Ba, 2015) with a learning rate of 0.001, and minibatch sizes of 512 words. This led to faster convergence than standard SGD, and more stable optimization than other update rules. Models were evaluated every 500k-3M words, and the model with the best development likelihood was used. In addition to the block dropout of §5.2, we used standard dropout with a rate of 0.5 for both feed-forward (Srivastava et al., 2014) and LSTM (Pham et al., 2014) nets in the neural LMs and neural/n-gram hybrids, but not in the neurally interpolated n-grams, where it resulted in slightly worse perplexities.

Features: If parameters are learned on the data used to train count-based models, they will heavily over-fit and learn to trust the count-based distributions too much. To prevent this, we performed 10-fold cross validation, calculating count-based elements of D for each fold with counts trained on the other 9/10. In addition, the count-based contextual features in §4.1 were normalized by subtracting the training set mean, which improved performance.

6 Experiments

6.1 Experimental Setup

In this section, we perform experiments to evaluate the neurally interpolated n-grams (§6.2) and neural/n-gram hybrids (§6.3), the ability of our models to take advantage of information from large data sets (§6.4), and the relative performance compared to post-facto static interpolation of already-trained models (§6.5). For the main experiments, we evaluate on two corpora: the Penn Treebank (PTB) data set prepared by Mikolov et al. (2010),[3] and the first 100k sentences in the English side of the ASPEC corpus (Nakazawa et al., 2015)[4] (details in Tab. 1). The PTB corpus uses the standard vocabulary of 10k words, and for the ASPEC corpus we use a vocabulary of the 20k most frequent words. Our implementation is included as supplementary material.

Table 1: Data sizes for the PTB and ASPEC corpora.

PTB	Sent	Word	ASP	Sent	Word
train	42k	890k	train	100k	2.1M
valid	3.4k	70k	valid	1.8k	45k
test	3.8k	79k	test	1.8k	46k

Table 2: PTB/ASPEC perplexities for traditional heuristic (HEUR) and proposed neural net (FF or LSTM) interpolation methods using ML or KN distributions, and count (C) or count+word representation (CR) features.

Dst./Ft.	HEUR	FF	LSTM
ML/C	220.5/265.9	146.6/164.5	144.4/162.7
ML/CR	-	145.7/163.9	142.6/158.4
KN/C	140.8/156.5	138.9/152.5	136.8/151.1
KN/CR	-	136.9/153.0	**135.2/149.1**

6.2 Results for Neurally Interpolated n-grams

First, we investigate the utility of neurally interpolated n-grams. In all cases, we use a history of $N = 5$ and test several different settings for the models:

Estimation type: $\boldsymbol{\lambda}(\boldsymbol{c})$ is calculated with heuristics (HEUR) or by the proposed method using feed-forward (FF), or LSTM nets.

Distributions: We compare $P_{ML}(\cdot)$ and $P_{KN}(\cdot)$. For heuristics, we use Witten-Bell for ML and the appropriate discounted probabilities for KN.

Input features: As input features for the neural network, we either use only the count-based features (C) or count-based features together with the word representation for the single previous word (CR).

From the results shown in Tab. 2, we can first see that when comparing models using the same set of

[3]`http://rnnlm.org/simple-examples.tgz`

[4]`http://lotus.kuee.kyoto-u.ac.jp/ASPEC/`

input distributions, the neurally interpolated model outperforms corresponding heuristic methods. We can also see that LSTMs have a slight advantage over FF nets, and models using word representations have a slight advantage over those that use only the count-based features. Overall, the best model achieves a relative perplexity reduction of 4-5% over KN models. Interestingly, even when using simple ML distributions, the best neurally interpolated n-gram model nearly matches the heuristic KN method, demonstrating that the proposed model can automatically learn interpolation functions that are nearly as effective as carefully designed heuristics.[5]

6.3 Results for Neural/n-gram Hybrids

In experiments with hybrid models, we test a neural/n-gram hybrid LM using LSTM networks with both Kronecker δ and KN smoothed 5-gram distributions, trained either with or without block dropout. As our main baseline, we compare to LSTMs with only δ distributions, which have reported competitive numbers on the PTB data set (Zaremba et al., 2014).[6] We also report results for heuristically smoothed KN 5-gram models, and the best neurally interpolated n-grams from the previous section for reference.

The results, shown in Tab. 3, demonstrate that similarly to previous research, LSTM LMs (2) achieve a large improvement in perplexity over n-gram models, and that the proposed neural/n-gram hybrid method (5) further reduces perplexity by 10-11% relative over this strong baseline.

Comparing models without (4) and with (5) the proposed block dropout, we can see that this method contributes significantly to these gains. To examine this more closely, we show the test perplexity for the three models using δ distributions in Tab. 5, and the amount of the probability mass in $\boldsymbol{\lambda}(\boldsymbol{c})$ assigned to the non-δ distributions in the hybrid models. From this, we can see that the model with block dropout quickly converges to a better result than the LSTM LM, but the model without converges to a worse result, assigning too much probability mass to the dense count-based distributions, demonstrating the learning problems mentioned in §5.2.

It is also of interest to examine exactly why the proposed model is doing better than the more standard methods. One reason can be found in the behavior with regards to low-frequency words. In Figure 4, we show perplexities for words that appear n times or less in the training corpus, for $n = 10$, $n = 100$, $n = 1000$ and $n = \infty$ (all words). From the results, we can first see that if we compare the baselines, LSTM language models achieve better perplexities overall but n-gram language models tend to perform better on low-frequency words, corroborating the observations of Chen et al. (2015).

Table 3: PTB/ASPEC perplexities for traditional KN (1) and LSTM LMs (2), neurally interpolated n-grams (3), and neural/n-gram hybrid models without (4) and with (5) block dropout.

	Dist.	Interp.	PPL
(1)	KN	HEUR	140.8/156.5
(2)	δ	LSTM	105.9/116.9
(3)	KN	LSTM	135.2/149.1
(4)	KN,δ	LSTM -BlDO	108.4/130.4
(5)	KN,δ	LSTM +BlDO	95.3 /104.5

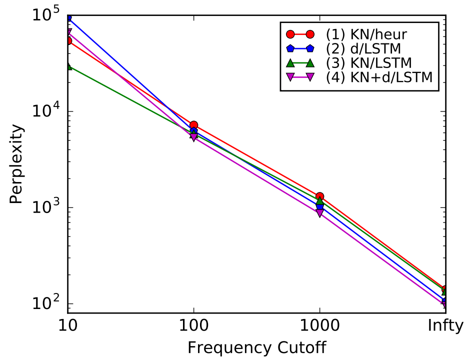

Figure 4: Perplexities of (1) standard n-grams, (2) standard LSTMs, (3) neurally interpolated n-grams, and (4) neural/n-gram hybrids on lower frequency words.

[5]Neurally interpolated n-grams are also more efficient than standard neural LMs, as mentioned in §4.1. While a standard LSTM LM calculated 1.4kw/s on the PTB data, the neurally interpolated models using LSTMs and FF nets calculated 11kw/s and 58kw/s respectively, only slightly inferior to 140kw/s of heuristic KN.

[6]Note that unlike this work, we opt to condition only on in-sentence context, not inter-sentential dependencies, as training through gradient calculations over sentences is more straightforward and because examining the effect of cross-boundary information is not central to the proposed method. Thus our baseline numbers are not directly comparable (i.e. have higher perplexity) to previous reported results on this data, but we still feel that the comparison is appropriate.

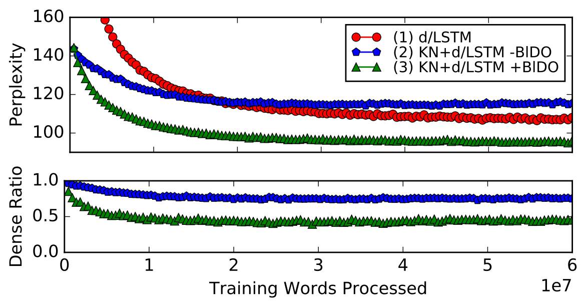

Figure 5: Perplexity and dense distribution ratio of the baseline LSTM LM (1), and the hybrid method without (2) and with (3) block dropout.

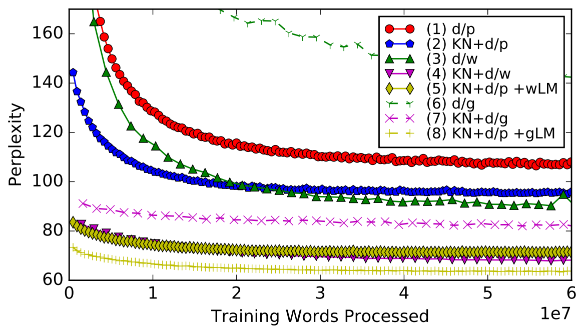

Figure 6: Models trained on PTB (1,2), PTB+WSJ (3,4,5) or PTB+WSJ+GW (6,7,8) using standard neural LMs (1,3,6), neural/n-gram hybrids trained all data (2,4,7), or hybrids trained on PTB with additional n-gram distributions (5,8).

The neurally interpolated n-gram models consistently outperform standard KN-smoothed n-grams, demonstrating their superiority within this model class. In contrast, the neural/n-gram hybrid models tend to follow a pattern more similar to that of LSTM language models, similarly with consistently higher performance.

6.4 Results for Larger Data Sets

To examine the ability of the hybrid models to use counts trained over larger amounts of data, we perform experiments using two larger data sets:

WSJ: The PTB uses data from the 1989 Wall Street Journal, so we add the remaining years between 1987 and 1994 (1.81M sents., 38.6M words).

GW: News data from the English Gigaword 5th Edition (LDC2011T07, 59M sents., 1.76G words).

We incorporate this data either by training net parameters over the whole large data, or by separately training count-based n-grams on each of PTB, WSJ, and GW, and learning net parameters on only PTB data. The former has the advantage of training the net on much larger data. The latter has two main advantages: 1) when the smaller data is of a particular domain the mixture weights can be learned to match this in-domain data; 2) distributions can be trained on data such as Google n-grams (LDC2006T13), which contain n-gram counts but not full sentences.

In the results of Fig. 6, we can first see that the neural/n-gram hybrids significantly outperform the traditional neural LMs in the scenario with larger data as well. Comparing the two methods for incorporating larger data, we can see that the results are mixed depending on the type and size of the data being used. For the WSJ data, training on all data slightly outperforms the method of adding distributions, but when the GW data is added this trend reverses. This can be explained by the fact that the GW data differs from the PTB test data, and thus the effect of choosing domain-specific interpolation coefficients was more prominent.

6.5 Comparison with Static Interpolation

Finally, because the proposed neural/n-gram hybrid models combine the advantages of neural and n-gram models, we compare with the more standard method of training models independently and combining them with static interpolation weights tuned on the validation set using the EM algorithm. Tab. 4 shows perplexities for combinations of a standard neural model (or δ distributions) trained on PTB, and count based distributions trained on PTB, WSJ, and GW are added one-by-one using the standard static and proposed LSTM interpolation methods. From the results, we can see that when only PTB data is used, the methods have similar results, but with the more diverse data sets the proposed method edges out its static counterpart.[7]

[7]In addition to better perplexities, neural/n-gram hybrids are trained in a single pass instead of performing post-facto interpolation, which may give advantages when training for other objectives (Auli and Gao, 2014; Li et al., 2015).

Table 4: PTB perplexity for interpolation between neural (δ) LMs and count-based models.

Interp	δ+PTB	+WSJ	+GW
Lin.	**95.1**	70.5	65.8
LSTM	95.3	**68.3**	**63.5**

7 Related Work

A number of alternative methods focus on interpolating LMs of multiple varieties such as in-domain and out-of-domain LMs (Bulyko et al., 2003; Bacchiani et al., 2006; Gülçehre et al., 2015). Perhaps most relevant is Hsu (2007)'s work on learning to interpolate multiple LMs using log-linear models. This differs from our work in that it learns functions to estimate the fallback probabilities $\alpha_n(\boldsymbol{c})$ in Eq. 3 instead of $\boldsymbol{\lambda}(\boldsymbol{c})$, and does not cover interpolation of n-gram components, non-linearities, or the connection with neural network LMs. Also conceptually similar is work on adaptation of n-gram LMs, which start with n-gram probabilities (Della Pietra et al., 1992; Kneser and Steinbiss, 1993; Rosenfeld, 1996; Iyer and Ostendorf, 1999) and adapt them based on the distribution of the current document, albeit in a linear model. There has also been work incorporating binary n-gram features into neural language models, which allows for more direct learning of n-gram weights (Mikolov et al., 2011), but does not afford many of the advantages of the proposed model such as the incorporation of count-based probability estimates. Finally, recent works have compared n-gram and neural models, finding that neural models often perform better in perplexity, but n-grams have their own advantages such as effectiveness in extrinsic tasks (Baltescu and Blunsom, 2015) and better modeling of rare words (Chen et al., 2015).

8 Conclusion and Future Work

In this paper, we proposed a framework for language modeling that generalizes both neural network and count-based n-gram LMs. This allowed us to learn more effective interpolation functions for count-based n-grams, and to create neural LMs that incorporate information from count-based models.

As the framework discussed here is general, it is also possible that they could be used in other tasks that perform sequential prediction of words such as neural machine translation (Sutskever et al., 2014) or dialog response generation (Sordoni et al., 2015). In addition, given the positive results using block dropout for hybrid models, we plan to develop more effective learning methods for mixtures of sparse and dense distributions.

Acknowledgements

We thank Kevin Duh, Austin Matthews, Shinji Watanabe, and anonymous reviewers for valuable comments on earlier drafts. This work was supported in part by JSPS KAKENHI Grant Number 16H05873, and the Program for Advancing Strategic International Networks to Accelerate the Circulation of Talented Researchers.

References

Waleed Ammar, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2016. One parser, many languages. *CoRR*, abs/1602.01595.

Michael Auli and Jianfeng Gao. 2014. Decoder integration and expected bleu training for recurrent neural network language models. In *Proc. ACL*, pages 136–142.

Michiel Bacchiani, Michael Riley, Brian Roark, and Richard Sproat. 2006. Map adaptation of stochastic grammars. *Computer Speech and Language*, 20(1):41–68.

Paul Baltescu and Phil Blunsom. 2015. Pragmatic neural language modelling in machine translation. In *Proc. NAACL*, pages 820–829.

Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Fréderic Morin, and Jean-Luc Gauvain. 2006. Neural probabilistic language models. In *Innovations in Machine Learning*, volume 194, pages 137–186.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. Large language models in machine translation. In *Proc. EMNLP*, pages 858–867.

Ivan Bulyko, Mari Ostendorf, and Andreas Stolcke. 2003. Getting more mileage from web text sources for conversational speech language modeling using class-dependent mixtures. In *Proc. HLT*, pages 7–9.

Stanley F. Chen and Joshua Goodman. 1996. An empirical study of smoothing techniques for language modeling. In *Proc. ACL*, pages 310–318.

W. Chen, D. Grangier, and M. Auli. 2015. Strategies for Training Large Vocabulary Neural Language Models. *ArXiv e-prints*, December.

Stephen Della Pietra, Vincent Della Pietra, Robert L Mercer, and Salim Roukos. 1992. Adaptive language modeling using minimum discriminant estimation. In *Proc. ACL*, pages 103–106.

Greg Durrett and Dan Klein. 2011. An empirical investigation of discounting in cross-domain language models. In *Proc. ACL*.

Irving J Good. 1953. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237–264.

Çaglar Gülçehre, Orhan Firat, Kelvin Xu, Kyunghyun Cho, Loïc Barrault, Huei-Chi Lin, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2015. On using monolingual corpora in neural machine translation. *CoRR*, abs/1503.03535.

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

Bo-June Hsu. 2007. Generalized linear interpolation of language models. In *Proc. ASRU*, pages 136–140.

Rukmini M Iyer and Mari Ostendorf. 1999. Modeling long distance dependence in language: Topic mixtures versus dynamic cache models. *Speech and Audio Processing, IEEE Transactions on*, 7(1):30–39.

Frederick Jelinek and Robert Mercer. 1980. Interpolated estimation of markov source parameters from sparse data. In *Workshop on pattern recognition in practice*.

Slava M Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3):400–401.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *Proc. ICLR*.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *Proc. ICASSP*, volume 1, pages 181–184. IEEE.

Reinhard Kneser and Volker Steinbiss. 1993. On the dynamic adaptation of stochastic language models. In *Proc. ICASSP*, pages 586–589.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2015. A diversity-promoting objective function for neural conversation models. *CoRR*, abs/1510.03055.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proc. InterSpeech*, pages 1045–1048.

Tomáš Mikolov, Anoop Deoras, Daniel Povey, Lukáš Burget, and Jan Černockỳ. 2011. Strategies for training large scale neural network language models. In *Proc. ASRU*, pages 196–201. IEEE.

Masami Nakamura, Katsuteru Maruyama, Takeshi Kawabata, and Kiyohiro Shikano. 1990. Neural network approach to word category prediction for English texts. In *Proc. COLING*.

Toshiaki Nakazawa, Hideya Mino, Isao Goto, Graham Neubig, Sadao Kurohashi, and Eiichiro Sumita. 2015. Overview of the 2nd Workshop on Asian Translation. In *Proc. WAT*.

Hermann Ney, Ute Essen, and Reinhard Kneser. 1994. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech and Language*, 8(1):1–38.

Vu Pham, Théodore Bluche, Christopher Kermorvant, and Jérôme Louradour. 2014. Dropout improves recurrent neural networks for handwriting recognition. In *Proc. ICFHR*, pages 285–290.

Ronald Rosenfeld. 1996. A maximum entropy approach to adaptive statistical language modelling. *Computer Speech and Language*, 10(3):187–228.

Holger Schwenk. 2007. Continuous space language models. *Computer Speech and Language*, 21(3):492–518.

Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. 2015. A neural network approach to context-sensitive generation of conversational responses. In *Proc. NAACL*, pages 196–205.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM neural networks for language modeling. In *Proc. InterSpeech*.

Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. 2014. Sequence to sequence learning with neural networks. In *Proc. NIPS*, pages 3104–3112.

Yee Whye Teh. 2006. A Bayesian interpretation of interpolated Kneser-Ney. Technical report, School of Computing, National Univ. of Singapore.

Will Williams, Niranjani Prasad, David Mrva, Tom Ash, and Tony Robinson. 2015. Scaling recurrent neural network language models. In *Proc. ICASSP*.

Ian H. Witten and Timothy C. Bell. 1991. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *CoRR*, abs/1409.2329.