# Word Embedding Reconstruction with Subword Embeddings for Language Modelling

*B214472*

*word count: 6781*

Master of Science

Speech and Language Processing

School of Philosophy, Psychology and Language Sciences

University of Edinburgh

2023

# Abstract

Subwords have increasingly become a standard unit for language processing because of their flexibility to rare and new words, but in languages such as English, some Automatic Speech Recognition (ASR) systems are still overseen by language models operating at a word-level in order to not hallucinate new words. To ensure rare words are able to be transcribed, modern systems hold a costly 2M+ word embeddings (2.4GB) in-memory while language modelling. We identify an opportunity to use the reconstruction of word embeddings with learned subword embeddings to hold a much smaller number of subword embeddings, and corresponding reconstruction parameters. Previous research in word embedding reconstruction has shown that the amount of space required in-memory can be decreased by a factor of 10 while maintaining over 80% of the integrity of the embedding. In this paper we use various sized models to reconstruct embeddings and evaluate their similarity to the reference embeddings in which the models are trained from. We extend the evaluation to language modelling tasks as well. We find that an existing self-attention reconstruction model, in combination with using reference embeddings for the most common words, results in an embedding reconstruction method a fraction of the size of a full word embedding table (364MB $<$ 2400MB) while creating embeddings nearly as good. We evaluate on distance between reconstructed and reference embeddings, as well as perplexity when used to represent words in a language model.

# Acknowledgements

This work was in conjunction with Speechmatics, and I'd like to give a special thank you to Chris, Caroline, and the rest of the Enhancements team for their guidance and support throughout the project. I appreciate everyone else at Speechmatics who made my month in Cambridge so wonderful and productive.

I'd be remiss to not also thank Shota Sasaki for his help with getting the self attention model up and running.

Simon, Catherine, and Korin, as leaders of the Speech and Language Processing MSc at Edinburgh, also deserve a special thank you for making my last year a rich and rewarding experience. Thanks to Simon for his guidance as the academic advisor of this project, and also to Catherine for her help throughout the last year as my personal tutor.

Most of all I'd like to thank all of lovely new friends in SLP who made the last year so special. We did it! I love you all.

# Table of Contents

# Chapter 1

# Introduction

Word embeddings are a cornerstone of almost any Neural Network (NN) based Natural Language Processing (NLP). More recently, subwords have become the standard base unit in favor of words due to their ability to generalize and represent unseen words. However, even modern Automatic Speech Recognition (ASR) systems for many languages still rely on supervision from a word-level model to ensure that new words are not hallucinated. As word vocabulary sizes continue to grow, the embedding table used to represent these words in Language Models (LM) grows as well. We spot an opportunity to use word embedding reconstruction techniques to drastically reduce required size in-memory.

## 1.1 Research Question

To this end we investigate: *To what degree can word embedding reconstruction techniques be used to shrink the size in-memory taken up by embeddings in a language model?* We explore the trade-offs between size, reconstruction time, and integrity of the reconstructed embeddings measured by distance and language modelling performance. We expect that by using trained subword reconstruction methods, we can reduce the amount of memory required by the LM by, at the least, a greater proportion than we degrade the language modelling performance.

## 1.2 Field Contributions

While various word embedding reconstruction methods exist, this paper serves as the first investigation into the suitability of word embedding reconstruction methods for

use in language modelling. Previous research focuses largely on these models being used to represent Out-of-Vocab (OOV) words. These are rare or new words that are previously unseen in training data. In this past work, size reduction is treated generally as a secondary benefit, with some proposed methods being even larger than a simple word embedding table. We then explore using a hybrid approach, and substitute in reference embeddings of common words instead of their reconstructed form. We finally investigate with further size reduction of previously proposed models.

## 1.3 This Paper

Chapter 2 provides the background surrounding language modelling in ASR, word embeddings, subwords, and existing reconstruction methods. We use this background to motivate these methods in Language Modelling. Chapter 3 details the reconstruction methods we consider in this evaluation and specifics of data and training process. In Chapter 4 we discuss the evaluation methods used to measure reconstruction integrity and language modelling ability of reconstructed word embeddings. Chapter 5 details our experiments and results, while Chapter 6 is a wider discussion on our work, including its impact and possible future work. Chapter 7 succinctly concludes.

# Chapter 2

# Background

In this chapter we define our scope, while providing relevant definitions and domain knowledge. We use this to build a motivating case for using word embedding reconstruction methods as a substitute for a word embedding table when language modelling.

## 2.1 Language Modelling in ASR

Consider an ASR pipeline that consists of separate acoustic and language model components separated by an intermediate output. The acoustic modelling component accepts audio and outputs characters, possibly graphemes or phones, representing the sequence of audio. An initial n-gram LM is then used to reconstruct a lattice of the most possible transcriptions, scored with their probabilities.

Some ASR pipelines stop here and use the scores from the n-gram LM to determine the most likely output. We instead consider a pipeline that discards scores on the lattice from the initial pass, and treats all paths as equally likely. In this way, we are using the n-gram LM to allow through only the most likely paths as a form of beam-search. We then pass this into the language model component which re-scores the lattice with a more powerful but expensive RNNLM to determine which path is the most probable.

Ideally, we would recombine the AM output into subwords which allows us to create any word, existing or new, while having a smaller vocabulary size. This works well for agglutinating languages where many small but independently meaningful pieces are stitched together, like Finnish and Turkish. For syntactic languages more dependent on word order like English, using a subword as the smallest base unit still has benefits, but is a much less obvious choice. The trade-off between making up non-

sense words compared to the benefit of correctly transcribing a few rare words must be closely considered.

For this reason, some modern ASR systems still use pipelines with word-level supervision. In this paper, we consider such a system that has an RNNLM decoding a word-level lattice produced from the AM, such as the one seen in Figure 2.1. Supervising at a word level is costly in that is requires every word you wish to transcribe to be present in a vocabulary. These 2M+ words are stored in a table in-memory and accessed as needed when words from the lattice are passed into the LM. This is the table we look to replace in this work.

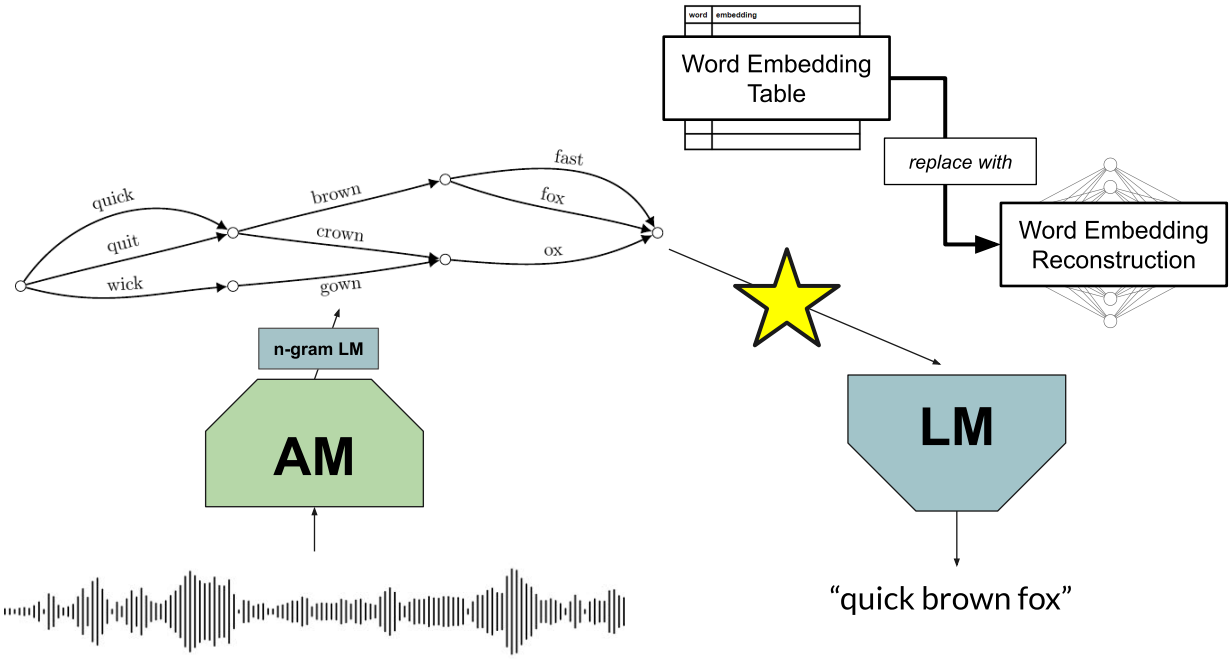


Figure 2.1: **Example AM-LM ASR pipeline**, where a representation of audio is fed into an acoustic model which, using a simple n-gram LM, produces a word level lattice of the most possible transcriptions. The lattice is re-scored by a language model which finds the most probable path through the lattice. This project, denoted by the star, concerns replacing the word embedding table with a word embedding reconstruction method that takes up less space in-memory.

## 2.2 Word Embeddings

Vocabulary words are stored in the table as word embeddings. Word embeddings, or word vectors (Bengio et al., 2003), represent all words in a vocabulary in a high dimensional space. Initially, these were a byproduct of the necessity of having a learnable layer between word and internal representation, and therefore heavily depended on task. For example, a system designed to differentiate between positive and negative sentiment would result in 'love' having a very different embedding than 'hate'. But a system designed for modelling language would likely have them much closer together, as they both are verbs relating to feelings and would fit a phrase like "Wow, I really (love/hate) this movie".

Pre-training a set of generalized word embeddings applicable to many tasks as a form of transfer learning (Collobert and Weston, 2008) became a fundamental idea in many NLP pipelines following efficient training framework word2vec (Mikolov et al., 2013) and state of the art embeddings GloVe (Pennington et al., 2014). These embeddings were trained to best represent semantic similarity between words by learning to predict which words often appear in a similar context. fastText (Bojanowski et al., 2017), the most recent major advancement in static word embeddings, achieved state of the art performance yet again by incorporating subword information into their training process.

## 2.3 Subwords in NLP

The ability to represent more words, even those that have never been seen in the data, while having a smaller vocabulary size is an obvious entice. For this reason, subwords have become commonplace in many NLP tasks in various sizes, exclusivity, and derivation method (Table 2.1). A very common subword choice is Byte-Pair Encoding, a variable length, exclusive set of subwords derived from frequency (Gage, 1994) shown to be effective in machine translation tasks (Sennrich et al., 2016). This could work, but we instead focus on those subword sets used by previous efforts in reconstruction sets. One of these is simply characters. The other, deemed 'span of n-grams' is a complete subset of all possible subwords seen in the vocab.

| | Size? | Exclusive? | Subword set of 'jingled' |
|---|---|---|---|
| **Characters** | 1 | Yes | j i n g l e d |
| **BPE** | variable but few | Yes | j ing l ed |
| **Span of n-grams** | 1 to len(word) | No | j i n ... ngl gle ... ingled jingled |

Table 2.1: **Overview of three subword sets often used in NLP tasks.** Characters are simplistic subword set of all possible characters. Subwords learned from BPE are variable in length but generally short as well. Both of these are exclusive, non-overlapping subword sets, meaning that the set does not contain subwords that are constituents of other subwords. E.g. if 'ing' is a subword, 'ng' cannot be. The final type we consider is Span of n-grams, which uses every possible subwords from length 1 up to the length of the word.

## 2.4 Word Embedding Reconstruction

In an effort to better represent OOV words and store significantly less word embeddings than the vocabulary size, various architectures have been proposed to, given a set of pre-trained word embeddings like fastText and a subword set, learn subword embeddings and reconstruction parameters to approximate the embedding of any given word. While trained only on reference embeddings, these subword models have also been found to create reasonable embeddings for OOV words it was not trained on. As overviewed in Figure 2.2, all reconstruction can be boiled into three parts: subword set, reconstruction architecture, and reference embedding set.

### 2.4.1 Character Based Methods

Initial reconstruction methods stick strictly to a character subword set, shown in the top half of Figure 2.3. Recurrent models such as MIMICK (Pinter et al., 2017) use a 512 hidden dimension bi-directional Long Short-Term Memory Network (LSTM). Recurrent architectures were an obvious first choice, but are lacking in their ability to build up different sized subwords.

Inspired by the success of character-based CNNs for text classification (Kim, 2014) and more directly work to improve word embeddings with subword information (Cao

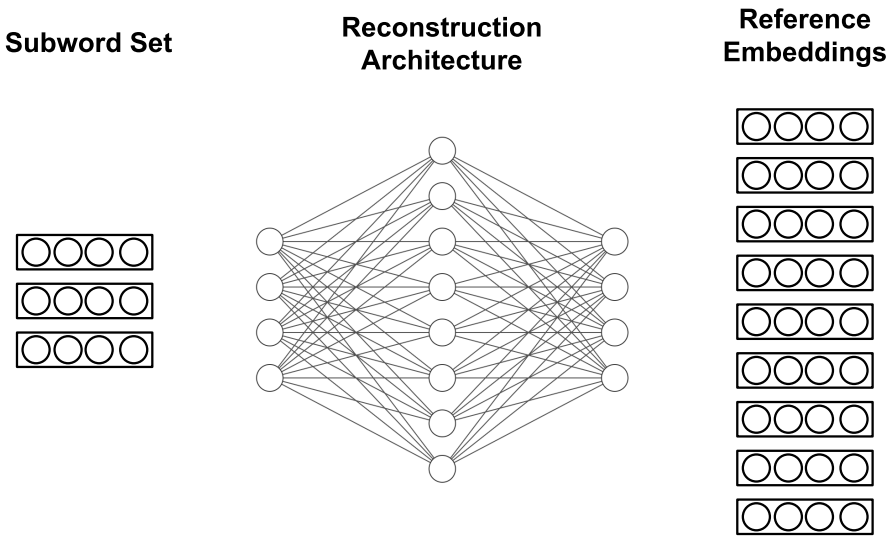


Figure 2.2: **The three components of any word embedding reconstruction method**. On the left, a designated subword set that will be trained with embeddings (shown as length 4 here). On the right, a set of pre-trained reference embeddings of the same length which will be used as a target when training the model. In the center is an architecture. A simple feed forward NN is shown here for a visual, but architectures can be nearly anything.

and Lu, 2017), a character CNN showed better reconstruction abilities while only being slightly larger (14MB $<$ 25MB) (Kim et al., 2018).

### 2.4.2 Span of n-gram Methods

Reconstruction methods based on n-grams do not need to build up different sized subwords from characters, but instead use them directly as a subword unit, shown in the bottom half of Figure 2.3. The first of these models, 'Bag of Subwords' (Zhao et al., 2018), learns an embedding for every subword present in the reference embedding vocabulary. A word's reconstructed embedding is the sum of its subword embeddings. While the best performing reconstruction technique for rare and unseen words, it is not suitable for size reduction as it keeps the entire set of subwords, which is far larger than the set of words. The model is further improved and this issue further exacerbated by its predecessor 'Probabilistic Bag of Subwords' (Zhao et al., 2020), which attaches a learnable parameter to each subword.

A self-attention model addresses the problem of size while still using n-grams as

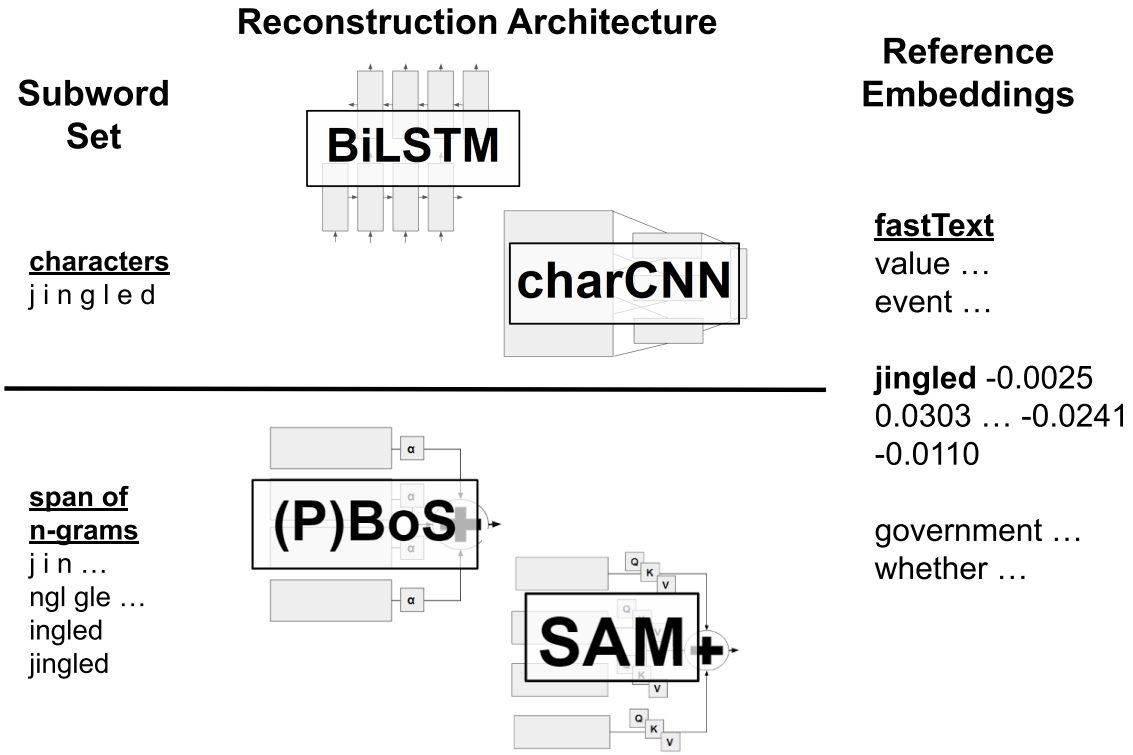


Figure 2.3: **Overview of existing word embedding reconstruction models.** Past work can be clearly split by which type of subwords are chosen. On top, recurrent and convolution architectures that work on a subword set. On bottom, those that work by summing with a learned weight or self-attention weight over a span of various sized n-grams. All architectures use a set of pre-trained set of reference embeddings to attempt to reconstruct, such as fastText.

the subword (Sasaki et al., 2021). This model keeps only the most common subwords, and then compacts them into shared embedding spaces. In combination with a self-attention mechanism operating between all subwords, this was shown to drastically reduce size while only slightly worsening reconstruction ability.

# Chapter 3

# Reconstruction Models

Of the models outlined above, we chose the character CNN proposed by Kim et al. (2018) and self-attention model by Sasaki et al. (2021), as they best suited our need of size reduction while limiting degradation of the original word embedding. Notably, ability to create reliable embeddings for OOV words is not important, since the system is lattice to re-score is at the word level.

We adhered to the subword set-type and model architecture put forth by the respective papers. In order to have a more fair comparison, we replaced the various reference embeddings they chose to use with fastText embeddings trained with subwords. We figured it sound to use embeddings trained with subword information given we will be using subword information to reconstruct them, but also believe it likely makes little difference. Further, we limited words in our reference embedding set to those that contain only characters within the regex:

```
[a-z0-9öüé\'-.+&°€$%£:]
```

That is, no capital letters and no symbols other than those provided. This brings the size of the reference embedding set from 2M to 776,200. We did this to replicate a typical ASR system, which would likely add capitals and necessary sentence-level punctuation back after the language modeling process. I wrote a simple script to do such filtering.

## 3.1 Character CNN

The first model we chose was the character-based CNN (charCNN), which has shown an obvious improvement over the recurrent architectures. While being almost double

the size of the recurrent architectures (25MB $>$ 14MB), it is still a 100x reduction compared to a full embedding set like fastText (25MB $<$ 2400MB). It even has been shown to perform as well as span of n-gram models which are almost three times larger (25MB $<$ 74MB) as reported in Table 1 of Ohashi et al. (2020).

### 3.1.1 Architecture

charCNN comprises of three main components: preprocessing, convolution filters, and a highway network. In order to perform convolutions on a word, we first need to create a matrix representation. To do this, each word is turned into its component characters, which are one-hot encoded and stacked to create a matrix as shown in Figure 3.1

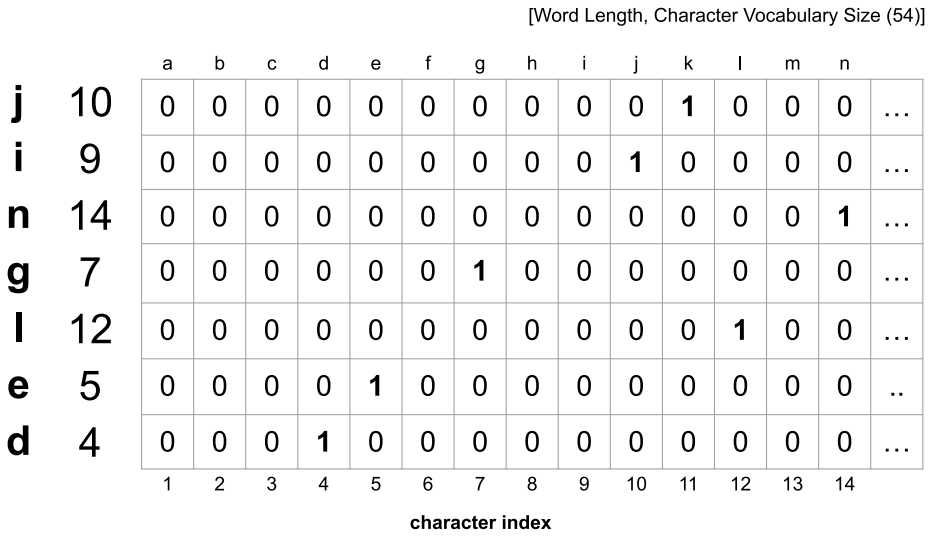


Figure 3.1: **One-hot character encoding representation of the word 'jingled'**, where a= 1. Zero is reserved for padding added later on.

To capture information from various sized subwords, filter widths between 1 and 7 are used. For each filter width, a 1-dimensional convolution of that width is applied the length of the matrix. The number of rows of the resultant matrix will be the same length as the word, but the number of columns is defined as the maximum of 200 and 50 $*$ filter width, or more specifically [200, 200, 200, 200, 250, 300, 350].

A width 3 max-pooling operation with a stride of 3 is applied to compress the information further. Finally, column-wise sums are taken to extract information from across the word and stored in a vector that represents the filter width representation for that word. This vector will be the same length as the number of columns in the post-convolution matrix. This process is shown from left to right in Figure 3.2.

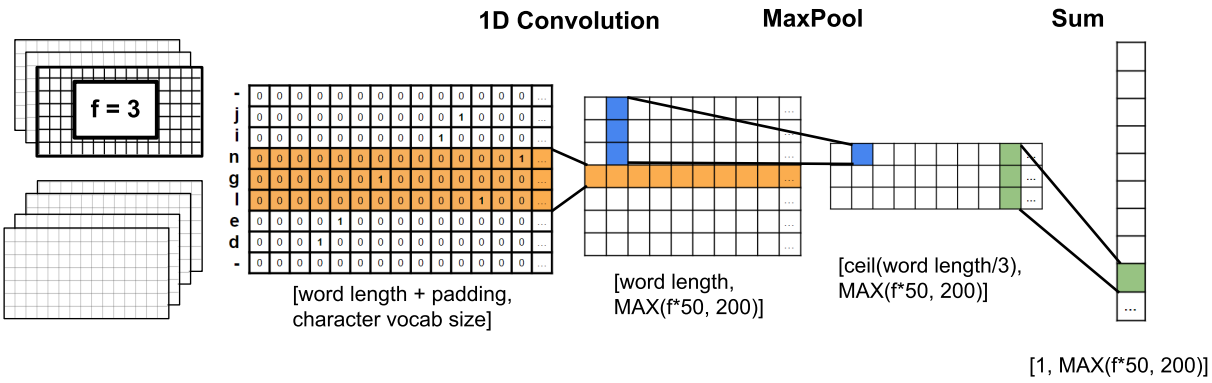


Figure 3.2: **An example of using a filter width of 3 on the matrix encoded form of 'jingled'.** At this point, filter widths of 1 and 2 have already been done. First, a single row of padding is added on each side so that when the convolution is taken it nicely maps back down to the original word length of 7. A 1D convolution of size [3 x character vocab size] is then applied. The resultant matrix is [word length x 200]. Max-pooling with a stride of 3 follows. Finally, a column-wise sum is taken and each is stored in a vector. The result is a length 200 vector that is the width 3 filter representation of the word 'jingled'.

After creating a 1D vector for each filter width 1-7, all vectors are concatenated to create a single vector. This is passed through two highway layers. While we don't focus on the speed and reconstruction accuracy gains from using a highway network in my own work, it is implemented as detailed in the Section 3.2 of Kim et al. (2018). Finally, a linear layer is used to get down to the size of the word embedding dimension. This process is detailed in Figure 3.3. During inference, this is the final reconstructed embedding. While training, this is compared against the reference embedding for the word and the squared Euclidean distance between the two is back-propagated through the network as the loss, according to the optimizer.

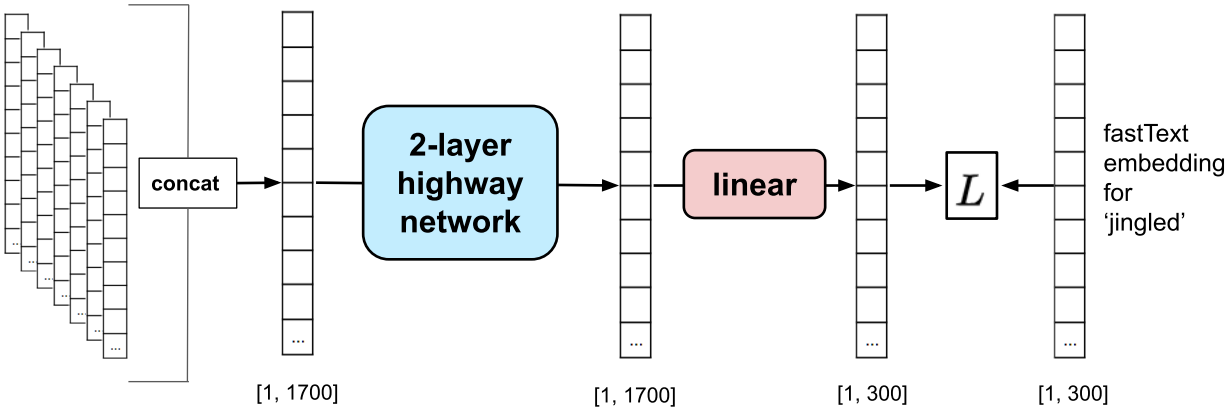


Figure 3.3: **Post-convolution steps of charCNN.** All filters widths 1-7 are concatenated to create a single vector representation of size 1700 that contains varying size subword information. This vector is then put through a 2-layer highway network. The resultant vector is cast down to the embedding dimension size of 300, and during training is compared with the reference embedding using the squared Euclidean distance, denoted as the loss $L$.

### 3.1.2 Implementation

A major part of my work for this project was the creation of my own open source implementation of the character CNN reconstruction based on the paper. [1] First I do the 'offline' preprocessing steps. In this step I accept the set of allowed characters as a string and create a mapping from character to id.

```
# Create character to id mapping
for i, c in enumerate(char_set):
    char_dict[c] = i+1
```

Then, I use this mapping to convert all words in the vocabulary into their character integer representation. All embeddings are then saved as a numpy object for easy loading during training and inference.

```
for word, vec in word_embeddings:
    char_list = []
    for c in word:
        char_list.append(char_dict[c])

    new_embedding = charlist, vec
```

I do the final preprocessing step, converting the character integer representation to a one-hot encoding, as the batches are loaded. I create a `CustomDataset` and define a new `getitem()` to use in our PyTorch data loaders as follows:

```
class CustomDataset(implements Dataset)
    def init(npy_path):
        data_list = np.load(npy_path)

    def getitem(index)
        data_item = data_list[index]
        word = torch.one_hot(data_item[0], num_classes = 55)
        vec = data_item[1]

        return word, vec
```

[1] https://github.com/Jamesetay1/subword-to-word/tree/main/charCNN

```
    def len():
        return len(self.data_list)
```

Once we have a two-dimensional representation of our word we can apply convolution filters. Note that this is done for all filter widths between 1 and 7, and so the layers are in practice defined in a list and accessed by index. We stack each filter as we define them. The `forward()` step is defined as such:

```
def forward(x):
    # Transpose matrix before doing and 1d convolution
    x = transpose(x, 1, 2)

    for n in filter_widths:
        e = nn.Conv1d(in_channels = V,
            out_channels = max(n*50, 200),
            kernel_size = n,
            padding = (n-1)/2
        )

        # Tranpose back to original shape
        e = torch.transpose(e, 1, 2)
        e = nn.tanh(e)

        # 3x1 MaxPool
        e = nn.MaxPool2d(kernel_size=(3,1))

        # Sum along columns and concat together
        e = e.sum(1)
        e = e.view(batch_size, -1)

        # Stack all widths together
        # Create it if doesn't exist yet
        y = e if y == None else torch.cat(y, e), dim = 1)

    e = y
```

After concatenating the seven vectors together, we then put that through two highway layers. This remains a part of `forward()`, but is separated here for clarity:

```
continued def forward(x):
    for i in range (0, num_layers = 2)
        #Sigmoid 'T' Gate
        T = linear(e) #1700 -> 1700
        T = Sigmoid(T)

        # Tanh 'H' Gate
        H = linear(e) #1700 -> 1700
        H = Tanh(H)

        # Carry 'C' Gate
        C = T - 1

        y = T * H + C * e
```

Finally, before returning out of `forward()` we use a final linear layer to get back down to the dimensions of the word embedding:

```
    out = linear(y) #1700 -> 300

    return out
```

### 3.1.3 Training Specifics

To train the reconstruction model, we use the same hyper-parameters detailed in Table 1 of Kim et al. (2018), including using squared Euclidean Distance as the objective function with the Adam optimizer and a learning rate of 0.001. We also use a batch size of 50. The only significant parameter not specified in the reference paper is how long to train the model for, so I decided to train until validation loss did decrease for 10 epochs, and use the previous low as the best model.

## 3.2 Self-Attention Model

The self-attention model (SAM) uses subword frequency and shared embeddings in combination with self-attention to harness the power of using all possible subwords while still reducing the size of embeddings in-memory. This makes SAM an attractive choice over (P)BoS which is not focused on size reduction, but OOV performance.

Size varies depending on number of embeddings chose, but sizes reported are as much as a 10x reduction on a typical word embedding table (230MB $<$ 2400MB). We expect it to perform better than the charCNN, at the cost of less space reduction. An implementation of the model training and inference code was available online from the authors, but I had to train a new model.[2]

### 3.2.1 Architecture

As preprocessing, SAM first determines all subwords in a reference embedding set. Then, it restricts these based on a minimum and maximum size (i.e. only subwords between 1 and 4). The most frequent $F$ subwords are kept. $F$ subwords are then hashed into $H$ embeddings, so that they share the same embedding space. Figure 3.4 shows an example.

[2] https://github.com/losyer/compact_reconstruction

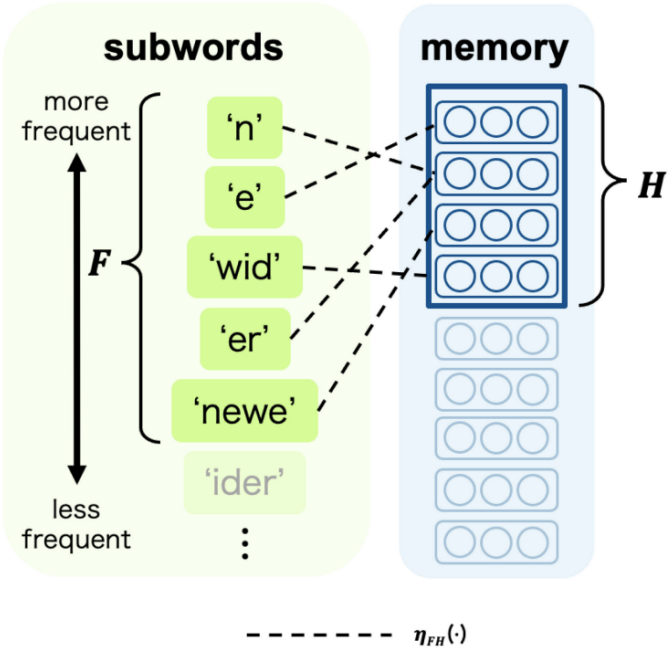


Figure 3.4: **Figure 1 from Sasaki et al. (2021)** shows a simple example of a 2 word vocabulary [newer, wider] using subword sizes between 1 and 4. In the example, Only the most frequent $F = 5$ words are kept and these are hashed in $H = 4$ shared embedding spaces. 'n' and 'er' end up sharing the same embedding.

With a set of shared embeddings defined, we then train a self-attention mechanism to find relationships between subwords of a word. The authors found that by using a self-attention mechanism, words could be reliably packed from 1M to 200k embeddings with almost no further degradation of reconstruction. When we want to reconstruct a new word, we simply push the trained subword embeddings through this self-attention layer to get an approximate embedding, which is further detailed in Figure 3.5.

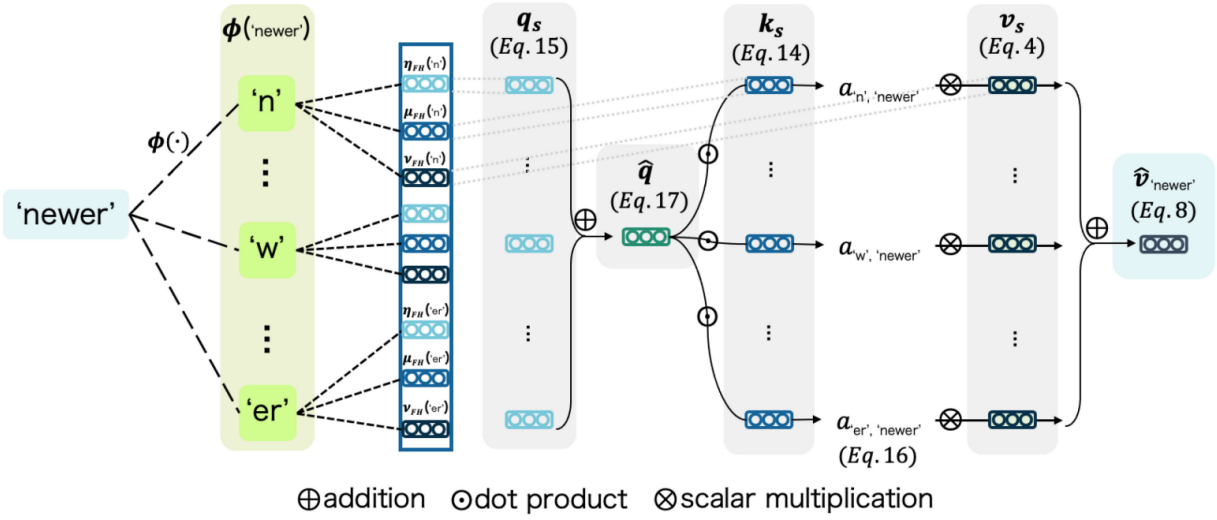


Figure 3.5: **Figure 3 from Sasaki et al. (2021)** shows how the self-attention mechanism is used to calculate a word embedding. First, all subwords that exist in the most frequent $F$ subwords are retrieved for given word 'newer'. For each subword, a weight $\alpha$ is computed using its own query vector and every other subword's key vectors. This weight is then multiplied by the vector value. Each subword's resultant values are then summed together to create the reconstructed word embedding.

### 3.2.2 Training Specifics

There are various sized spans of n-grams proposed throughout Sasaki et al. (2021), but we decided use 3-30 (essentially $\geq 3$) as it performs the best. 3 might seem an odd choice for a minimum instead of 1, but based on fastText and confirmed by initial results from models I trained, including subwords of size 1 or 2 introduces noise and

degrades reconstruction ability. As reported in Table IV of Sasaki et al. (2021), We decide to set $F$ to 1 million and use the smaller of the two $H$ values proposed at 200K for our original experiment. Later on, we scale these numbers down to see how making the model even smaller affects reconstruction ability.

The size reported by the paper for a model with a value of $H = 200K$ is 230MB, but after training our own model with the same setting we found that the size was actually 680MB. The 230MB reported appears to only account for the subword embeddings themselves, and no other necessary parameters. I reached out to the authors for clarification, but did not receive any response.

# Chapter 4

# Evaluation Methodology

We evaluate reconstruction methods in terms of factor of reduction, increase of inference time, and reconstruction accuracy. We measure reconstruction accuracy in various ways.

## 4.1 Size

As the principle reason for this research, we measure the size occupied by word embeddings in-memory, which we report in megabytes (MB). While the complete set of fastText embeddings has a size of 2400MB (2M words * 300 dimension embeddings stored as floats * 4 bytes/float = 2.4GB), our casing and symbol restriction brings this number down to 931MB. We believe that 2.4GB may still be a fair size baseline given we know that vocabularies of this size exist within our restrictions and that the words not represented in our testing are likely very rare, but can't say for sure.

## 4.2 Throughput

While we are mostly concerned with reducing the size in-memory dedicated to word embeddings and limiting the degradation of reconstructed embeddings, we cannot overlook the decrease in throughput that adding an extra inference step entails compared to simply using a look-up table. In a real-time ASR system, decreasing throughput is particularly undesirable as it. I measured the time taken to reconstruct and write all 776k words, but assumed that the model was already loaded. From this I derived words per second. I ensured that I had exclusive use of the GPU while running the reconstruction. The time taken to look up an embedding in a word:embedding dictionary

created from fastText is treated as the benchmark.

## 4.3 Reconstruction Integrity

Before measuring language modelling performance, we first measure reconstruction integrity by comparing a reconstructed embedding to its reference embedding. I created a single script that loaded in a set of reconstructed and reference word embeddings and computed each of these measures.

### 4.3.1 Distance Measuring

The simplest measure of closeness of reconstruction to reference embedding is distance. For this reason, the first measure of reconstruction integrity is the mean cosine similarity between a reconstructed embedding and its reference embedding. Many such distance measures between vectors exist, but we choose cosine similarity. Cosine similarity is a common measure of difference between word embeddings because it measures the difference between the trajectory of the vectors as the measure of closeness, and does not factor in the magnitude or length of the vectors.

### 4.3.2 Precision at k-nearest neighbours

Perhaps a more meaningful metric of reconstruction performance is to measure what reference word embeddings are around the reconstruction embedding. Precision at k-nearest neighbours (P@k) evaluates reconstruction by first putting the reference embeddings into the space, and then placing a reconstructed embedding in the space. We then ask, for each reconstructed embedding, is the reference embedding within the k closest neighbors? We evaluate with $k = 10$ and $k = 100$. Because this is an incredibly costly operation, we only perform it on the 10k most frequent words as those are the most likely to affect language modelling.

## 4.4 Language Modelling Performance

The true language modelling task we look to address is word lattice re-scoring. However, I ran out of time to get to this. Of all the ways this work could be extended this would be the most important.

### 4.4.1 Approximating Lattice Re-scoring with Perplexity

As a substitute, we train LMs with the reconstructed embeddings and find the degradation in perplexity. Past ASR research that uses RNNLMs and reports both metrics (Khassanov and Chng, 2018) (Tarján et al., 2019) has shown the relationship between Word Error Rate (WER) and perplexity of LMs in ASR systems are related, though not necessarily linearly. A large change in perplxity may only result in a small change in WER.

### 4.4.2 Dataset

For our language modelling dataset we use wikitext-2, a dataset of 2M words [1]. Past research has shown a simple RNNLM using an LSTM achieves a test perplexity of 97, but that this is improved to 46 simply by using wikitext-103 (103M words) (Merity et al., 2016). Because we are interested in the difference in perplexities and have limited time and resources, we opt for the smaller of the datasets, while still ensuring the language is being well modeled.

### 4.4.3 Training Specifications

For our language model, we use a simple LSTM to mimic a typical ASR system which uses an RNNLM. I did also experiment with using a Transformer LM, but found it took too long to get reasonable results for the amount experimenting we wish to do. We use the exponential of the validation and test loss as or perplexity value.

The perplexity values reported in the paper are with the embedding layer frozen. When experimenting we reconstructed all embeddings and treated them as if they were a set of pre-trained embeddings. In a system that actually uses a reconstruction system, embeddings would be created as they are needed. Both reconstruction models we use are deterministic, and each produces identical reconstructed embeddings for a given word. The language model would then not have an embedding layer to fine-tune as the reconstructed embedding is passed straight in as the word representation.

The language modelling code is a fork of a PyTorch tutorial repository [2], where I added the ability to use pre-trained word embeddings. After trying a few different codebases that claimed to have this functionality but did not, I chose to instead add it

[1] https://paperswithcode.com/dataset/wikitext-2

[2] https://github.com/pytorch/examples/tree/main/word_language_model

myself to very simple tutorial code. I first added arguments to take in a file of embeddings and whether they should be frozen or unfrozen, used these when initializing the language model, and finally had to edit further model initialization code to ensure that they weren't being overwritten.

# Chapter 5

# Experiments & Results

## 5.1 Experiment 1: Initial Comparison

To see to what degree charCNN and SAM can reduce the size of in-memory storage of embeddings while limiting the degradation of the reconstructed embeddings, we do an initial comparison of the the two architectures. We expect that SAM will significantly outperform charCNN in all reconstruction metrics given its much larger size. Given its much larger size, we also expect the throughput of SAM to be lower than that of charCNN, and for neither model to match the speed of simply looking up an embedding in a table as done with the reference embeddings.

### 5.1.1 Experimental Setup

We train both reconstruction models as detailed in Chapter 3. For our benchmark for this experiment, we use the pre-trained fastText embeddings. We are most interested in using this to set perplexity benchmarks, since calculating cosine similarity and P@k-nn between the reference embeddings and themselves is redundant. As a baseline, we include a random set of embeddings as well. If the reconstructed embeddings are worse or close to random embeddings, then either something as gone awry, or the reconstruction is simply not worth doing.

### 5.1.2 Results & Discussion

In Table 5.1, we first notice that embeddings from SAM, while performing quite well in reconstruction integrity metrics, does worse than random embeddings in language modelling. Further inspection shows that the SAM model predicts all single character

| | Size | Throughput | CosSim | P@10 | P@100 | valid ppl | test ppl |
|---|---|---|---|---|---|---|---|
| **fastText** | 931MB | 67.6k w/s | 1 | 1 | 1 | 127 | 137 |
| **charCNN** | 25MB | 23k w/s | .6783 | .3198 | .4911 | 143 | 155 |
| **SAM** | 680MB | 48.3k w/s | .8763 | .6867 | .7732 | 159 | 170 |
| **random** | $<$ 931MB | $<$ 67.6k w/s | .0012 | .0004 | .0099 | 154 | 167 |

Table 5.1: **Comparison of our two word embedding reconstruction methods.** Size, throughput, integrity, and language modelling results for a basic comparison between the benchmark reference embeddings fastText, charCNN, SAM, and baseline randomly initialized embeddings.

words as all zeroes, and words of length 2 are also much worse than any embeddings of words length 3+. This isn't particularly surprising given this model only considers subwords size three or greater. Given all reconstructed embeddings are extremely poor when subwords size 1 and 2 are included, we consider this a limitation of the system.

Also remarkable is that SAM has over twice the throughput of charCNN. Perhaps surprising initially, it makes more sense when you consider that only a fraction of the parameters are used for each reconstruction in SAM. Unlike the charCNN which uses the entire model for each reconstruction, SAM only needs to use the subword embedding and attention parameters for those subwords which are present in any given word. Here we also point out that the size of SAM is nearing the size of the embedding table it was trained to reconstruct.

## 5.2 Experiment 2: Hybrid Models

Inspired by the obvious flaw in SAM for words of size 1 and 2, we decide to create a model where reference embeddings for these words are saved and used directly instead of being reconstructed. We believe that this change alone will bring SAM performance in language modelling above that of charCNN as originally expected. As an extension of this experiment, we also see that this is an approximate for using the reference embeddings for the most common words in the language. We expect that as more common words reference embeddings are used directly, performance in all metrics will improve, but especially language modelling as it relies most heavily on the most common words.

### 5.2.1 Experimental Setup

For this experiment, we first derive a list of all words that have between 1-4 characters, and save their embeddings. We use the same embeddings reconstructed from experiment 1 but replace the eligibly small reconstructed embeddings with their reference fastText embedding. This new set of hybrid embeddings is named with the original model and a suffix e.g. charCNN+2 is the reconstructed embeddings from the character CNN with all embeddings for words of size 1 and 2 replaced with their reference embeddings. Table 5.2 details number of word and size of embeddings for each hybrid model variation.

| Model Suffix | n-gram Span | Additional Words | Total Words | Total Size |
|---|---|---|---|---|
| **+1** | 1-1 | 54 | 54 | 65KB |
| **+2** | 1-2 | 1587 | 1641 | 2MB |
| **+3** | 1-3 | 21903 | 23544 | 28MB |
| **+4** | 1-4 | 60477 | 84021 | 100MB |

Table 5.2: **Overview of hybrid model variations** and corresponding n-gram spans, word count, and size of embeddings.

### 5.2.2 Results & Discussion

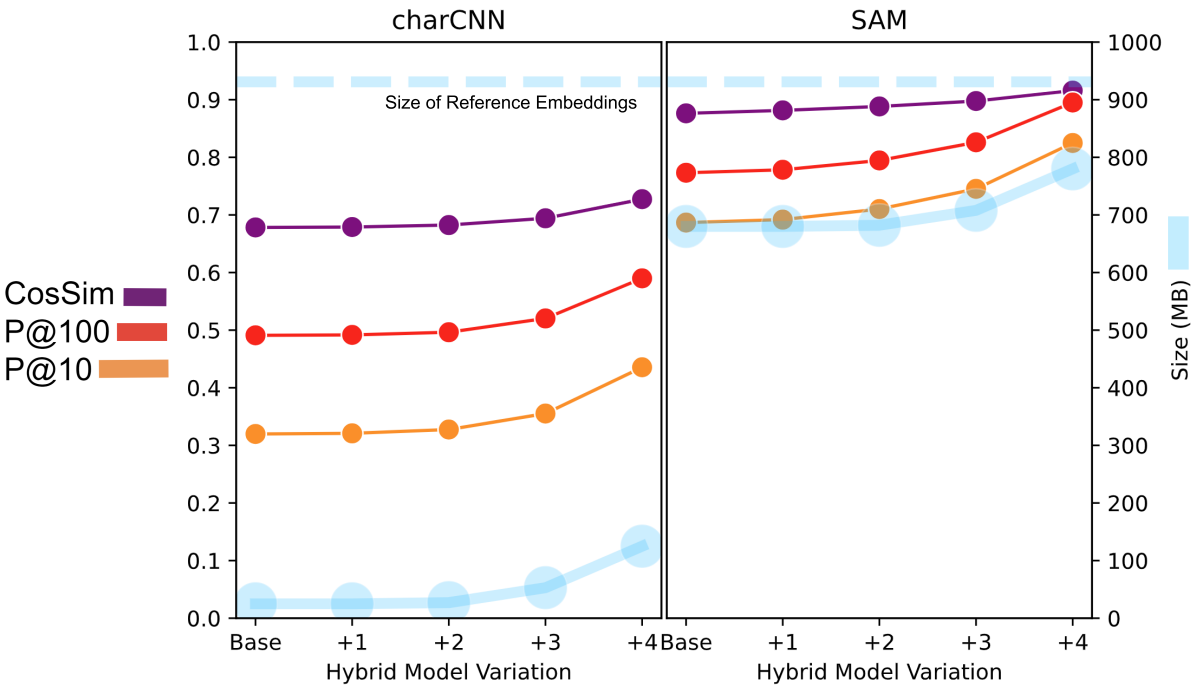


Figure 5.1: **Reconstruction integrity measures of hybrid charCNN and SAM.** Each x-axis starts from no reference embeddings being used to all reference embeddings for words up to and including length 4. On the left y-axis, we track cosine similarity, Precision@100, and Precision@10. The benchmark for these metrics is 1.0. On the right y-axis is the size of each model, with the baseline size of the reference embeddings dashed across the top.

Assessing the hybrid models as a whole (Figure 5.1), performance in reconstruction integrity metrics and size expectantly increase together. The trade-off between size and performance is one that should be made on a case-by-case basis according to priorities, but adding subwords up to size 3 (+28MB) or 4 (+100MB) are both attractive choices given additional gains in reconstruction integrity without adding a significant amount of size to the model.

In a language modelling task (Figure 5.2), we first notice a marked improvement in perplexity by adding back just size 1 embeddings back into SAM. The largest issue was that they were were all the same and, when frozen, the model could not learn to differentiate between them. Adding these words back into charCNN did very little, further confirming the problem with SAM.

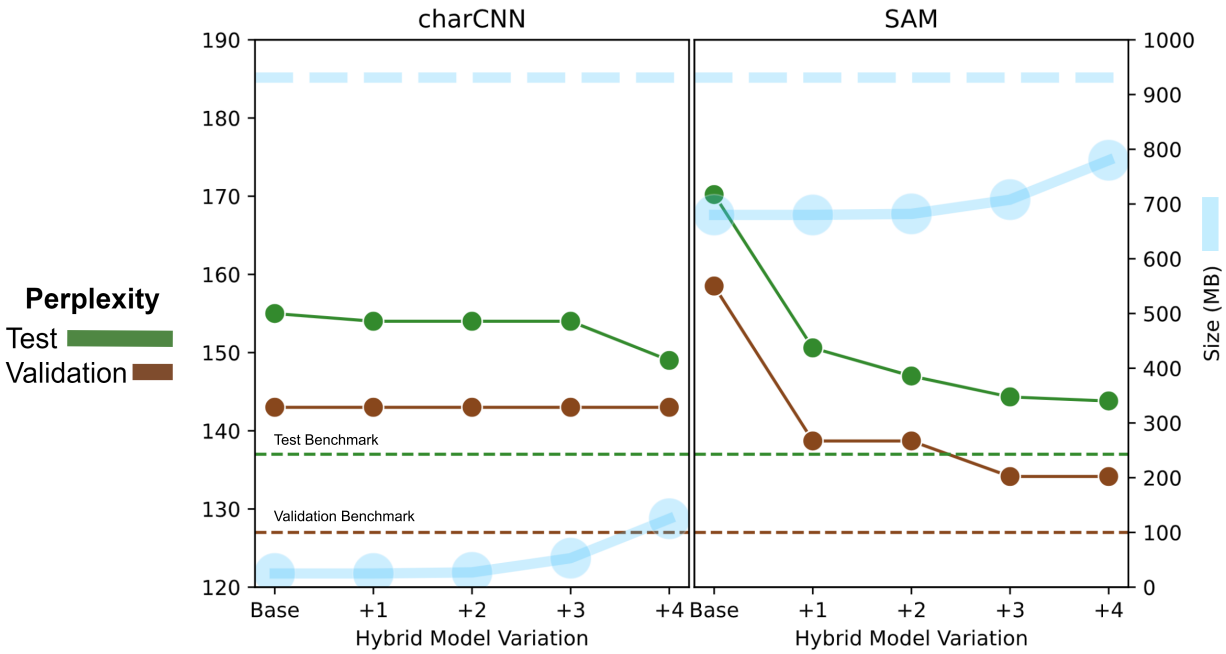


Figure 5.2: **Perplexity values of hybrid charCNN and SAM.** Each x axis starts from no reference embeddings being used to all reference embeddings for words up to and including length 4. On the left y-axis, we show test and validation perplexity values from the language modelling evaluation. Benchmark values from using fastText embeddings are dashed across the bottom. The size is again displayed on the right y-axis as before.

The perplexity values using embeddings from hybrid SAM continue to improve as reference embeddings are added, as expected. Interestingly, perplexity values using embeddings from hybrid charCNN do not improve at the same rate even through the +3 and +4 model variations, and are actually rather stagnant. Confirmed now by both reconstruction integrity and language modelling metrics, SAM is a far more capable reconstruction model at the expense of being much larger.

An obvious limitation of this study is that using word length is only an approximate, and further experiments could redo this experiment with the most frequent (100, 1k, 10k, etc.) words. A further limitation that would be exacerbated when using the most common words on a small test dataset is how much of the test dataset would be covered by reference embeddings. wikitext-2, for example, has a vocabulary size of 33K, which if used to determine which words are most common, would be almost entirely covered by +3 and covered twice over by +4. A further extension to this work would be to use a much larger test set, ideally created with a higher proportion of uncommon words.

## 5.3 Experiment 3: Shrinking SAM

Even using the smaller version of the proposed SAM architectures is still quite large, with 200K shared embeddings. Now that we've added reference embeddings back into the model we've increased that number further. By using a smaller number of shared embeddings in combination with our hybrid model approach, we believe that we once again can shrink the reconstruction model at a rate disproportionately higher to the decrease in reconstruction integrity and language modelling capacity.

### 5.3.1 Experimental Setup

| Model Suffix | Top $F$ Subwords | $H$ Shared Embeddings |
|---|---|---|
| **-200k** | 1M | 200k |
| **-100k** | 500k | 100k |
| **-10k** | 50k | 10k |
| **-1k** | 5k | 1k |
| **-100** | 500 | 100 |

Table 5.3: **Different sizes of SAM that we experimented with.** The model suffix denotes the parameter $H$, which how many shared embeddings are being used. For each $H$, the top $F$ subwords being considered is $H * 5$, in line with the paper.

To this end, we retrain SAM with the $H$ and $F$ parameters described in Table 5.3. We then create two hybrid versions (+3 and +4) of each sized model. We chose to omit +1 and +2 as they are so small including them for the benefit they bring seems almost like a no-brainer. As we add longer words, the number of words included grows rapidly and so does the total size. This becomes a more interesting trade-off to explore.

### 5.3.2 Results & Discussion

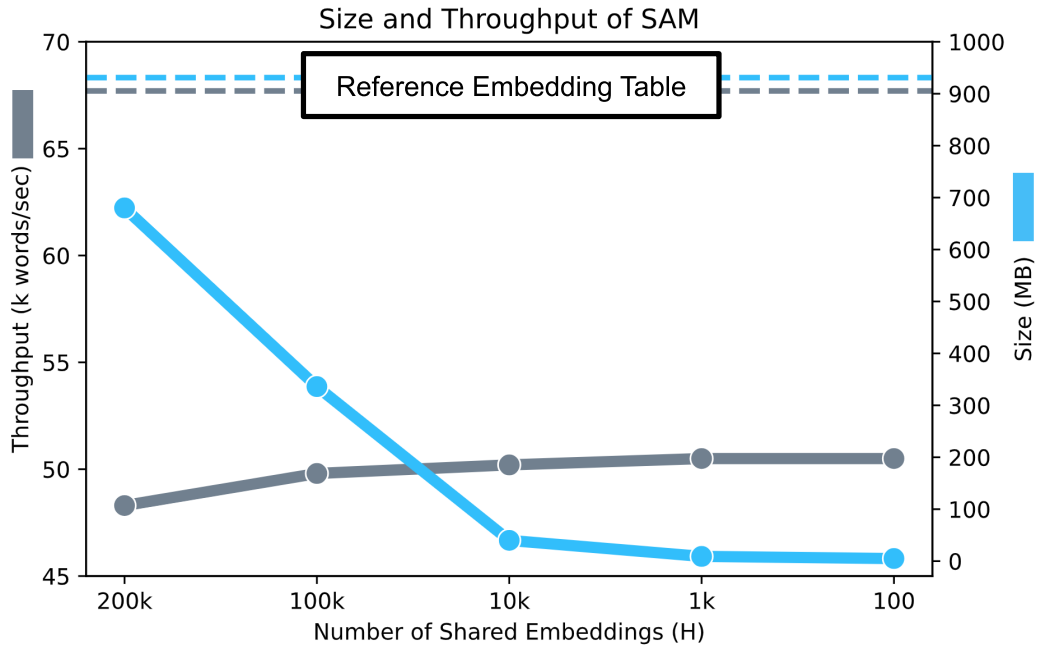


Figure 5.3: Size and throughput of Base SAM as we shrink the number of shared embeddings. The left y-axis reports throughput of the model in thousands of words per second. The right y-axis reports size in megabytes. The benchmark for throughput and baseline for size from using a simple table of reference embeddings are shown dashed across the top.

Figure 5.3 shows that as we decrease the number of shared embeddings used the size of the model shrinks drastically, from the original 680MB all the way to just 5MB. We also see that the throughput improves but not to near the same rate, only by a few thousand words per second overall. We don't report throughput on the hybrid models, but it should be slightly higher as the time to look-up is faster than reconstruction at any size.

Assessing the reconstruction and language modelling performance of SAM +3 and +4 (Figure 5.4), both see a slight improvement going from 200k embeddings to 100k embeddings. I did retrain the reconstruction model and language model because it seemed as if something had gone wrong, but got similar results. This could point to not training on enough data for 200k embeddings to be truly separated and so that model is under-trained, but the loss curves during training don't suggest this. To investigate the issue further it would be interesting to extend this graph to the left and see if the perplexity goes down as expected, or if there is some jitter. What this data does tell us

that is that a hybrid SAM using only 100k embeddings has good reconstruction abilities and achieves a test perplexity close to that of fastText ($137 < 142$) while being much smaller ($364\text{MB} < 931\text{MB}$).

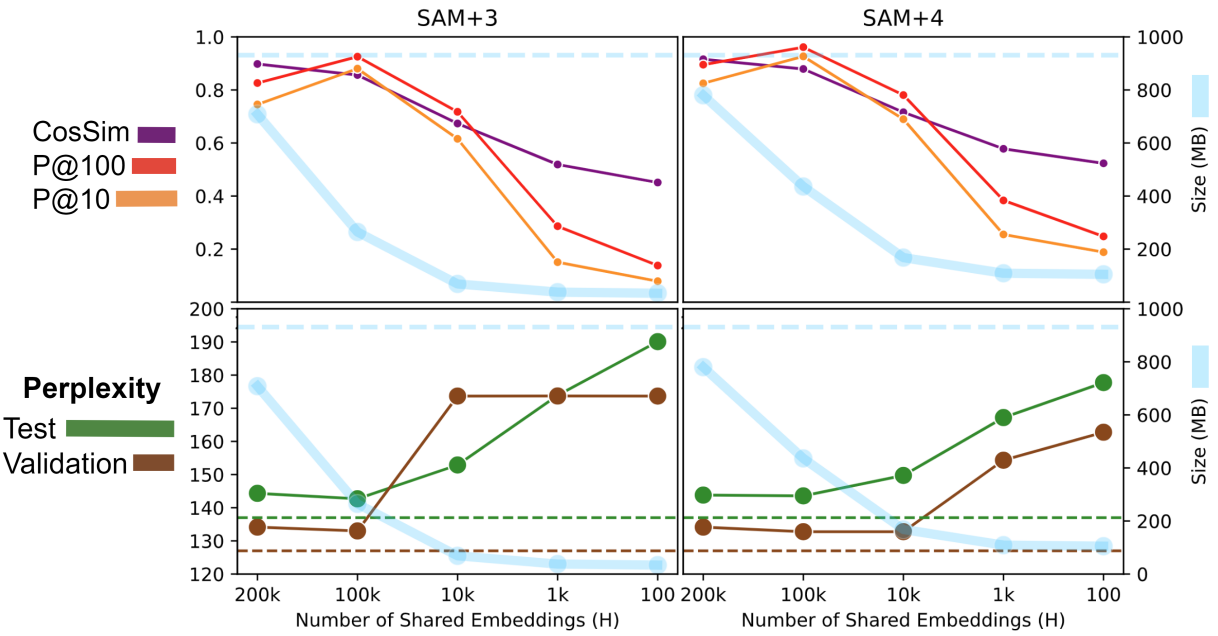


Figure 5.4: **Performance of +3 and +4 hybrid SAM as the number of shared embeddings shrinks.** The upper graphs concern the reconstruction integrity metrics. The benchmark for these is 1. The lower graphs show test and validation perplexity, with the benchmark perplexities using fastText dashed across the bottom. Size is shown on the right y-axis and in the background, with the baseline across the top.

We also notice an oddity at SAM+3-10k where perplexity suddenly jumps and then levels out. We theorize that because all words sized 1-3 have their reference fastText embeddings, this is the ceiling of perplexity that won't change regardless of how bad the other embeddings become. This notably doesn't happen in SAM+4, indication adding reference embeddings of size 4 are still quite useful when language modelling.

# Chapter 6

# Discussion

This chapter serves as an overview of the results and how we can make use of such findings, followed by immediate extensions to this paper, and longer term work in this field.

## 6.1 General Discussion

Through experimentation with two different word embedding reconstruction architectures proposed in previous research, we found that while the self-attention model performed well in metrics which compares distances between reconstructed and reference embedding, it performed worse than random embeddings in language modelling performance measured by perplexity. We discovered that, as a result of using subwords with a minimum length of 3, embeddings of length one were being reconstructed as all zeroes, and those of length two were also particularly poor. We find that creating a hybrid model using the reference embeddings for common words makes little difference for charCNN, but adding just 28MB of embeddings decrease the perplexity for SAM from 159 to 134. To decrease the size of SAM we find that halving the number of shared embeddings being used doesn't deteriorate the embeddings performance, perhaps suggesting that the larger model is under-trained.

We conclude that a model like SAM-100k as a +3 or +4 hybrid model, ideally amended to use the n-most popular words instead of a crude approximation like character length, creates word embeddings that when used in a language model in place of a table of pre-trained embeddings, results in only slightly worse performance while being almost a quarter of the size of the 776k reference embeddings it was trained from.

## 6.2 Impact of Work

We believe that a reconstruction model such as SAM-100k+3 (364MB) could reliably replace a 2M+ table of word embeddings (2400MB) when representing words from a lattice in the LM step of an ASR pipeline. Done for the pipeline of a single language such as English, we could see up to a 2GB savings. While not a significant amount in today's cloud computing world, this could be a significant savings for systems required to run on a single computer.

More importantly, if a similar method could be applied to each language still being supervised at a word-level in ASR, the savings becomes much more prominent. In a setting where ASR models for many different languages are all being hosted together, it could mean that less compute nodes are necessary overall, saving money and electricity.

If vocabularies grow larger as times goes on, the benefits of saving space will only grow. Originally designed with the construction of OOV word embeddings as a primary goal, SAM should not need to be retrained every time a word is added or perhaps even at all.

## 6.3 Future Work

### 6.3.1 Immediate Extensions to Paper

There are a few instances of immediate future work that I would have done given more time. Firstly, I would create a secondary hybrid model experiment that uses the n-most common vocabulary words to use as the reference embeddings instead of using word length as an approximate. Alongside this, I would use a much larger test set and ensure that rare words are included.

Next, I would extend evaluation of reconstructed embeddings to lattice re-scoring. This system was created to be used in an AM-LM ASR pipeline where a lattice of possible word sequences is passed into, and scored by, a language model. While perplexity of the language model is a decent approximation, its necessary to confirm that the results carry over.

### 6.3.2 Longer Term Work

This work is designed for, and evaluated in, a system with a closed vocabulary. That is, it reconstructs embeddings from a word-level lattice from a defined set. However, such systems could be used in a open vocabulary system, and be used to reliably create embeddings from words that have never been seen before. Such a system is an ideal goal for ASR as it allows new words to be transcribed without having to explicitly add them to a vocabulary.

# Chapter 7

# Conclusions

In a pursuit to reduce the amount of space resident in-memory by storing a word embedding table necessary for lattice re-scoring in ASR, we experimented with two different word embedding reconstruction models. We found that by using a hybrid self-attention based model with the reference embeddings used for common words, we can reduce space resident in-memory by a factor of at least 4 while seeing less than a 5% increase in perplexity. More evaluation work is needed to confirm that this slight degradation in perplexity doesn't translate to a larger degradation when re-scoring lattices.

Beyond the immediate scope of this paper, the challenge of meeting word-level accuracy while not inventing new words remains a limitation of using subwords as a base unit for all language remains. A subword vocabulary, with limited exceptions, will take up less space than a whole word vocabulary while being able to represent any possible word and is a worthwhile goal requiring further clever solutions.

# Bibliography

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, 2003.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. doi: 10.1162/tacl_a_00051. URL https://aclanthology.org/Q17-1010.

Shaosheng Cao and Wei Lu. Improving word embeddings with convolutional feature learning and subword information. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017. doi: 10.1609/aaai.v31i1.10993. URL https://ojs.aaai.org/index.php/AAAI/article/view/10993.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *International Conference on Machine Learning*, 2008.

Philip Gage. A new algorithm for data compression. *The C Users Journal archive*, 12: 23–38, 1994.

Yerbolat Khassanov and Eng Chng. Unsupervised and efficient vocabulary expansion for recurrent neural network language models in asr. pages 3343–3347, 09 2018. doi: 10.21437/Interspeech.2018-1021.

Yeachan Kim, Kang-Min Kim, Ji-Min Lee, and SangKeun Lee. Learning to generate word representations using subword information. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2551–2561, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics. URL https://aclanthology.org/C18-1216.

Yoon Kim. Convolutional neural networks for sentence classification, 2014.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. 2013.

Sora Ohashi, Mao Isogawa, Tomoyuki Kajiwara, and Yuki Arase. Tiny word embeddings using globally informed reconstruction. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 1199–1203, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.103. URL https://aclanthology.org/2020.coling-main.103.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL https://aclanthology.org/D14-1162.

Yuval Pinter, Robert Guthrie, and Jacob Eisenstein. Mimicking word embeddings using subword RNNs. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 102–112, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1010. URL https://aclanthology.org/D17-1010.

Shota Sasaki, Jun Suzuki, and Kentaro Inui. Subword-based compact reconstruction for open-vocabulary neural word embeddings. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 29:3551–3564, nov 2021. ISSN 2329-9290. doi: 10.1109/TASLP.2021.3125133. URL https://doi.org/10.1109/TASLP.2021.3125133.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162.

Balá zs Tarján, György Szaszák, Tibor Fegyó, and Péter Mihajlik. Investigation on n-gram approximated RNNLMs for recognition of morphologically rich speech. In *Statistical Language and Speech Processing*, pages 223–234. Springer International

Publishing, 2019. doi: 10.1007/978-3-030-31372-2_19. URL https://doi.org/10.1007%2F978-3-030-31372-2_19.

Jinman Zhao, Sidharth Mudgal, and Yingyu Liang. Generalizing word embeddings using bag of subwords. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 601–606, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1059. URL https://aclanthology.org/D18-1059.

Jinman Zhao, Shawn Zhong, Xiaomin Zhang, and Yingyu Liang. PBoS: Probabilistic bag-of-subwords for generalizing word embedding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 596–611, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.53. URL https://aclanthology.org/2020.findings-emnlp.53.

# Appendix A

# Experiment 1 Full Results

| | Size | Throughput |
|---|---|---|
| **fastText** | 931MB | 67.6k w/s |
| **charCNN** | 25MB | 23k w/s |
| **SAM** | 680MB | 48.3k w/s |
| **random** | $<$ 931MB | $<$ 67.7k w/s |

| | CosSim | P@10 | P@100 | valid ppl | test ppl |
|---|---|---|---|---|---|
| **fastText** | 1 | 1 | 1 | 127 | 137 |
| **charCNN** | .6783 | .3198 | .4911 | 143 | 155 |
| **SAM** | .8763 | .6867 | .7732 | 158 | 170 |
| **random** | .0012 | .0004 | .0099 | 154 | 167 |

# Appendix B

# Experiment 2 Full Results

## B.1 Self-Attention Model (SAM)

| | Size | Throughput |
|---|---|---|
| **SAM** | 680MB | 48.3k w/s |
| **SAM+1** | 680MB (+64.8KB) | $<$ 48.3k w/s |
| **SAM+2** | 682MB (+1.97MB) | $<$ 48.3k w/s |
| **SAM+3** | 708MB (+28.25MB) | $<$ 48.3k w/s |
| **SAM+4** | 780MB (+100MB) | $<$ 48.3k w/s |

| | CosSim | P@10 | P@100 | valid ppl | test ppl |
|---|---|---|---|---|---|
| **SAM** | .8763 | .6867 | .7732 | 159 | 170 |
| **SAM+1** | .8814 | .6918 | .7783 | 139 | 151 |
| **SAM+2** | .8883 | .7102 | .7945 | 139 | 147 |
| **SAM+3** | .8977 | .7452 | .826 | 134 | 144 |
| **SAM+4** | .9158 | .8249 | .8951 | 134 | 144 |

## B.2 Character CNN Model (charCNN)

| | Size | Throughput |
|---|---|---|
| **charCNN** | 25MB | 23k w/s |
| **charCNN+1** | 25MB (+64.8KB) | $<$ 23k w/s |
| **charCNN+2** | 27MB (+1.97MB) | $<$ 23k w/s |
| **charCNN+3** | 53MB (+28.25MB) | $<$ 23k w/s |
| **charCNN+4** | 125MB (+100MB) | $<$ 23k w/s |

| | CosSim | P@10 | P@100 | valid ppl | test ppl |
|---|---|---|---|---|---|
| **charCNN** | .6783 | .3198 | .4911 | 143 | 155 |
| **charCNN+1** | .6790 | .3208 | .4917 | 143 | 154 |
| **charCNN+2** | .6824 | .3275 | .4963 | 143 | 154 |
| **charCNN+3** | .6941 | .3552 | .5203 | 143 | 154 |
| **charCNN+4** | .7273 | .4355 | .5899 | 143 | 149 |

# Appendix C

# Experiment 3 Full Results

## C.1 Shrinking SAM

| | Size | Throughput |
|---|---|---|
| **SAM(-200K)** | 680MB | 48.3k w/s |
| **SAM-100K** | 336MB | 49.8k w/s |
| **SAM-10K** | 40MB | 50.2k w/s |
| **SAM-1k** | 9MB | 50.5k w/s |
| **SAM-100** | 5MB | 50.5k w/s |

| | CosSim | P@10 | P@100 | valid ppl | test ppl |
|---|---|---|---|---|---|
| **SAM(-200K)** | 0.8763 | 0.6867 | 0.7732 | 159 | 170 |
| **SAM-100K** | 0.8329 | 0.8332 | 0.8902 | 157 | 168 |
| **SAM-10K** | 0.6411 | 0.5611 | 0.6716 | 258 | 177 |
| **SAM-1K** | 0.4749 | 0.0828 | 0.2213 | 199 | 212 |
| **SAM-100** | 0.3939 | 0.01 | 0.0693 | 258 | 279 |

## C.2 Shrinking SAM+3

| | Size (+28MB) | Throughput |
|---|---|---|
| **SAM+3(-200K)** | 708MB | $<$ 48.3k w/s |
| **SAM+3-100K** | 364MB | $<$ 49.8k w/s |
| **SAM+3-10K** | 68MB | $<$ 50.2k w/s |
| **SAM+3-1K** | 37MB | $<$ 50.5k w/s |
| **SAM+3-100** | 33MB | $<$ 50.5k w/s |

| | CosSim | P@10 | P@100 | valid ppl | test ppl |
|---|---|---|---|---|---|
| **SAM+3(-200K)** | 0.8977 | 0.7452 | 0.826 | 134 | 144 |
| **SAM+3-100K** | 0.8562 | 0.8804 | 0.9254 | 133 | 143 |
| **SAM+3-10K** | 0.6734 | 0.6156 | 0.7174 | 174 | 153 |
| **SAM+3-1K** | 0.5188 | 0.1507 | 0.2861 | 174 | 174 |
| **SAM+3-100** | 0.4510 | 0.0789 | 0.1382 | 174 | 190 |

## C.3 Shrinking SAM+4

| | Size (+100MB) | Throughput |
|---|---|---|
| **SAM+4(-200K)** | 780MB | $<$ 48.3k w/s |
| **SAM+4-100K** | 436MB | $<$ 49.8k w/s |
| **SAM+4-10K** | 168MB | $<$ 50.2k w/s |
| **SAM+4-1K** | 109MB | $<$ 50.5k w/s |
| **SAM+4-100** | 105MB | $<$ 50.5k w/s |

| | CosSim | P@10 | P@100 | valid ppl | test ppl |
|---|---|---|---|---|---|
| **SAM+4(-200K)** | 0.91582 | 0.8249 | 0.8951 | 134 | 144 |
| **SAM+4-100K** | 0.8790 | 0.9265 | 0.9611 | 133 | 144 |
| **SAM+4-10K** | 0.7158 | 0.6895 | 0.7808 | 133 | 150 |
| **SAM+4-1K** | 0.5780 | 0.2553 | 0.3836 | 154 | 167 |
| **SAM+4-100** | 0.5234 | 0.1885 | 0.2477 | 133 | 178 |