Algorithmic Diversity and Tiny Models: Comparing Binary Neural Networks and the Fruit Fly Algorithm on Document Representation Tasks

Tanise Ceron * [△] **Nhut Truong** * [□] **Aurelie Herbelot** * [□O]

[△] Institute for Natural Language Processing, University of Stuttgart, Germany

[□] Center for Mind/Brain Sciences, University of Trento, Italy

[O] Department of Information Engineering and Computer Science, University of Trento, Italy

tanise.ceron@ims.uni-stuttgart.de,

{leminhnhut.truong,aurelie.herbelot}@unitn.it

Abstract

Neural language models have seen a dramatic increase in size in the last years. While many still advocate that 'bigger is better', work in model distillation has shown that the number of parameters used by very large networks is actually more than what is required for state-of-the-art performance. This prompts an obvious question: can we build smaller models from scratch, rather than going through the inefficient process of training at scale and subsequently reducing model size? In this paper, we investigate the behaviour of a biologically inspired algorithm, based on the fruit fly's olfactory system. This algorithm has shown good performance in the past on the task of learning word embeddings. We now put it to the test on the task of semantic hashing. Specifically, we compare the fruit fly to a standard binary network on the task of generating locality-sensitive hashes for text documents, measuring both task performance and energy consumption. Our results indicate that the two algorithms have complementary strengths while showing similar electricity usage.

1 Introduction

In 2022, the vast majority of state-of-the-art NLP systems are implemented as deep neural models, that is, neural networks with complex architectures which can contain hundreds of billions of parameters. Such models have become so expensive to train that most institutions cannot afford anymore to generate them from scratch. They have also been shown to generate non-negligible amounts of CO_2 emissions (Strubell et al., 2019).

An active research area focuses on model distillation (e.g. Sanh et al., 2019), that is, the process of pruning pretrained large models to only retain the weights truly essential to the system's performance. The result of distillation is a much smaller architecture, faster to run, and less memory-hungry. However, training a large model to then reduce its size seems to be a waste of resources. Ideally, we would make the right design choices to directly implement a model with a reasonable number of parameters. With this goal in mind, the present paper looks at a biologically-inspired architecture which have shown potential as a 'small model' for language processing: The Fruit Fly Algorithm (FFA).

The FFA is inspired by the olfactory system of the fruit fly, *Drosophila melanogaster*. It algorithmically describes how the fly encodes smells in its environment into a binary pattern of activations, using just two layers of neurons. The usefulness of the biological algorithm for computer science was first noted by Dasgupta et al. (2017), who modeled the mechanism as a kind of local-sensitivity hashing relying on random projections, and used it to hash pre-trained document and image vectors. Preissner and Herbelot (2019, 2020) ported the original algorithm to a Natural Language Processing setting and made the fly learn word embeddings. Liang et al. (2021) also applied the FFA to the task of creating word embeddings, serving downstream tasks such as word-sense disambiguation and document classification.

The FFA produces word embeddings of a quality comparable to that of traditional, classic methods such as GLOVE (Pennington et al., 2014) and Word2Vec (Mikolov et al., 2013). While it lags behind the performance of large language models like BERT (Devlin et al., 2019), it consumes only a fraction of inference time as well as computing power (Liang et al., 2021). Moreover, the FFA is an explainable model, thanks to a shallow architecture and sparse, binary feature representations. Taken together, these features promise to alleviate the drawbacks of mainstream giant language models, such as the need for expensive computing resources, environmental concerns and lack of interpretability.

In our work, we aim to take the FFA one step

*Equal contribution

further and use it to perform **semantic hashing**, that is, the task of learning locality-sensitive, binary vectors for various types of text input. In the general area of computational efficiency, semantic hashing is an extremely useful task: it generates meaningful vector representations at a low number of bits (typically between 8 and 128). The produced hashes can be stored very efficiently and similarity computations can be performed extremely fast over them, using hamming distance.

In the present paper, we provide a comparison of the FFA with another efficient hashing algorithm, namely the Binary Neural Network (BNN) (Hubara et al., 2016). Both techniques are very different in terms of architecture and training regime. In the spirit of increasing 'algorithmic diversity' (Preissner and Herbelot, 2020), we think it worthwhile to investigate the respective benefits of the two methods, and we pitch them against each other on the tasks of document classification and information retrieval. Our results show that the two techniques are complementary in strength while both satisfying requirements of energy efficiency.

2 Related work

Semantic hashing The task of semantic hashing comes from the area of information retrieval, where it is crucial to be able to cluster documents according to their semantic similarity, in order to retrieve documents given a query. One way of going about this problem is to assign a code, hash or vector representation to each document. The more similar the hash of two given documents, the more semantically related they are. These representations should be storable in a fixed number of bits so that the retrieval of documents – through hamming distance – is fast and computationally efficient.

Models for semantic hashing vary, from very simple methods involving counts or tf-idf values (Salton and Michael, 1986), to more complex ones involving deep learning. The current unsupervised state-of-the-art systems rely on heavy machinery such as generative models with variational autoencoders (Chaidaroon and Fang, 2017; Zhang and Zhu, 2019; Hansen et al., 2020), which are capable of reducing the high-dimensional data into a low latent space.

Document classification The task of document classification is a traditional one in NLP, and like many other tasks, it has become associated with more and more complex architectures. A few years ago, classification used to be tackled using different architectures of neural network such as CNNs (Liu et al., 2017) and biLSTMs (Adhikari et al., 2019b) with static word embeddings. Nowadays, the preferred method is to input the entire document into a large language model (LLM), retrieve its representation, and feed it into a fully connected layer or a linear classifier. LLMs are based on Transformers and have a massive amount of parameters, (e.g. 170 billion in GPT-3: Brown et al., 2020). They have fostered substantial progress in search engines in the last few years[1] and indeed create excellent text representations. But they also have the many pitfalls mentioned in our introduction, from low interpretability to high computational cost, as well as erroneous filtering of content in the pretraining process, disfavouring minority groups (Dodge et al., 2021; Bender et al., 2021).

From a purely engineering point of view, the drawbacks of LLMs have prompted the publication of various papers trying to tackle core issues in the models. For instance, Adhikari et al. (2019a) implement knowledge distillation as compression technique in the BERT-large model to deal with the problem of run-time memory caused by these large systems. Another known issue is that standard LLMs such as BERT (Devlin et al., 2019) can only take up to 512 tokens input due to their self-attention mechanisms. Therefore, dealing with long documents can still be a challenge. Researchers have dealt with it by creating Transformer-based models that can take even longer texts as input (Beltagy et al., 2020).

Our own stance is that, beside improving LLMs, our community should experiment with more diverse computational architectures to solve the outstanding problems. As Bender et al. (2021) point out, we should be careful not to focus only on state-of-the-art architectures and encourage instead research efforts and funding into diversifying natural language processing models.

3 Datasets

Six datasets were used to evaluate our algorithms: 20 Newsgroups[2] (20news) (Lang, 2008), Agnews[3]

[1] https://blog.google/products/search/search-language-understanding-bert/

[2] qwone.com/~jason/20Newsgroups/20news-bydate.tar.gz

[3] groups.di.unipi.it/~gulli/AG_corpus_of_news_articles.html

(Zhang et al., 2015), Reuters[4] (Lewis, 1997), TMC[5] (Oza, 2010), Wikipedia (Wiki), and Web of Science[6] (WoS) (Kowsari et al., 2017). Statistics and short descriptions of all datasets can be found in Table 1.

All datasets are available to download, except Wiki, which we collected by scraping English Wikipedia pages. For the 20news dataset, we used the specific version sorted by date. For Reuters, we used the version ModApte R(90). For WoS, we used the medium-size version. Three datasets (20news, Agnews, and Reuters) had already been provided separate train and test sets. Thus we kept these test sets intact for the purpose of comparison, and further split the train sets to extract validation sets.

3.1 Pre-processing

All datasets are lowercased and tokenized with a SentencePiece[7] model, generated from the train split. Using this tokenization method makes our system ready to be used with languages other than English. SentencePiece transforms the data into tokens belonging to a vocabulary of d wordpieces, with d set here at $10,000$. It also returns the log-probabilities of each token in the training data. We will write l_i to refer to the log-probability of token i in the SentencePiece model.

After this initial pre-processing, each dataset is vectorized. For a dataset of n documents, we obtain a matrix of size $n \times d$, where each row represents a document and each column a token in our vocabulary of word pieces. Each cell in the matrix shows the normalized frequency of a token in the document, reweighted by l_i^p, where p is an exponent used to increase or decrease the effect of l_i. The reason for weighing frequencies in this way, rather than using a conventional measure such as tf-idf, is that it allows the system to be incremental at test stage. That is, any new document seen by the model can be vectorized without needing access to the entire document collection.

Finally, we experiment with keeping only the top t words in each (reweighted) document vector, which lets us optimize the number of infrequent and/or uncharacteristic words seen by the system. That is, before feeding the input to the system, we zero out the $d - t$ cells with lowest weights in each row of the matrix.

[4]Downloaded from nltk library.

[5]catalog.data.gov/dataset/siam-2007-text-mining-competition-dataset

[6]data.mendeley.com/datasets/9rw3vkcfy4/6 under the license CC BY 4.0

[7]https://github.com/google/sentencepiece

4 Models

4.1 The Binary Neural Network (BNN)

BNN (Hubara et al., 2016) is an example of a light and efficient model, which bridges the gap between production in industry and research. Thus, the model provides an ideal comparison for our FFA. BNNs have a reduced cost of computation with respect to continuous neural networks because weights and activations are binarized at run-time, as well as during gradient computation, with +1 and -1 values. Weights and activations undergo a deterministic binarization step:

$$x^b = Sign(x) = \begin{cases} +1 & \text{if } x \geq 0, \\ -1 & \text{otherwise} \end{cases}$$

where x is the continuous variable and x^b is the binarized value. The Stochastic Gradient Descent, however, is computed with the accumulated continuous-valued weights in order to have high precision.

Our BNN has one input layer, one hidden layer and one output layer. It is trained to predict which class(es) a certain document belongs to. The input is a matrix $R^{n \times d}$ with the pre-processed documents (cf. 3.1) where n is the number of documents in the batch and d is 10,000. The output layer is a binary vector representation of dimensionality $R^{h \times l}$ where h is the number of neurons in the hidden layer and l is the number of labels in the dataset. Cross Entropy loss is computed for single label documents whereas in the multilabel classification the loss combines a sigmoid layer and Binary Cross Entropy.

We experiment with three sizes of hidden layers (32, 64 and 128) and two different learning rates (0.01 and 0.001). The training batch size is kept at 32 across datasets. Training is run for 50 epochs with early stopping after 5 epochs. The number of epochs of each final model varies between 6 and 42.

While the BNN is trained on a classification task, it can also be used for semantic hashing. To get an unseen document's hash, we feed it into the neural network and extract the hidden layer before the classification layer as the representation of the

Dataset	Multi-class	Classes	Num docs	Train-val-test split (%)	Topics
20news	No	20	18846	42-18-40	Newsgroups, such as motorcycles, computer, politics, etc
Agnews	No	4	127600	75-19-6	News articles about the world, sports, business, and science/tech
Reuters	Yes	90	10788	56-16-28	News articles on various topics, such as jobs, gas, housing, wheat, etc
TMC	Yes	22	28596	60-15-25	Air traffic reports
Wiki	No	15	29924	60-20-20	Wikipedia pages in categories, such as music, football, law, etc
WoS	No	35	11967	60-20-20	Scientific papers' abstracts in different fields, such as biochemistry, psychology, etc

Table 1: Statistics and description of datasets

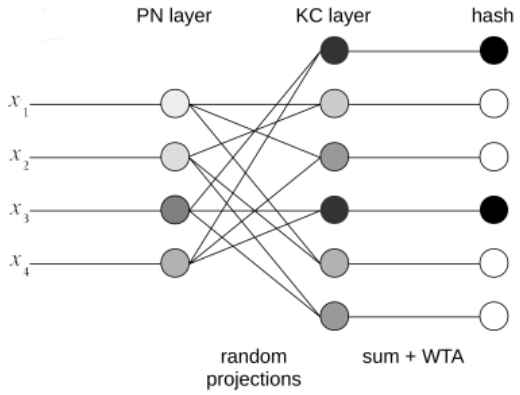


Figure 1: FFA architecture (figure adapted from Preissner and Herbelot (2020).

document. Since the activations of the hidden layer are already binarized, the representation naturally implements binary hashing.

4.2 The Fruit Fly Algorithm (FFA)

The Fruit Fly Algorithm (FFA) takes inspiration from the fruit fly's olfactory system. Specifically, the fruit fly's brain is composed of sparse connections between only two layers of neurons which can assign binary activations to a particular smell (defined as a combination of different types of chemicals). These patterns of activations allow the fruit fly to 'conceptualise' the environment and react to new smells by comparing them to previous smells the fly has been exposed to. Dasgupta et al. (2017) first proposed an implementation of FFA to hash existing pretrained embeddings. In our work, we extend the FFA to learn binary document embeddings from scratch, with low energy consumption.

Following the implementation of (Dasgupta et al., 2017), the FFA model consists of a small feedforward architecture which transforms a document to a binary vector to represent the hash of the document (Fig 1). The input layer, the *projection neuron layer* or *PN layer*, is a vector of d elements $\{x_1...x_d\}$, generated by the vectorization process described in §3.1. Next, this input layer is multiplied by a random projection matrix to form the input to the second layer. The second layer, (*Kenyon Cell layer* or *KC layer*), is represented as a vector of k elements $\{y_1...y_k\}$, which is *larger* than the PN layer ($k >> d$) and is kept at fixed size. The projection matrix is sparse, that is, PN and KC layers are not fully connected. Each KC is connected with a constant number i of nodes in the PN layer, and these connections are randomly allocated at initialization time. The activation value of each KC cell is then simply the sum of i activations from the PN layer. Note that although these connections are uniformly distributed, certain allocations will result in better performance. Finally, hashing is done by a winner-takes-all (WTA) function that sorts the activations in the KC layer, then takes only a small percentage of the most activated values to produce a compact representation of the document. Specifically, $WTA(y_i) = 1$ if y_i is one of the k top values in y and 0 otherwise.

Moreover, we include a Locality Sensitive Hashing (LSH) algorithm as a simple baseline. It is a vanilla random projection method with one non-binary weight matrix, followed by a binarization step to achieve hash values. Random projection LSH is similar to FFA, as the two are from the random projection group of algorithms.

5 BNN and FFA comparison

Our two algorithms have many differences, from their architecture to their optimization regime. We will highlight those differences here to make our experimental results more interpretable.

First, the BNN is a supervised method while

the FFA is in essence unsupervised. This means that when training and optimizing the BNN, we are looking for the best network weights *and* ideal hyperparameters (for both pre-processing and training regime). In contrast, the FFA does not require training of weights, since the projections are random, but still requires hyperparameter setting (including pre-processing and fly-specific features such as the number of random projections per KC or the WTA rate). Note that the BNN uses backpropagation while the FFA only performs forward propagation.

Second, the BNN's natural training grounds is the classification task, which allows us to use a straightforward and efficient objective function. The FFA being unsupervised, it can be optimized on any task that can make direct use of its binary embeddings. In practice, we found that optimizing the FFA's hyperparameters on *Prec@k* gave us better performance than the classification task, so the results we report are based on this choice.

Finally, we should note that the task of this paper, learning binary representations at low-dimensionality, is a natural setting for the BNN but actually challenges the FFA. The strength of the FFA is the massive expansion in dimensionality at the level of the KC layer. Through this mechanism, an actual fruit fly transforms a perceptual input in 50 dimensions into a conceptual representation contained in 2000 neurons. That 40-fold expansion in dimensionality allows the fly to capture many latent features of the perceptual data, some of which, presumably, end up being useful for classification.[8] But our task requires instead that the document features be compressed in at most 128 dimensions (down from 10,000 in input). As we will see later, this will necessitate some adjustments to the original FFA.

5.1 Evaluation

Classification Classification is simply the process of predicting a class for a given document. The accuracy for multi-label and single-label documents is computed as the sum of all true positives and negatives divided by the sum of all true and false positives and negatives from the dataset.

Prec@k Precision at k is a typical information retrieval task. Given some document representation v, the k nearest neighbours of v are computed. Precision is then given as the number of nearest neighbours that have the same class as v, divided by k. All datasets are evaluated with a k value of 100. In the case of multi-label documents, the precision is counted as correct if at least one of the labels is retrieved.

Carbon footprint Aside from task performance, we also measure how the algorithms compare in terms of energy use. For each system, and for each dataset, we compute electricity use in kWh. Since optimization happens differently in the BNN and the FFA, and involves different numbers of hyperparameters, we report the average consumption of a single run (one given set of hyperparameters) for each architecture. In order to show the environmental advantage of the BNN and FFA algorithms, we also report the consumption of a pretrained BERT model for comparison. We use the *CodeCarbon* library (Schmidt et al., 2021) to measure the electricity consumption.

6 Experimental Design

We implement a BNN and an FFA[9] with the aim of generating document hashes at 32, 64 and 128 bits. To evaluate the respective strengths of the two systems, we compare the two architectures according to the two methods described above: precision at k (*prec@k*) and classification (*acc*).

Further, we divide our evaluation of the FFA into three different settings, to investigate how the relation between the dimensionality of the input of the size of the KC layer affects results. Recall that the original FFA expects an expansion in dimensionality which is undesirable from the point of view of the task at hand, where we seek to obtain 32-64-128 bits hashes. To alleviate this issue, we attempt to combine the original architecture with a dimensionality reduction step implemented as Principle Component Analysis (PCA). We apply this step in two different conditions. In the first one (subsequently referred to as PCA+FFA), we apply PCA to the input matrix and feed the first c principal components to the FFA, where c is a hyperparameter to optimize. In the second one (FFA+PCA), we apply PCA 'inside' the FFA, just before the WTA step, in effect reducing the size of the KC layer. As control condition, we also show the results of the original FFA without intervention (henceforth 'Raw FFA').

[8] In that sense, one might argue that the natural fruit fly implements the kind of 'wastefulness' we criticised in Large Language Models – but only across two partially connected layers, and without backpropagation.

[9] Our code is freely available at https://github.com/minimalparts/SemanticHashingFFA.

Bits	Eval mode	BNN	LSH	Raw FFA	PCA + FFA	FFA + PCA
32	pre@k	**0.31**	0.08	0.25	0.30	0.25
	acc	0.45	0.17	0.38	0.45	**0.48**
64	pre@k	0.31	0.09	0.25	**0.35**	0.29
	acc	0.48	0.25	0.41	0.55	**0.56**
128	pre@k	0.32	0.11	0.27	**0.39**	0.29
	acc	0.54	0.38	0.48	**0.61**	0.59

Table 2: Results for 20news dataset

Bits	Eval mode	BNN	LSH	Raw FFA	PCA + FFA	FFA + PCA
32	pre@k	**0.79**	0.27	0.31	0.62	0.62
	acc	0.75	0.33	0.42	0.73	**0.78**
64	pre@k	**0.80**	0.27	0.64	0.69	0.59
	acc	**0.81**	0.37	0.43	0.80	0.80
128	pre@k	**0.80**	0.28	0.67	0.72	0.54
	acc	**0.86**	0.41	0.45	0.84	0.80

Table 3: Results for Agnews dataset

To perform classification with the FFA representations, we feed a simple Logistic Regression[10] model with the documents' hashes, as generated by the fly, and perform one-vs-rest classification.

For all settings, we tune the values of the two hyperparameters of the pre-processing stage: the log-probability exponent p and the number t of top words considered for each document (see §3.1). For the BNN, we also investigate the learning rate of the network. For the FFA, we tune the projection size and WTA rate, as well as the number of principal components retained from the PCA, wherever applicable. For the LSH, there is no tuning.

Since the FFA generates random projections for each fly it creates, we run it 10 times for each combination of dataset and hyperparameters, and select the instance with the best performance on the train set. (Recall that the FFA is unsupervised, so we simply compute *Prec@k* on the n documents seen in training.) We also run 10 times for LSH and average the results. All results reported in §7 are for the best models obtained from the optimization process.

7 Results

7.1 Task performance

Our results are reported in Tables 2 to 8. We provide the best hyperparameter sets in the appendix, for reproducibility purposes. It emerges that the BNN and the FFA complement each other, with one or the other algorithm taking the lead in particular combinations of datasets and tasks. We summarize the main trends in our results below, starting with a description of each experimental setting individually and then highlighting their respective strengths.

[10]Implementation from scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

Bits	Eval mode	BNN	LSH	Raw FFA	PCA + FFA	FFA + PCA
32	pre@k	**0.61**	0.25	0.50	0.58	0.54
	acc	**0.99**	0.45	0.49	0.62	0.56
64	pre@k	**0.68**	0.27	0.50	0.64	0.54
	acc	**0.99**	0.58	0.48	0.66	0.59
128	pre@k	**0.70**	0.29	0.49	0.66	0.50
	acc	**0.99**	0.74	0.50	0.68	0.61

Table 4: Results for Reuters dataset

Bits	Eval mode	BNN	LSH	Raw FFA	PCA + FFA	FFA + PCA
32	pre@k	0.53	0.24	0.41	0.56	**0.58**
	acc	**0.89**	0.42	0.06	0.19	0.20
64	pre@k	0.58	0.24	0.4	**0.61**	0.56
	acc	**0.91**	0.43	0.11	0.22	0.20
128	pre@k	0.59	0.25	0.43	**0.64**	0.53
	acc	**0.91**	0.46	0.14	0.25	0.21

Table 5: Results for TMC dataset

Bits	Eval mode	BNN	LSH	Raw FFA	PCA + FFA	FFA + PCA
32	pre@k	**0.67**	0.14	0.19	0.55	0.43
	acc	**0.82**	0.33	0.34	0.73	0.70
64	pre@k	**0.70**	0.18	0.21	0.63	0.45
	acc	**0.84**	0.46	0.43	0.81	0.79
128	pre@k	**0.69**	0.24	0.27	0.68	0.40
	acc	0.85	0.62	0.54	**0.86**	0.81

Table 6: Results for Wiki dataset

Bits	Eval mode	BNN	LSH	Raw FFA	PCA + FFA	FFA + PCA
32	pre@k	0.20	0.07	0.27	**0.35**	0.15
	acc	0.51	0.22	0.45	**0.58**	0.46
64	pre@k	0.22	0.09	0.28	**0.37**	0.14
	acc	0.56	0.35	0.46	**0.64**	0.51
128	pre@k	0.22	0.10	0.26	**0.40**	0.13
	acc	0.58	0.60	0.50	**0.71**	0.54

Table 7: Results for WoS dataset

BNN baseline: The BNN performs generally very well on the classification task, which is to be expected since it is specifically optimized for that task. It reaches accuracies over 90% on Reuters and

TMC, at all hash sizes. At 128 bits, it also achieves over 85% for Agnews and Wiki. Its performance is somewhat more disappointing on 20news and WoS (54% and 58% respectively). The results on *prec@k* are more variegated and, interestingly, do not fully follow the classification patterns. The best performance is on Agnews (around 0.8) followed by Reuters and Wiki (around 0.7 at 64 and 128 bits). TMC only reaches 0.59 while the performance on 20news and Wos is again lower (around 0.3 and 0.2 respectively).

Raw FFA: Results with the raw FFA are consistently low, although this setting surprisingly outperforms the BNN on WoS against the *prec@k* measure. This result is not entirely surprising, as the natural fly has a dramatic expansion factor of 40, projecting 50 PN neurons to 2000 KC cells. In our setting, conversely, the PNs get projected onto a *lower* number of KCs.

FFA with PCA postprocessing: Integrating a PCA dimensionality reduction step within the FFA, just before the WTA function, only occasionally improves performance. In some cases, it actually degrades the score of the Raw FFA.

FFA with PCA preprocessing: Out of all FFA settings, this is the one with the best performance. For classification, it does best on Wiki (86%) and WoS (71%), followed by Reuters (68%) and 20news (61%). For *prec@k*, the best results are obtained on Wiki, Reuters and TMC (0.68, 0.66 and 0.64 at 128 bits), followed by 20news and WoS (around 0.40).

LSH: It gets lowest scores for all datasets because it is the simplest algorithm, which serves as a good sanity check.

When comparing both algorithms, we see that for classification, PCA+FFA clearly outperforms the BNN on 20news and WoS, while very much failing to encode TMC and lagging behind on Reuters. Performance is comparable on the Wiki dataset and Agnews. As far as *prec@k* is concerned, PCA+FFA outperforms the BNN again on 20news and WoS, as well as TMC. The two algorithms have more comparable results on Reuters and Wiki, especially at higher hash sizes. The BNN obtains the highest results on Agnews.

7.2 Energy consumption

Table 9 shows energy consumption for the BNN and the PCA+FFA model, measured on a 32-core Linux machine using CPUs only (model AMD Opteron(TM) Processor 6272), with 32GB RAM. For comparison purposes, we also report the consumption of a fine-tuning procedure over a pre-trained BERT model, for the classification task. It is run parallel on a 4 x Nvidia GeForce GTX 1080 Ti, 12 GB. The table reports average figures for single optimization steps with an intuitive measure of electricity usage, by showing how many minutes one could run a single 40W light bulb for the same consumption. The results concerning the exact kWh consumption can be found in Table 1 of appendix A.1. While for the FFA, we give results for the overall consumption as well as a breakdown showing the individual demands of the PCA and the actual fruit fly, the results for BERT represent the fine-tuning of the model run once with the same hyperparameters across datasets [11]. Note that the values of consumption for BERT in the table only consider the fine-tuning of the model for the classification task, refer to Strubell et al. (2019) for more information about the pre-training consumption.

8 Discussion

8.1 Task performance

We first remark that as far as classification is concerned, it is possible to obtain high accuracies on nearly all datasets with at least one of our two lightweight algorithms: results at 128 bits range from 80% to 99% for Agnews, Reuters, TMC and Wiki. The more 'difficult' datasets are 20news and WoS, and interestingly, those are the ones where the FFA outperforms the BNN. As far as *prec@k* is concerned, a very similar picture emerges. Agnews, Reuters, TMC and Wiki reach between 0.69 and 0.80 precision, while 20news and WoS lag behind. Here again, the FFA outperforms the BNN by a very substantial margin.

One notable aspect of our results is that performance in classification does not necessarily transfer to *prec@k*, and vice-versa. The BNN achieves good classification accuracies on TMC but rather low prec@k, while the FFA behaves in exactly the opposite manner. Further, results vary widely depending on datasets, with the BNN and the FFA respectively leading the game on Reuters and WoS for both performance metrics. This seems to indicate that particular data distributions might be

[11] lr=3e-05, seed=42, number of training epochs=3, maximum sequence length=512, batch size for training and validating=8

	Similarity search				Classification			
dataset	**prec@100**	**KC size**	**proj size**	**WTA**	**Acc.**	**KC size**	**proj size**	**WTA**
20news	0.12	14896	10	45	0.69	14239	2	100
agnews	0.32	14885	10	80	0.91	9106	2	100
reuters	0.48	4462	10	62	0.73	14033	2	100
tmc	0.46	14866	10	94	0.22	8233	2	57
wiki	0.24	13848	10	35	0.92	14336	2	100
wos	0.15	2337	7	3	0.82	11236	2	100

Table 8: Bayesian Optimization on the raw FFA, with (nearly) unlimited KCs.

dataset	**LSH**	**BNN**	**PCA+Fly = FFA (Overall)**	**BERT**	**Diff. BERT**
20news	19	13	2+18 = 20	548	31x
agnews	34	41	87+81 ≈ 169	20153	247x
reuters	8	27	1+7 = 8	288	7x
tmc	18	16	4+16 = 20	1076	25x
wiki	31	21	6+22 ≈ 27	1281	48x
wos	11	11	1+9 ≈ 11	320	12x

Table 9: Energy consumption of the models in minutes run in a 40W lightbulb (more info about the kWh consumption in appendix A.1). Figures represent one complete run of the models with one pre-determined set of hyperparameters. Values in red and green are the models spending the most and least energy respectively. *Diff. BERT* represents the number of times BERT consumes more over the avg. of all tiny models.

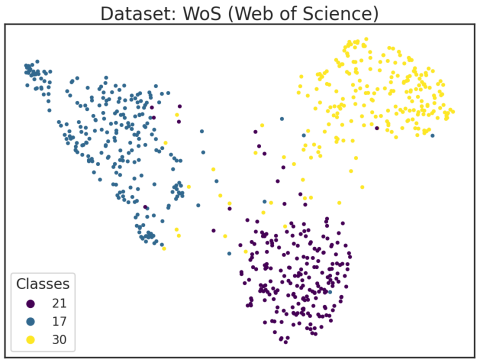
better captured by one or the other algorithm.

Unlike the BNN, the FFA demonstrates a consistent improvement as the hash size increases. This is not surprising, as we mentioned earlier that the FFA is in some sense designed to work at high dimensionality. We investigated this effect further and performed Bayesian Optimization[12] on the FFA's hyperparameters, this time allowing the size of the KC layer to grow up to 15,000 bits. Results are shown in Table 8. Against expectation, a higher KC size does not necessarily translate into better *prec@k*. And while the higher size does make a substantial difference to the classification task, we note that this is obtained with very low projection sizes, meaning that the algorithm reverts to looking at single words in the output.

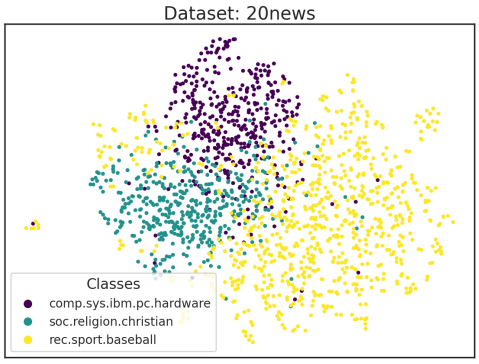
An inspection into the hashes shows that the representation derived from the FFA is able to create a distinctive semantic space between documents from different labels. Fig. 2 shows a 2D representation of four randomly-chosen classes in WoS and 20news.

8.2 Energy efficiency

Both the BNN and FFA algorithms prove to be very energy efficient. In table 9, we see that a single run of the BNN is equivalent to running the light bulb between 11 and 41 minutes, depending on the size of the dataset being processed. The FFA itself (without PCA pre-processing) has a wider range, from 7 to 81 minutes, but still remains very



(a) neighbors=50, dist=0.1



(b) neighbors=15, dist=0.8

Figure 2: Dimensionality reduction of the 128 dimension vector from FFA with UMAP. Classes are randomly selected.

[12]Library from github.com/fmfn/BayesianOptimization

affordable.

We note that the main cost of the FFA is of course evaluation. We are optimizing the algorithm on *Prec@k*, which is an expensive function to run, as it involves a computationally-intensive nearest neighbour computation. As we pointed out previously, optimizing on classification did not seem to give the best possible results for the FFA. But this aspect would require further investigation, since it is the main efficiency bottleneck for the algorithm.

We also note that when including PCA preprocessing, large datasets like agnews become more expensive to run: the highest overall consumption in the tiny models in the table comes from to the PCA on agnews (81 minutes). This is a substantial cost in getting the best out of the FFA, and one that could potentially be reduced. In particular, we are experimenting with computing dimensionality reduction on a restricted subset of our data and subsequently learning a regression function from high- to low-dimensional space to 'simulate' the effect of the PCA. Preliminary results are encouraging, with limited loss in task performance.

Lastly, the discrepancy of energy consumption is striking between our tiny models and BERT as observed in columns *BERT* and *Diff. BERT* of table 9. The latter requires a considerable larger amount of resources because of the millions of parameters being updated during fine-tuning. Across datasets, it consumes 61 times more than the average consumption of all our tiny models in a single run. Note that the difference increases according to the size of the dataset. For instance, the smallest datasets Reuters and WoS consume 7 and 12 times more than the tiny models respectively, while the largest dataset (agnews) consumes 247 times more. These results highlight the importance of putting more effort in developing smaller models because of the high energy costs of solely fine-tuning these huge models.

9 Conclusion

This paper set out to compare the respective strengths of two lightweight binary hashing algorithms, the binary neural network (BNN) and the fruit fly algorithm (FFA), on the task of generating highly-compressed document representations. We adapted the original FFA to this new context by prepending a dimensionality reduction step to the architecture, implemented with PCA. We found that the two methods display different strengths and achieved their top performance on different tasks and datasets. Both are energy-efficient, with the FFA's consumption being mostly taken by evaluation at optimization stage.

Both BNNs and biologically-inspired algorithms are relatively new in NLP, and therefore require a lot of community efforts to fully understand their respective behaviours. As immediate further work, we would perform an in-depth analysis on our datasets' distributions to understand better why some data seem more suited to one or the other algorithm, and why discrepancies emerge in the way that classification and precision at k are tackled. From an efficiency point of view, we will also further investigate how to reduce the cost of the dimensionality reduction step before applying the FFA.

One of the main strengths of both models is their low running costs. All steps can be run in a CPU or even on a mobile phone – as in the case of the BNN. This is a crucial point when it comes to providing high quality systems based on artificial intelligence models for low-resource communities. As an example application, we have integrated the PCA+FFA pipeline to the PeARS search engine.[13] PeARS implements local Internet search by allowing users to index and search Web documents on their home machine. It requires an indexing method that is as lightweight as possible for users with limited hardware resources. The FFA is a natural choice here, with its good performance on the *Prec@k* metric. We hope that the work presented in this paper will inspire other researchers to invest effort in developing lightweight techniques to solve core NLP problems, and share them with the communities that benefit from them.

Acknowledgments

We gratefully acknowledge the funding provided by the NLnet Foundation, with financial support from the European Commission's Next Generation Internet programme, under the aegis of DG Communications Networks, Content and Technology (grant agreement No 825322).

[13] https://pearsproject.org/, code at https://github.com/PeARSearch/PeARS-orchard.

References

Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. 2019a. Docbert: Bert for document classification. *arXiv preprint arXiv:1904.08398*.

Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. 2019b. Rethinking complex neural network architectures for document classification. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4046–4051, Minneapolis, Minnesota. Association for Computational Linguistics.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big? . In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 610–623, New York, NY, USA. Association for Computing Machinery.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Suthee Chaidaroon and Yi Fang. 2017. Variational deep semantic hashing for text documents. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 75–84.

Sanjoy Dasgupta, Charles F Stevens, and Saket Navlakha. 2017. A neural algorithm for a fundamental computing problem. *Science*, 358(6364):793–796.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. 2021. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. *arXiv preprint arXiv:2104.08758*.

Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. 2020. Unsupervised semantic hashing with pairwise reconstruction. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2009–2012.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. *Advances in neural information processing systems*, 29.

Kamran Kowsari, Donald E Brown, Mojtaba Heidarysafa, Kiana Jafari Meimandi, Matthew S Gerber, and Laura E Barnes. 2017. Hdltex: Hierarchical deep learning for text classification. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)*, pages 364–371. IEEE.

Ken Lang. 2008. The 20 news groups data set. *http://people. csail. mit. edu/jrennie/20Newsgroups/*.

David D. Lewis. 1997. Reuters-21578 text categorization collection data set.

Yuchen Liang, Chaitanya K. Ryali, Benjamin Hoover, Leopold Grinberg, Saket Navlakha, Mohammed J. Zaki, and Dmitry Krotov. 2021. Can a fruit fly learn word embeddings?

Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. 2017. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*, pages 115–124.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Nikunj Oza. 2010. Siam 2007 text mining competition dataset.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Simon Preissner and Aurélie Herbelot. 2019. To be fair: a case for cognitively-inspired models of meaning. In *Proceedings of the Sixth Italian Conference on Computational Linguistics*.

Simon Preissner and Aurélie Herbelot. 2020. Biodiversity in nlp: modelling lexical meaning with the fruit fly algorithm. *IJCoL. Italian Journal of Computational Linguistics*, 6(6-1):11–28.

Gerard Salton and J Michael. 1986. Mcgill. *Introduction to modern information retrieval*, 1(4.1):4–1.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Victor Schmidt, Kamal Goyal, Aditya Joshi, Boris Feld, Liam Conell, Nikolas Laskaris, Doug Blank, Jonathan Wilson, Sorelle Friedler, and Sasha Luccioni. 2021. CodeCarbon: Estimate and Track Carbon Emissions from Machine Learning Computing.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28.

Yifei Zhang and Hao Zhu. 2019. Doc2hash: Learning discrete latent variables for documents retrieval. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2235–2240.

A Appendix

A.1 Energy consumption

dataset	LSH	BNN	PCA+Fly = FFA (Overall)	BERT
20news	0.013	0.009	0.001+0.012 = 0.14	0.365
agnews	0.022	0.028	0.058+0.054 = 0.112	13.435
reuters	0.005	0.018	0.001+0.005 = 0.006	0.192
tmc	0.012	0.011	0.003+0.011 = 0.014	0.717
wiki	0.021	0.014	0.004+0.015 = 0.018	0.854
wos	0.008	0.007	0.001+0.006 = 0.007	0.213

Table 1: Energy consumption of the models in minutes run in kWh. Figures are averages for single optimization steps. Values in red and green are the models spending the most and least energy respectively.

A.2 Model hyperparameters

SentencePiece was run with default hyperparameters and a vocabulary size of $10,000$. For the BNN, the best log-probability exponent p varied depending on the dataset: 3 for Reuters, Agnews and TMC, 4 for 20news and WoS, 5 for Wiki. For the FFA, $p = 4$ emerged as the best choice for all datasets. The number of top words t gave optimal results at $t = 300$ for most datasets, apart from WoS ($t = 500$).

Tuning the learning rate for the BNN did not result in any statistically significant changes, so all results are reported for $lr = 0.01$.

For the FFA, we tuned the projection size from 4 to 16 and the WTA rate from 10 to 70. The best hyperparameter combinations for each pair of hash size and dataset are included in the table below. The Logistic Regression classifier used the default sklearn hyperparameters, in multiclass mode.

	32 bits		64 bits		128 bits	
dataset	WTA	proj size	WTA	proj size	WTA	proj size
20news	16	70	16	50	16	50
agnews	8	50	8	30	4	50
reuters	4	50	4	50	4	50
tmc	4	50	8	50	12	30
wiki	8	50	8	30	8	30
wos	8	70	8	50	4	70

Table 2: Hyperparameters table.