Error-Correcting Neural Sequence Prediction

James O' Neill [* 1]   Danushka Bollegala [1]

Abstract

In this paper we propose a novel neural language modelling (NLM) method based on *error-correcting output codes* (ECOC), abbreviated as ECOC-NLM. This latent variable based approach provides a principled way to choose a varying amount of latent output codes and avoids exact softmax normalization. Instead of minimizing measures between the predicted probability distribution and true distribution, we use error-correcting codes to represent both predictions and outputs. Secondly, we propose multiple ways to improve accuracy and convergence rates by maximizing the separability between codes that correspond to classes proportional to word embedding similarities. Lastly, we introduce a novel method called *Latent Mixture Sampling*, a technique that is used to mitigate exposure bias and can be integrated into training latent-based neural language models. This involves mixing the latent codes (i.e variables) of past predictions and past targets in one of two ways: (1) according to a pre-defined sampling schedule or (2) a differentiable sampling procedure whereby the mixing probability is learned throughout training by replacing the greedy argmax operation with a smooth approximation. In evaluating Codeword Mixture Sampling for ECOC-NLM, we also baseline it against CWMS in a closely related Hierarhical Softmax-based NLM.

1. Introduction

Language modelling (LM) is a fundamental task in natural language that requires a parametric model to generate tokens given past tokens. LM underlies all other types of structured modelling tasks in natural language, such as Named Entity Recognition, Constituency/Dependency Parsing, Coreference Resolution, Machine Translation (Sutskever et al., 2014) and Question Answering (Mikolov et al., 2010). The goal is to learn a joint probability distribution for a sequence of length T containing words from a vocabulary $\mathcal{V}$. This distribution can be decomposed into the conditional distributions of current tokens given past tokens using the chain rule, as shown in Equation 1. In Neural Language Modelling (NLM), a Recurrent Neural Network (RNN) $f_\theta(\cdot)$ parameterized by θ is used to encode the information at each timestep t into a hidden state vector h_t^l which is followed by a decoder $z_t^l = h_t^l W^l + b^l$ and a normalization function $\phi(z_t^l)$ which forms a probability distribution $\hat{p}_\theta(y_t|x_t, h_{t-1}), \ \forall t \in T$.

$$P(w_1, ..., w_T) = \prod_{t=1}^{T} P(w_t|w_{t1}, ..., w_1) \qquad (1)$$

However, training can be slow when $|\mathcal{V}|$ is large while also leaving a large memory footprint for the respective input embedding matrices. Conversely, in cases where the decoder is limited by an information bottleneck (Yang et al., 2017), the opposite is required where more degrees of freedom are necessary to alleviate information loss in the decoder bottleneck. Both scenarios correspond to a trade-off between computation complexity and out-of-sample performance. Hence, we require that a newly proposed model has the property that the decoder can be easily configured to deal with this trade-off in a principle way.

Lastly, standard supervised learning (self-supervised for language modelling) assumes inputs are i.i.d. However, in sequence prediction, the model has to rely on its own predictions at test time, instead of past targets that are used as input at training time. This difference is known as *exposure bias* and can lead to errors compounding along a generated sequence. This approach to sequence prediction is also known as *teacher forcing* where the teacher provides targets that are used at training time. We also require that exposure bias is addressed in our approach while dealing with the aforementioned challenges related to computation and performance tradeoffs in the decoder.

We propose an error-correcting output code (ECOC) based NLM (ECOC-NLM) that address this desiderata. In the approximate case where codeword dimensionality $|c| < |\mathcal{V}|$, we show that that given sufficient error codes ($|\mathcal{V}| \ll |c| \gg \log_2(|\mathcal{V}|)$), we maintain accuracy compared to traditional NLMs that use the full softmax and other approximate methods. Lastly, we show that this latent-based NLM approach can be extended to mitigate the aforementioned problem of compounding errors by using *Latent Mixture Sampling*

(LMS). LMS in an ECOC-NLM model also outperforms an equivalent Hierarchical Softmax-based NLM that uses Scheduled Sampling (Bengio et al., 2015) and other closely related baselines. To our knowledge, this is the first latent-based technique to mitigating compounding errors in recurrent neural networks (RNNs).

Our main contributions are summarized as the following:

- An error-correcting output coded neural language model that requires less parameters than its softmax-based language modelling counterpart given sufficient separability between classes via error-checks.

- An embedding cosine similarity rank ordered codebook that leads to well-separated codewords.

- A *Latent-Mixture Sampling* method to mitigate exposure bias in latent-variable models. This is then extended to *Differentiable Latent Mixture Sampling* that uses the Gumbel-Softmax so that discrete categorical variables can be backpropogated through.

- Novel baselines such as Scheduled Hierarchical Sampling (SS-HS) and Scheduled Adaptive Sampling (SS-AS), are introduced in the evaluation of our proposed ECOC method. This applies scheduled sampling to two closely related softmax approximation methods.

2. Background

2.1. Error-Correcting Codes

Error-Correcting Codes (Hamming, 1950) originate from seminal work in solid-state electronics around the time of the first digital computer. Later, binary codes were introduced in the context of artificial intelligence via the NETtalk system (Sejnowski & Rosenberg, 1987), where each class index is represented by their respective binary code C. A parametric model $f_\theta(\cdot)$ is then used to predict a probability for each n binary bit positions being active or not. This results in a predicted $\hat{C}$ which can then be measured against the ground truth C. At training time we optimize for some objective $\mathcal{L}(C, \hat{C})$. At test time we choose the codeword that is closest to the predicted code, allowing for Hamming distances between codewords and the error-correcting codes. When $|\mathcal{V}| \neq 2^n$, the remaining codes are used as error-correction bits $k = |\mathcal{V}| - 2^n$. This tolerance can be used to account for the information loss due to the sample size by measuring the distance (e.g Hamming) between the predicted code word and the true codeword with d error-correction bits. If the minimum distance between codewords is d then at least $(d-1)/2$ bits can be corrected for and hence, if the Hamming distance $d \leq (d-1)/2$ we will still retrieve the correct codeword. In contrast to using one bit per k classes

in standard multi-class classification, error-correction cannot be achieved. Both error-correction bits and class bits make up the *codebook* $\mathcal{C}$.

In order to achieve good separation between k classes (i.e codewords that are assigned so that it less likely to make mistakes due close Hamming distances), the codes should have good row and column separation. The former refers to having equidistant Hamming distances, where remaining codes are the error-correcting codes. Column separation refers to ensuring that the functions for each bit-position are uncorrelated with one another. This can be achieved by maximizing the Hamming distance between the columns, similar to row-separation. A primary aim in ECOC is that the bit errors are uncorrelated and that likelihood of simultaneous errors occurring is low. This property makes it easier for error-correcting codes to re-correct errors. If many simultaneous errors are made, this becomes more difficult.

ECOCs have been used for multi-class document classification (Berger, 1999). The authors also propose a coding theory argument as to why randomly assigned codes can result in well-separated codes. However, they make the strong assumption of class independence which is weak for typical natural language problems. This work addresses by using semantically-driven separation for language modelling.

ECOC can be considered an ensembled method for multi-classification since each model needs to make a prediction for each binary unit for b binary codes on the output (similar to Bagging in ensemble learning), albeit a distinct code that is being predicted in binary classification. Kong & Dietterich (1995) have shown that this distinction over voting methods leads to variance reduction and bias-correction in each respective ECOC classifier. This is different to regular multi-class classification where one prediction is made from a distribution over k classes and $\log_2(k) \leq b \leq k$.

2.2. *Why Latent Codes for Neural Language Modelling?*

Targets in standard training of NLMs are represented as 1-hot vectors (i.e kronecker delta) and the problem is treated as a 1-vs-rest multi-class classification. This can be considered a special case of ECOC classification where the codebook $\mathcal{C}$ with n classes is represented by an identity $\mathbb{I}^{n \times n}$. ECOC classification is well suited over explicitly using observed variables when the output space is structured. Flat-classification (1-vs-rest) ignores the dependencies in the outputs, in comparison to using latent codes that share some common latent variables between associative words. For example, in the former case, if we train a model that only observes the word "silver" in a sequence "...silver car.." and then at test-time observes "silver-back", because there is high association between "silver" and "car", the model is more likely to predict "car" instead of "gorilla". ECOC is less prone to such mistakes because although a/some bit/s

may be different between the latent codes for "car" and "gorilla", the potential misclassifications can be re-corrected with the error-correcting bits. In other words, latent coding can reduce the variance of each individual classifier and has some tolerance to mistakes induced by sparse transitions, proportional to the number of error-checks used. Furthermore, 1-vs-rest classification requires $n-1$ class boundaries be learned at once, whereas in ECOC we must only build class boundaries for $\log_2(|\mathcal{V}|) - 1 \leq n \leq |\mathcal{V}| - 1$, typically closer to the lower-bound of this interval. In fact, in the case of language modelling where n is commonly large (e.g $n > 10^5$), the boundary is learned multiple times and therefore is more likely to recover from mistakes, in the same way ensembles reduce variance in prediction (Kong & Dietterich, 1995).

2.3. Methods for Softmax Approximation

2.3.1. LOSS-BASED METHODS

Hierarchical Softmax (HS) Goodman (2001); Morin & Bengio (2005) propose to use short codes that represent marginals of the conditional distribution, where the product of these marginals that are gotten along a path in the tree approximate the conditional distribution. This speeds up training by summing over the paths of a binary tree where intermediate nodes assign relative probabilities of child nodes. Therefore, only few sums are necessary, along the binary path to a given leaf (i.e word). The probability for an embedded word vector v_Γ at an intermediate node Γ is the product of taking left or right turn at every intermediate node. The probability of transitioning right is $p(r \to w_s | n) = \sigma(v_\Gamma'^T v_{ws})$. The conditional $p(w_t | w_s; \theta)$ is then $\prod_{k=1}^{L(w_t)} \sigma\big(\mathbb{I}_b(\Gamma(w_t, k), \Gamma(w_t, k+1)) \cdot v_{\Gamma(w_t,k)^T v_{ws}}'\big)$ where $L(w_t)$ is the path depth to w_t, I_b is a decision point along a path that transitions to the left child $n(w_0, k+1)$ or the right child $n(w_t, k)$. Defining a good tree structure improves performance since semantically similar words have a shorter path and therefore similar representations are learned for similar words. This can be achieved by either clustering words via term frequency using a Huffman Tree (Mikolov et al., 2013) or using already defined word groups from semantic networks (Morin & Bengio, 2005) such as Word-Net. As we will discuss in section 4, we also build upon HS by proposing a method that interpolates between predicted codes and target codes to make the model more robust to its own errors and use this as a reasonable baseline for ECOC-NLM that also integrates this method into training.

Differentiated Softmax (DS) uses a sparse linear block of weights for the decoder where a set of partitions are made according to the unigram distribution, where the number of weights are assigned proportional to the term frequency. This intuitive since rare words require less degrees of freedom to account for the little amount of contexts in which

they appear, in comparison to common words. For a sparse decoder weight matrix $\mathbb{R}^{k \times d}$, each p partition has dimensionality d_p where for common words $d_p \approx d$ and rare words $d_p \ll d$. Both the number of partitions and the dimensionality $d_p \forall p$ can be tuned at training time.

Adaptive Softmax (AS) Grave et al. (2016) provide an approximate hierarchical model that directly accounts for the computation time of matrix multiplications. AS results in 2x-10x speedups when compared to the standard softmax, dependent on the size of the corpus and vocabulary. Interestingly, they find on sufficiently large corpora (Text8, Europarl and 1-Billion datasets), they manage to maintain accuracy while reducing the computation time.

2.3.2. SAMPLING-BASED METHODS

Importance Sampling (IS) is a classical Monte-carlo sampling method used to approximate probability distributions. A proposal distribution Q is used to approximate the true distribution P which allows us to draw Monte-Carlo samples at much less cost than if we were to draw directly from P. Q is often chosen to be simple and close to P. In language modelling, it is common that the unigram distribution is used for Q. The expectation $\mathbb{E}_{w_i \sim P[\nabla_\theta \varphi(w_i)]} \approx \sum_{i=1}^m \nabla_\theta \varphi(w_i)$ over sampled word w_i that is approximately the gradient of $\varphi(w_i)$. However, computing $p(w)$ for each sample is still required and therefore there have been methods to compute the product of marginals that avoids expensive normalization over the MC samples (Bengio et al., 2003a). **Adaptive Importance Sampling** (Jean et al., 2014) (AIS) only considers a set fraction of target tokens to sample from. This involves partitioning the training set where each partition is designated a subset of the vocabulary $V' \subset V$. Therefore, there is a separate predictive distribution for each partition and for a given partition V' all words within V' are assigned some probability.

Noise Contrastive Estimation (Mnih & Teh, 2012) propose to use Noise Contrastive Estimation (NCE) (Gutmann & Hyvärinen, 2010) as a sampling method in an unnormalized probabilistic model that is more stable than IS (can lead to diverging model distribution Q in relation to underlying distribution P). Similar to our proposed ECOC-NLM, NCE treats density estimation as multiple binary classification problems, but different in that it does so by discriminating between both data samples and a known noise distribution (e.g noise proportional to the unigram distribution). The posterior can be expressed in the context independent case as $P^h(D = 1 | w, \theta) = P_\theta^h(w) / \big(P_\theta^h(w) + k P_n(w)\big)$ for hidden state h, parameterized by θ for k generated samples. NCE is different from IS since it does not estimate the word probabilities directly, but instead uses an auxiliary loss that maximizes the probability of the correct words from noisy samples.

2.4. Recent Applications of Latent Codes

Shu & Nakayama (2017) recently used compositional codes for word embeddings to cut down on memory requirements in mobile devices. Instead of using binary coding, they achieve word embedding compression using multi-codebook quantization. Each bit $c \in C$ comprises of a discrete code (0-9) and therefore at minimum $\log_{10}(k)$ bits are required. They too propose to use Gumbel-Softmax trick but for the purposes of learning the discrete codes. The performance was maintained for sentiment analysis and machine translation with 94% and 98% respective compression rates. Shi & Yu (2018) propose a product quantizatioon structured embedding that reduces memory by 10-20 times the number of parameters, while maintaining performance. This involves slicing the embedding tensor into groups which are then quantized independently. The embedding matrix is represented with an index vector and a codebook tensor of the quantized partial embedding. Zhang et al. (2013) propose a weighted Hamming Distance to avoid ties in ranking Hamming Distances which is common, particularly for short codes. This is relevant to our work as well in the context of assigning error-checking bits by Hamming distance to codewords that correspond to classes in $\mathcal{V}$.

3. Codebook Construction

A challenging aspect of assigning codewords is ordering the codes such that if errors made that the resulting incorrect codeword is at least, semantically closer to that of the codewords that are less related, while ensuring good separation between codes. Additionally, we have to consider the amount of error-checking bits to use. In theory, $\log_2(k)/k$ is sufficient to account for all k classes. However, this alone can lead a degradation in performance. Hence, we also consider a large amount of error-checking bits. In this case, the error-checking bits can account for more mistakes given by other classes, which may be correlated. In contrast, using probability distributions naturally account for these correlations, as the mass needs to shift relative to the activation of each output. This point is particularly important for language modelling because of the high-dimensionality of the output. The most naive way to create the codebook is to simply assign binary codes to each word in random order. However, it is preferable to assign similar codes to words in the vocabulary that are semantically similar while maximizing the Hamming distance between codes where leftover error codes separate class codes.

3.1. Codebook Arrangement

A fundamental challenge in creating the codebook $\mathcal{C}$ is in how error-codes are distributed between codes that maximize the separability between codewords that are more likely to be interchangeably and incorrectly predicted. This is related to the second challenge of choosing the dimensionality of C. The latter is dependent on the size of the corpus, and in some cases might only require $|\log_2(\mathcal{V})| \leq d \leq |\mathcal{V}|$ bits to represent all classes with leftover error-checking bits. These two decisions correspond to a tradeoff between computational complexity and accuracy of our neural language model, akin to tree expressivity in the Hierarhcial Softmax to using the Full Softmax. Below we describe a semantically motivated method to achieve well-separated codewords, followed by a guide on how to choose codebook dimensionality $d_{\mathcal{C}}$.

3.1.1. EMBEDDING SIMLARITY-BASED CODEBOOKS

Previous work on ECOC has focused on theories as to why randomly generated codes lead to good row and column separation (Berger, 1999). However, this assumes that class labels are conditionally independent and therefore it does not apply well for language modelling where the output space is loosely structured. To address this, we propose to reorder $C \in \mathcal{C}$ such that Hamming distance between any two codewords is proportional to the embedding similarity. Moreover, separating codewords by semantic similarity can be achieved by placing the amount of error-checking bits proportional to rank ordered similarity for a chosen query word embedding. A codebook ordered by embedding similarity for w_* is denoted as $\mathcal{C}_{\Lambda_{w_*}}$. The similarity scores between embeddings is given as $\mathcal{F}(\mathcal{M}_*, M_i) \; \forall i$ is used reorder $M \to \mathcal{M}'$. Good separation is achieved when codes are separated proportional to the cosine similarity between embeddings of the most frequent word $w_* \in \mathcal{V}$ and the remaining words w'. Therefore, words with high similarity have corresponding codes that are closer in Hamming distance $H(\cdot, \cdot)$ in $\mathcal{C}$. This ensures that even when codes are correlated, that incorrect latent predictions are at least more likely to correspond to semantically related words. We are not guaranteed that codes close in Hamming distance are closer in a semantic sense in the random case. Given k redundant codewords C_k, we require an assignment that leads to a strongly separated $\mathcal{C}$. Let $\delta(\cdot, \cdot)$ denote a function that assigns C_i^k error-checking codewords assigned to the ith class codeword and $\delta(C_*, C_i) \propto \mathcal{F}(\mathcal{M}_*, \mathcal{M}_i) \forall k$. In practice $\delta(\cdot, \cdot)$ normalizes the resulting embedding similarities $S = \mathcal{F}(M_*, M)$ using a normalization function $\text{cumsum}(\phi(S) \times |\mathcal{C}|)$ to assign the intervals between adjacent codeword spans. This results in greater distance between words that are more similar to $w*$, and less error-checking codewords to relatively rarer words that tend to have little neighbouring words in the embedding space.

3.1.2. RANDOM CODEBOOKS

Berger (1999) find that a well row-separated binary $\mathcal{C}$ can be defined as one where all rows have a relative Hamming sep-

aration at least $\frac{1}{2}4\sqrt{\frac{\log m}{n}}$. The probability that a randomly-constructed binary matrix $\mathcal{C}$ is not well row-separated is at most $1/m^4$. Further, it holds that for any two rows in a well-separated $\mathcal{C}$ which have a relative Hamming separation in the range, $\left[\frac{1}{2} - 4\sqrt{\frac{\log m}{n}}, \frac{1}{2} + 4\sqrt{\frac{\log m}{n}}\right]$ the probability that a randomly constructed $\mathcal{C}$ is not strongly well-separated is at most $2/m^4$. We consider these random codebooks as one of the baselines when evaluating ECOC against other related approximate methods in NLM.

4. Latent Mixture Sampling

To mitigate exposure bias for latent-based language modelling we propose a sampling strategy that interpolates between predicted and target codewords. We refer to this as Latent Mixture Sampling (LMS) and its application to ECOC as *Codeword Mixture Sampling* (CMS).

4.1. Curriculum-Based Latent Mixture Sampling

In Curriculum-Based Latent Mixture Sampling (CLMS), the mixture probability is $p_c = 0 \; \forall c \in C$ at epoch $\epsilon = 0$ and throughout training the probability monotonically increases $p_c = \delta_c \; \forall c \in C$ where δ_c is the threshold for the c th bit after ϵ epochs. A Bernoulli sample $\tilde{C} = \mathbb{B}(\hat{C}_t, C_t) \forall c \in C$ is carried out for each timestep t in each minibatch. The probabilities per dimension p_c are independent of keeping a prediction $\hat{y}_{t-1,c}$ instead of the c th bit in the target codeword $y_{(t-1,c)}$ at timestep t-1. The reason for having individual mixture probabilities per bits is because when we consider a default order in $\mathcal{C}$, this results in tokens being assigned codewords ranked by frequency. Therefore, the leftmost bit predictions are more significant than bit errors near the beginning (e.g $2^0 = 1$ only 1 bit difference). In this paper we report results when using a sigmoidal schedule as shown in Equation 2 where τ_{max} represents the temperature at the last epoch and δ is a scaling factor that controls the slope of the sigmoid (in our experiments $\delta = 0.1$).

$$[\hat{y}_{t-1}, y_{t-1}] \sim \frac{\tau_{max}}{1 + \exp(-\epsilon/\delta)} \quad \forall \epsilon \in [-N/2, N/2] \quad (2)$$

4.2. Latent Soft-Mixture Sampling

In standard CMS, we pass token index w_t which is converted to an input embedding e_w based on the most probable bits predictions at the last time step $\arg\max_\theta p(y_{t-1}|x_{t-1}; \theta)$. We can instead replace the argmax operator with a soft argmax that uses a weighted average of embeddings $e \in E$ where weights are assigned from the previous predicted output via the softmax normalizatio $\phi(x_{t-1}, \tau)$, where τ controls the kurtosis of the probability distribution ($\tau \to 0$ tends to the argmax), as shown in Equation 3.
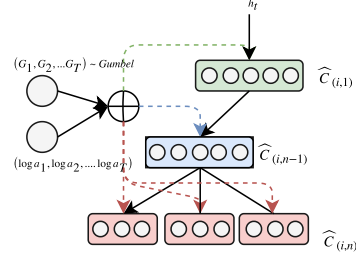

Figure 1. Differentiable Latent Mixture Sampling

$$x_t = \sum_{w \in \mathcal{V}} e_w \left(\frac{\exp(h_w^T \theta / \tau)}{\sum_{w \in \mathcal{V}} \exp(h_w^T \theta / \tau)} \right) \quad (3)$$

In the ECOC-NLM, we consider binary codewords and therefore choose the top k least probable bits to flip according to the curriculum schedule. Hence, this results in k codewords where each $\hat{C}$ has at least hamming distance $\text{H}(\hat{C}, C) = 1 \; (2^0)$. Concretely, this is a soft interpolation between past targets and a weighted sum of the k most probable codewords $\hat{C}_K = \arg\max_k \left(\sigma(h_w^T W) \right)$ such that $x_t = \mathbb{B}_K \left(C, \sum_k^K \phi(\hat{C}_k) \right)$ where B_K samples one or the other for each k dimension of C.

5. Differentiable Latent Sampling

The previous curriculum strategies disregard where the errors originate from. Instead, they interpolate between model predictions of latent variables $\hat{Y}$ and targets Y in a way that does not distinguish between cascading errors and localized errors. This means that it only recorrects errors after they are made instead of directly correcting for the origin of the errors. Maddison et al. (2016) showed that such operations can be approximated by using a continuous relaxation using the reparameterization trick, also known as the *Concrete Distribution*. By applying such relaxation it allows us to sample from the distribution across codes while allowing for a fully differentiable objective, similar to recent work (Goyal et al., 2017). We extend this to mixture sampling by replacing the argmax operation with the Concrete distribution to allow the gradients to be adjusted at points where prior predictions changed value throughout training. This not only identifies at which time-step the error occurs, but what latent variables (i.e output codes) had the most influence in generating the error. This is partially motivated by the finding that in the latent variable formulation of simple logistic regression models, the latent variable errors form a Gumbel distribution. Hence, sampling latent codes inversely proportional to the errors from a Gumbel distribution would seem a suitable strategy.

Gumbel-Softmax Similarly, instead of passing the most

likely predicted word $\hat{y}^{w*}_{t-1}$, we can instead sample from $\hat{y}_{t-1} \sim \phi(h_{t-1}, w)$ and then pass this index as $\hat{x}_t$. This is an alternative to always acting greedily and allow the model to seek other likely actions. However, in order to compute derivatives through samples from the softmax, we need avoid discontinuities, such as the argmax operation. The Gumbel-Softmax (Maddison et al., 2016; Jang et al., 2016) allows us to sample and differentiate through the softmax by providing a continuous relaxation results in probabilities instead of a step function (i.e argmax). For each componentwise Gumbel noise k for latent variable z, we find k that maximizes $\log \alpha_k - \log(-\log U_k)$ and then set $D_k = 1$ and $D \neg k = 0$, where $U_k \sim \text{Uniform}(0, 1)$ and α_k is drawn from a discrete distribution $D \sim \text{Discrete}(\alpha)$.

$$\hat{p}(y_t | x_t; \theta) = \prod_{d=1}^{D} \frac{\exp((\log \alpha_k + G_k)/\tau)}{\sum_{i=1}^{n} \exp((\log \alpha_i + G_i)/\tau)} \quad (4)$$

For ECOC, we instead consider Bernoulli random variables which for the Concrete distribution can be expressed by means of two arbitrary Gumbel distributions G_1 and G_2. The difference between G_1 and G_2 follows a Logistic distribution and so $G_1 - G_2 \sim \text{Logistic}$ and is sampled as $G_1 - G_2 \equiv \log U - \log(1 - U)$. Hence, if $\alpha = \alpha_1/\alpha_2$, then $P(D_1 = 1) = P(G_1 + \log \alpha_1 > G_2 + \log \alpha_2) = P(\log U \log(1 - U) + \log \alpha > 0)$. For a step function $\mathcal{H}$, $D_1 \equiv \mathcal{H}(\log \alpha + \log U - \log(1 - U))$, corresponding to the Gumbel Max-Trick (Jang et al., 2016). The sampling process for a Binary Concrete random variable involves sampling Z, sample $L \sim \text{Logistic}$ and set Z as shown in Equation 5, where $\alpha, \tau \in (0, \infty)$ and $Z \in (0, 1)$. This Binary Concrete distribution is henceforth denoted as $\text{BinConcrete}(\cdot, \cdot)$ with location α and temperature τ.

$$Z \equiv \frac{1}{1 + \exp\left((\log \alpha + L)/\tau\right)} \quad (5)$$

This is used for ECOC and other latent variable-based models, such as Hierarchical Sampling, to propogate through past decisions and make corrective updates that backpropogate to where errors originated from along the sequence. Hence, we also carry out experiments with BinConcrete (Equation 5) and Gumbel-Softmax(Equation 4) for HS and ECOC. The temperature τ can be kept static, annealed according to a schedule or learned during training, in the latter case this is equivalent to entropy regularization (Szegedy et al., 2016; Grandvalet & Bengio, 2005)that controls the kurtosis of the distribution. In this work, we consider using an annealed τ, similar to Equation 2 where $\tau \to 2.5$ and starts with $\tau = 0.01$. This is done to allow the model to avoid large gradient variance in updates early on. In the context of using the Gumbel-Softmax in LMS, this allows

the model to become robust to non-greedy actions gradually throughout training, we would expect such exploration to improve generalization proportional to the vocabulary size.

6. Experimental Setup

We carry out experiments for a 2-hidden layer Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) model with embedding size $|e| = 400$, Back-propogation Through Time (BPTT) length 35 and variational dropout (Gal & Ghahramani, 2016) with rate p_d=0.2 for input, hidden and output layers. The ECOC-NLM model is trained using Binary Cross Entropy loss $\mathcal{L}_\theta = \max_k \prod_c^C \left[y_c \log \left(\sigma_c(h^T\theta) + (1 - y_c) \log \left(1 - \sigma_c(h^T\theta)\right)\right)\right]$ for k error-checking codewords, with respective gradients $\frac{\delta\mathcal{L}}{\delta\theta} = (y\sigma(h^T\theta)) \cdot h^T$.

Baselines for ECOC-Neural Language Model The first set of experiments include comparisons against the most related baselines, which include Sample-Softmax (Bengio et al., 2003b; Bengio & Senécal, 2008),Hierarchical Softmax (HS), AS (Grave et al., 2016), and NCE (Mnih & Teh, 2012). For HS, we use a 2-hidden layer tree with a branching factor (number of classes) of $\sqrt{|\mathcal{V}|}$ by default. For AS, we split the output into 4 groups via the unigram distribution (in percentages of total words 5%-15%-30%-100%). For NCE, we set the noise ratio to be 0.1 for PTB and 0.2 for WikiText-2 and WikiText-103. Training is carried out until near convergence ($\epsilon \approx 40$), the randomly initialized HS and Sampled Softmax of which take longer ($\epsilon \in$ [55-80]). Table 1 reports the results for $\log_2 |\mathcal{V}|^2$ number of samples in the case of Rand/Unigram-Sample-SM. For Rand/Unigram Hierarchical SM, we use a 2-hidden layer tree with 10 classes per child node.

Baselines ECOC Mixture Sampling To test Latent Mixture Sampling (LMS), we directly compare its application in HS and ECOC, two closely related latent NLM methods. Additionally, we compare the performance of LMS against the most related sampling-based supervised learning technique called scheduled sampling (SS) (Bengio et al., 2015). For SS with cross-entropy based training (SS-CE), we also consider using a baseline of the soft-argmax (Soft-SS-CE) where a weighted average embedding is generated proportional to the predicted probability distribution.

Evaluation Details In order to compute perplexities for ECOC-NLM, we must view the codewords in terms of a product of marginal probabilities. At training time we choose the most confident prediction within the span of k error checks for a codeword C_k such that $p_{\hat{C}} = \max_k \prod_{c_i}^{C_i} \left(\phi(x_t, h_t)\right)$, $i = [0, 1.., k]$ i.e among the error checks corresponding to a particular token, we choose the most probable of these checks as the value when computing the binary cross-entropy loss. At test time, if the predicted

codeword $\hat{C}$ falls within the k error-checking bits of codeword C, then it is deemed a correct prediction and assigned the highest probability of all k predictions. We note that we only convert the ECOC predictions to perplexities to be comparable against baselines. ECOCs can also be evaluated using Hamming Distance or Mean Reciprocal Rank when the codes are ordered semantically or by Hamming distance (i.e Unigram-ECOC or Embedding-ECOC).

7. Results

7.1. Error-Correcting Output Coded NLM Results

We first compare our proposed ECOC-NLM to aforementioned methods that approximate softmax normalization, using binary trees and latent codes that are ordered according to unigram frequency (Unigram-Hierarchical-SM and Unigram-ECOC). This is also the same ordering we use to compare our proposed CMS-ECOC sampling method to scheduled sampling (Bengio et al., 2015) in standard cross-entropy training with softmax normalization. Although, these are not directly comaprable, since ECOC-NLM introduces a whole new paradigm, we use the common evaluation measures of Hamming distance and accuracy to have some kind of baseline with which we can compare our proposed method to. Figure 2 shows how the reduction in perplexity as the number of ECOC-LSTM decoder parameters increase as more bits are added to the codeword. For PTB, large perplexity reductions are made between 14-100 codebits, while between 100-1000 codebits there is a gradual decrease. In contrast, we see that there is more gained from increasing codeword size for WikiText-2 and WikiText-103 (which preserve the words that fall within the long-tail of the unigram distribution). We find the discrepancy in performance be-
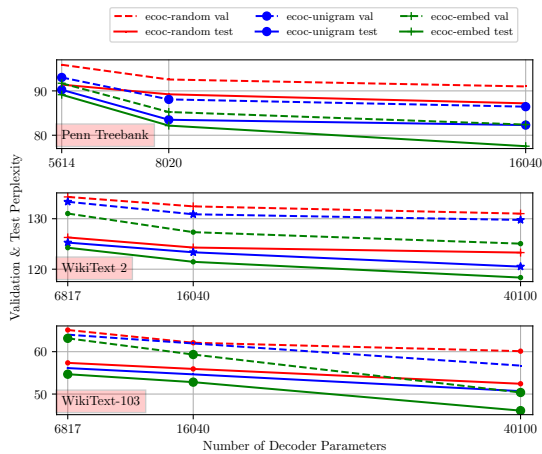
tween randomly assigned codebooks and ordered codebooks is more apparent for large compression ($|C| < |\mathcal{V}|/10$). Intuitively, the general problem of well-separated codes is alleviated as more bits are added.

Table 1 shows that overall ECOC with a rank ordered embedding similarity $\mathcal{C}$ (Embedding-ECOC) almost performs as well as the full-softmax (8.02M parameters) while only using 1000 bits for PTB ($|\mathcal{V}|/20$ and) and 5K bits for WikiText-2 ($|\mathcal{V}|/25$) and WikiText-103 ($|\mathcal{V}|/30$). The HS-based models use a 2-hidden layer tree with 10 tokens per class, resulting in 4.4M parameters for PTB, 22.05M parameters for WikiText-2 (full softmax - 40.1M) and WikiText-103. Moreover, we find there is a consistent improvement in using Embedding-ECOC over using a random codebook (Random-ECOC) and a slight improvement over using a unigram ordered codebook (Unigram-ECOC). Note that in both Embedding-ECOC and Unigram-ECOC, the number of error-checking bits are assigned inversely proportional to the rank position when ordering embedding similarities (as discussed in subsubsection 3.1.1) and unigram frequency respectively. We also found that too many bits e.g $|C| = |\mathcal{V}|$ takes much longer ($\epsilon \in$ [20-30] more for PTB) to converge with negligible perplexity reductions. Hence, the main advantage of ECOC-NLMs is the large compression rate while maintaining performance (e.g PTB with $|C| = 40$, there is less than 2 perplexity points compared to the full softmax).

7.2. Latent Mixture Sampling Results

Figure 3 shows how validation perplexity on WikiText-2 changes throughout training an LSTM as τ begins to tend to τ_{max}=2.5, τ_{max}=10 and the case where τ_{const}=1 is kept constant. We see that too much exploration (τ_{max}=10) leads to an increase in perplexity, as $\tau > 5$, the validation perplexity begins to rise. In contrast, we find a slow monotonic increase to τ_{max}=2.5 leads to a steady increase, at which $\tau = 2$ (epoch 24) the model has almost converged. Table 2 shows all results of LMS when used in HS and ECOC-based

Figure 2. ECOC-NLM Performance with Increasing Number of Decoder Parameters (corresponding to 14/20/40 codeword bits for Penn-TreeBank and 17/40/100 codeword bits for WikiText-2/103)
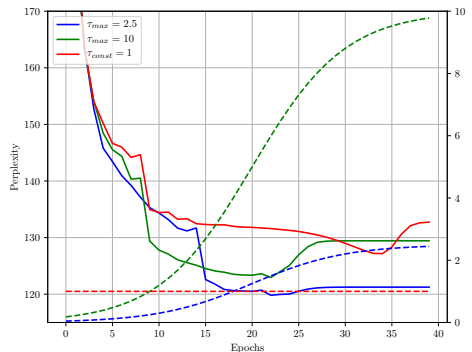
Figure 3. WikiText-2 Validation Perplexity When Varying τ (corresponding dashed lines) in CLMS-ECOC (LSTM)

Model	PTB		WikiText-2		WikiText-103	
	Val.	Test	Val.	Test	Val.	Test
Full SM						
GRU	85.49	78.81	126.28	122.66	59.23	51.44
LSTM	*86.19*	*79.24*	*124.01*	*119.30*	*56.72*	*49.35*
Rand-Sample-SM						
GRU	94.42	83.79	138.91	131.48	70.08	60.80
LSTM	92.14	81.82	136.47	129.29	68.95	59.34
Unigram-Sample-SM						
GRU	91.23	82.45	134.49	128.29	67.10	57.62
LSTM	90.37	81.36	133.08	127.19	66.23	57.09
Rand-Hierarchical-SM						
GRU	96.83	89.93	134.11	127.88	65.01	55.79
LSTM	94.31	88.50	133.69	127.12	62.29	54.28
Unigram-Hierarchical-SM						
GRU	94.35	87.67	131.34	124.91	63.18	54.67
LSTM	92.38	86.70	130.26	124.83	62.02	54.11
Adaptive-SM						
GRU	92.11	85.74	129.90	122.26	60.95	53.03
LSTM	91.38	85.29	118.89	120.92	60.27	52.63
NCE						
GRU	98.62	92.03	131.34	126.17	62.68	54.90
LSTM	96.79	89.30	131.20	126.82	61.11	54.52
Random-ECOC						
GRU	92.47	87.28	132.61	124.22	61.33	52.80
LSTM	91.00	87.19	131.01	123.29	56.12	52.43
Unigram-ECOC						
GRU	87.43	80.39	127.79	120.97	58.12	51.88
LSTM	*86.44*	*82.29*	*129.76*	*120.51*	*52.71*	*48.37*
Embedding-ECOC						
GRU	86.03	80.45	127.40	122.01	58.28	51.67
LSTM	*84.40*	*77.53*	*125.06*	*120.34*	*57.37*	*49.09*

Table 1. Perplexities for Full Softmax (SM), Sample-Based SM (Sample-SM), Hierarchical-SM (HSM), Adaptive-SM, NCE and ECOC-NLM.

Model	PTB		WikiText-2		WikiText-103	
	Val.	Test	Val.	Test	Val.	Test
SS-SM						
GRU	82.49	75.36	123.39	119.71	57.22	49.39
LSTM	81.17	75.24	124.01	119.30	56.72	49.35
Soft-SS-SM						
GRU	78.23	70.60	120.34	116.04	54.49	46.59
LSTM	*77.48*	*69.81*	*119.93*	*115.27*	*54.02*	*45.77*
SS-Adaptive-SM						
GRU	82.11	77.88	122.23	118.57	58.36	51.01
LSTM	82.45	78.03	122.37	118.59	57.81	49.08
SS-Hierarchical-SM						
GRU	85.29	78.83	124.24	121.60	60.48	52.19
LSTM	85.56	78.17	123.88	120.91	59.76	51.59
SS-ECOC						
GRU	86.14	78.44	125.12	121.52	58.49	50.68
LSTM	86.02	78.57	124.39	120.81	58.23	50.29
Soft-SS-ECOC						
GRU	85.78	78.12	124.69	121.13	58.18	50.33
LSTM	85.11	77.59	123.94	120.82	57.01	49.26
CLMS-Hierarchical-SM						
GRU	84.09	77.83	124.31	120.60	59.69	51.27
LSTM	84.11	77.13	123.23	121.35	59.56	50.41
DLMS-Hierarchical-SM						
GRU	82.47	78.03	124.07	122.27	59.31	53.18
LSTM	81.83	77.40	123.51	121.78	58.63	52.72
CLMS-ECOC						
GRU	*80.34*	*78.55*	122.89	118.07	58.29	50.37
LSTM	80.67	78.39	*122.27*	*117.90*	*57.81*	*50.03*
DLMS-ECOC						
GRU	*79.34*	*74.25*	120.89	117.89	*58.71*	*50.28*
LSTM	79.67	76.39	*119.27*	*117.41*	59.35	51.67

Table 2. Perplexities for Techniques that Mitigate Exposure Bias. Hierarchical Softmax uses Categorical Concrete distribution for DLMS-HS and Binary Concrete Distribution for DCMS-ECOC. CLMS-Hierarchical-SM and CLMS-ECOC both montonically increase τ according to Equation 2. Both HS and ECOC use Embedding ordered decoder matrix (we omit the -Embedding extension)

NLM models. We baseline this against both SS and the soft-argmax version of SS, the most related sample-based supervised learning approach to LMS. Furthermore, we report results on CLMS-ECOC (Curriculum-LMS ECOC) which mixes between true targets and codewords predictions according to the schedule in Equation 2 and a differentiable extension of LMS via samples from the Gumbel-Softmax (DCMS-ECOC). At training time for both DCMS-ECOC and DLMS-Hierarchical-SM we sample from each softmax defined along the path to the target code. We find that using a curriculum in CMS-ECOC to perform better in general when mixing code predictions and targets, outperforming the full softmax that uses scheduled sampling (SS-SM). Lastly, we note that DLMS-ECOC is comparable in performance to CLMS-ECOC, and improves performance on WikiText-2. Consistently, there has been an improvement using LMS over SS which suggests that LMS is an effective alternative when directly optimizing over latent variables i.e mixture sampling is less suited when using the full softmax since the target is extremely sparse (dirac delta distribution).

8. Conclusion

This work proposed an error-correcting neural language model and a novel Latent Mixture Sampling method for latent variable models. We find that performance is maintained compared to using the full conditional and related approximate methods, given a sufficient codeword size to account for correlations among classes. This corresponds to 40 bits for PTB and 100 bits for WikiText-2 and WikiText-103. Furthermore, we find that performance is improved when rank ordering the codebook via embedding similarity where the query is the embedding of the most frequent word. Lastly, we introduced Latent Mixture Sampling which can be integrated into training latent-based language models, such as the ECOC-based language model, to mitigate exposure bias. We find that this method outperforms well-known sampling-based methods for reducing exposure bias when training neural language models with maximum likelihood.

References

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 1171–1179, 2015.

Bengio, Y. and Senécal, J.-S. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Transactions on Neural Networks*, 19(4): 713–722, 2008.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003a.

Bengio, Y., Senécal, J.-S., et al. Quick training of probabilistic neural nets by importance sampling. In *AISTATS*, pp. 1–9, 2003b.

Berger, A. Error-correcting output coding for text classification. In *IJCAI-99: Workshop on machine learning for information filtering*, 1999.

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

Goodman, J. T. A bit of progress in language modeling. *Computer Speech & Language*, 15(4):403–434, 2001.

Goyal, K., Dyer, C., and Berg-Kirkpatrick, T. Differentiable scheduled sampling for credit assignment. *arXiv preprint arXiv:1704.06970*, 2017.

Grandvalet, Y. and Bengio, Y. Semi-supervised learning by entropy minimization. In *Advances in neural information processing systems*, pp. 529–536, 2005.

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

Gutmann, M. and Hyvärinen, A. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 297–304, 2010.

Hamming, R. W. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

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.

Kong, E. B. and Dietterich, T. G. Error-correcting output coding corrects bias and variance. In *Machine Learning Proceedings 1995*, pp. 313–321. Elsevier, 1995.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

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

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Mnih, A. and Teh, Y. W. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*, 2012.

Morin, F. and Bengio, Y. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pp. 246–252, 2005.

Sejnowski, T. J. and Rosenberg, C. R. Parallel networks that learn to pronounce english text. *Complex systems*, 1(1): 145–168, 1987.

Shi, K. and Yu, K. Structured word embedding for low memory neural network language model. *Proc. Interspeech 2018*, pp. 1254–1258, 2018.

Shu, R. and Nakayama, H. Compressing word embeddings via deep compositional code learning. *arXiv preprint arXiv:1711.01068*, 2017.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 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.

Zhang, L., Zhang, Y., Tang, J., Lu, K., and Tian, Q. Binary code ranking with weighted hamming distance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1586–1593, 2013.