Comply: Learning Sentences with Complex Weights inspired by Fruit Fly Olfaction

Alexei Figueroa[1]   **Justus Westerhoff**[1◇]   **Golzar Atefi**[1◇]   **Dennis Fast**[1]
Benjamin Winter[2*]   **Felix Alexader Gers**[1]   **Alexander Löser**[1]   **Wolfang Nejdl**[3]

[1]DATEXIS, Berliner Hochschule für Technik (BHT)
[2]Medical AI Analytics & Information GmbH
[3]L3S, Leibniz University Hannover
{afigueroa,justus.westerhoff}@bht-berlin.de
◇ Authors contributed equally.

Abstract

Biologically inspired neural networks offer alternative avenues to model data distributions. FlyVec is a recent example that draws inspiration from the fruit fly's olfactory circuit to tackle the task of learning word embeddings. Surprisingly, this model performs competitively even against deep learning approaches specifically designed to encode text, and it does so with the highest degree of computational efficiency. We pose the question of whether this performance can be improved further. For this, we introduce Comply. By incorporating positional information through complex weights, we enable a single-layer neural network to learn sequence representations. Our experiments show that Comply not only supersedes FlyVec but also performs on par with significantly larger state-of-the-art models. We achieve this without additional parameters. Comply yields sparse contextual representations of sentences that can be interpreted explicitly from the neuron weights.

1 Introduction

A large variety of deep learning models have centered on creating distributional language representations in recent years. Among these, Transformer-based architectures such as BERT [Kenton and Toutanova, 2019] have consistently set the state-of-the-art and are now considered baselines for Natural Language Processing (NLP) tasks. However, better performance in metrics has come at the expense of a seemingly ever-growing computational complexity. Naturally, there is a competing interest in developing efficient architectures, and often, these are inspired by intelligent biological systems that are efficient by nature. Recently, FlyVec was proposed as a simple approach for creating contextual word embeddings motivated by the olfactory circuit of the fruit fly [Yuchen et al., 2021]. These embeddings are binary vectors (hash codes) that are formed through neural inhibition via the k-Winner-Takes-All (k-WTA) activation. While FlyVec is competitive with traditional word embedding methods like word2vec [Mikolov et al., 2013] and GloVe [Pennington et al., 2014], it is still superseded by contextual embedding methods such as BERT [Kenton and Toutanova, 2019]. Although FlyVec demonstrates an elegant approach that is more computationally efficient, it motivates the question of whether the performance can still be improved, while maintaining the same scale and architectural simplicity.

We attempt to tackle this by, yet again, gathering inspiration from how fruit flies learn to identify odors. Previous research has focused on the spatial bundling and modularity of neurons to create high-dimensional representations. However, olfactory processing also involves processing temporal

---

patterns, e.g., firing delays and duration. We highlight two ways in which fruit flies utilize temporal information.

Relevance of odor arrival time: The way fruit flies perceive a mixture of odors hints at how the arrival order of stimuli leads to better segregation and classification capabilities. Multiple odors can be emitted either from the same source or from different sources. Odors emitted from a single source have an approximately simultaneous arrival time due to the physical process of turbulent diffusion, as opposed to odors propagating from multiple sources, which results in distinct arrival times. Several studies have shown that fruit flies [Szyszka et al., 2012, Hopfield, 1991, Sehdev et al., 2019], as well as other insects [Baker et al., 1998, Saha et al., 2013, Nikonov and Leal, 2002, Witzgall and Priesner, 1991, Andersson et al., 2011], likely use this asynchronous arrival time to perceptually segregate mixed odors.

Using time as a medium to encode odors: Olfactory receptor neurons (ORNs) respond in a temporally heterogeneous manner that depends on both the type of odor and the type of ORNs [Raman et al., 2010]. This heterogeneity encompasses the firing rate and the time course (latency and duration), which results in odor identification irrespective of stimulus concentration [Stopfer et al., 2003]. Mutant flies that are ablated of spatial coding retain odor discriminatory capacity only through the temporal responses of ORNs [DasGupta and Waddell, 2008, Olsen et al., 2007].

These two perspectives regarding time illustrate how spatial encoding (in the sense of neuron collocation) is *not the only process* employed to generate odor embeddings. We investigate the impact of integrating this temporal information in the quality of the embeddings. Our method, `Comply`, generalizes `FlyVec` to incorporate time information (word positions) in the analogy of mirroring text sequences (sentences) to odor segregation. This is accomplished by expressing the input with phase differences and exploiting the semantics of complex numbers. Consequently, we learn a complex-valued parameter matrix through an extension of an unsupervised energy function. We subject our models to a comprehensive evaluation in semantic sentence similarity tasks, as well as a text-based Reinforcement Learning environment. Our experiments show how the features generated by `Comply` boost the performance significantly, while preserving the parameter complexity, computational efficiency, and biological plausibility of `FlyVec`. The main contributions of this work can be summarized as follows:

- We introduce a novel architecture that can produce sparse encodings (binary hashes) incorporating positional information, while preserving computational efficiency and significantly boosting performance.
- Through a comprehensive quantitative evaluation of the unsupervised representations, we show the enhanced quality of the produced embeddings, being competitive with significantly larger architectures in NLP.
- We showcase qualitatively that the simplicity of our approach, `Comply`, retains interpretable properties of `FlyVec`, while extending to the temporal dimension of sequences.

We make the source code[2] available to reproduce the results of our experiments.

2 Related Work

Bio hashing. The mushroom body of the fruit fly is considered to be the center of olfactory learning and memory formation [de Belle and Heisenberg, 1994]. It is where projection neurons (PN) make synaptic connections with Kenyon cells (KC). In contrast to PNs, KCs have much more odor-selective responses [Turner et al., 2008, Honegger et al., 2011]. In other words, only a small proportion of them becomes activated for a given odor due to inhibition [Lin et al., 2014, Papadopoulou et al., 2011]. Therefore, the KCs represent odors as sparse and distributed hash codes [Lin et al., 2014]. These codes are high-dimensional representations that the mushroom body output neurons (MBONs) use to form memories. The sparsity and high specificity of these KC responses are thought to support the accuracy of this process [Laurent, 2002].

The `FlyVec` [Yuchen et al., 2021] model repurposes bio hashing into learning word embeddings from raw text. Using a one-layer neural network from PNs to KCs, it creates representations of a

---

[2]https://github.com/DATEXIS/comply

binarized input of context and target words. The synaptic weights from PNs to KCs are learned with a biologically plausible unsupervised learning algorithm [Krotov and Hopfield, 2019] inspired by Hebbian learning. During inference, the k-Winner-Takes-All (k-WTA) mechanism creates sparse, binary hash codes.

Our method, `Comply`, uses a similar architecture and algorithm. However, we augment the inputs with positional information and extend the loss and parameters accordingly. Specifically, `Comply` generalizes `FlyVec`, enabling bio hashes to represent sequences.

Text embeddings. The main goal of embedding text is to produce vectors that have a semantic correspondence to characters, words, or arbitrary sequences of text elements (tokens). Examples are `word2vec` [Mikolov et al., 2013], `GloVe` (Global Vectors) [Pennington et al., 2014], and `BERT` (Bidirectional Encoder Representations from Transformers) [Kenton and Toutanova, 2019]. The former two are examples of static word embeddings, which are constant regardless of context. In contrast, `BERT` learns a function that can yield representations conditioned on the context: a contextual word embedding.

Transformer-based encoder models like `BERT` learn language representations using self-attention and masked language modeling among other objectives, which are not necessarily motivated by biological mechanisms. These representations are "contextual", i.e., they capture the ambiguity of words (sentences) given the context in which they appear.

The `FlyVec` model highlights how a biologically inspired architecture is also capable of learning contextual *word* embeddings (binary hash codes). In our work, we extend this model to *sentence* representations. We do this by learning positions as a complementary distributional property of text.

Positional encoding. Recurrent Neural Networks (RNNs) [Rumelhart et al., 1986], e.g., Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997, Gers et al., 2000], inherently capture positional information given that their state (and output) is always conditioned on the sequential input. Similarly, Convolutional Neural Networks (CNNs) can implicitly learn to incorporate positional information [Islam et al., 2020]. Although self-attention [Lin et al., 2017] is designed to focus on different parts of the sequence depending on the task, it does not infer positional information. When introducing Transformers, the work [Vaswani et al., 2017] explicitly incorporates positional encodings in an additive manner. In fact, this component is crucial for the success of these architectures [Sinha et al., 2021]. Closer to our work, `RoPe` [Su et al., 2024] expresses positional information in Transformer models through a multiplicative approach. Namely, an embedding vector of a token is *rotated* based on its position. Although this approach and ours are both non-parametric and leverage the complex plane to implement such rotations, we differ by explicitly expressing the input in the complex field and learning a complex parameter matrix.

Complex weights. Complex-valued Neural Networks (CVNNs) offer advantages in specific domains, as they can handle both amplitudes and phases of the signals. Examples of such areas are image processing, and more generally signal processing where a datum is inherently complex. For a comprehensive overview see, e.g., [Lee et al., 2022]. Close to our work, the method in [Wang et al., 2020] uses a learned complex-valued map to enrich text embeddings with positional information. In our work, using the semantics of the complex field yields a simple mathematical extension of the `FlyVec` model. While CVNNs focus on end-to-end functions in the complex field, we use complex numbers only as a way of efficiently representing the order in sequences learned in neurons. However, we do *not* use complex activation functions, our embeddings are purely real and are constructed with the k-WTA mechanism.

3 Methods

3.1 `FlyVec` [Yuchen et al., 2021].

This model adapts a single parameter matrix $\mathbf{W} \in \mathbb{R}^{K \times 2N_{voc}}$, modeling the PNs→KCs synapses (projection neurons to Kenyon cells) and using a biologically inspired mechanism. Here, K is the number KCs, and N_{voc} is the size of the word vocabulary. For training, every input sentence is split into sliding text windows of fixed length $\omega \in \mathbb{N}$ disregarding positions, thus modeling sets (bag-of-words). The inputs $\mathbf{v} \in \mathbb{R}^{2N_{voc}}$ are crafted as follows: the first N_{voc} components are the sum

You get more flies with honey than with vinegar.

context | **target**

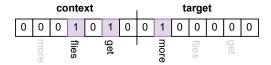| more	flies	get		more	flies	get
0	0	0	1	0	1	0	0	1	0	0	0	0	0

Figure 1: Sample input vector $\mathbf{v} \in \mathbb{R}^{2N_{\mathrm{voc}}}$ to the `FlyVec` model. The two blocks (context and target) of length N_{voc} are separated by the long vertical line in the middle. Here, "more" is the *target word* and sliding window $\omega = 3$. Adapted from [Yuchen et al., 2021], Figure 2, p.3.

of one-hot encoded vectors of context tokens surrounding a *target word*, and the second N_{voc} entries are a single one-hot vector corresponding to the *target word*, see fig. 1. The parameter matrix $\mathbf{W}$ is learned by minimizing the energy function

$$E := - \sum_{\mathbf{v} \in Data} \frac{\langle \mathbf{W}_{\widehat{\mu}}, \mathbf{v}/\mathbf{p} \rangle}{\langle \mathbf{W}_{\widehat{\mu}}, \mathbf{W}_{\widehat{\mu}} \rangle^{1/2}}, \tag{1}$$

where

$$\widehat{\mu} := \operatorname*{argmax}_{\mu \in \{0, \dots, K-1\}} \langle \mathbf{W}_{\mu}, \mathbf{v} \rangle, \tag{2}$$

$\mathbf{v}/\mathbf{p}$ is the input vector as described above divided by the word frequencies $\mathbf{p} \in \mathbb{N}^{N_{\mathrm{voc}}}$ in the whole training corpus, and $Data$ is the set of preprocessed inputs. The model emphasizes learning through lateral inhibition, i.e., during training, there is only a single neuron $\widehat{\mu}$ (a row $\mathbf{W}_{\widehat{\mu}}$ in the $\mathbf{W}$ matrix), which is adapted for a single sample.

3.2 `Comply`

Our extension revolvese around capturing the sequential nature of text data through added positional information. We leverage the semantics of complex numbers and keep the same biologically inspired mechanisms, namely sparsity through k-Winner-Takes-All (k-WTA) activations. We aim to preserve the properties of `FlyVec` regarding efficiency, simplicity, and interpretability. Thus, we constrain ourselves to a single-parameter matrix. Additionally, we focus on an extension that considers a complex-valued input, a single complex parameter matrix $\mathbf{W}$, and a compatible energy function E.

Complex sequential input. For each *sentence* expressed as a concatenation of one-hot encoded vectors, we multiply each vector by $e^{i\pi \frac{l}{L}}$, where L is the length of the sentence, $l \in \{0, 1, \dots, L-1\}$ denotes the position of the respective word in the sentence and i is the imaginary unit, see fig. 2. We denote this preprocessed input by $\mathbf{z} \in \mathbb{C}^{L \times N_{\mathrm{voc}}}$ and its complex conjugate by $\overline{\mathbf{z}}$. Note that this
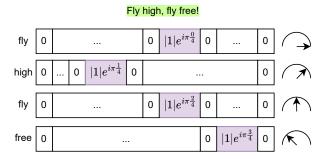
Fly high, fly free!

fly	0	...	0	$|1|e^{i\pi\frac{0}{4}}$	0	...	0	
high	0	...	0	$|1|e^{i\pi\frac{1}{4}}$	0	...	0	
fly	0	...	0	$|1|e^{i\pi\frac{2}{4}}$	0	...	0	
free	0	...	0	$	1	e^{i\pi\frac{3}{4}}$	0	

Figure 2: Sample input $\mathbf{z} \in \mathbb{C}^{L \times N_{\mathrm{voc}}}$ to our `Comply` model, where $L = 4$ is the length of the sample sentence "*Fly high, fly free!*". Positions are encoded as the complex $2L$th roots of one in the upper half-plane.

modification, compared to `FlyVec`, expands the dimension of the input by L, and reduces the length of each word vector by half since we do not have a target word as in fig. 1. Consequently, our

parameter matrix $\mathbf{W}$ is in $\mathbb{C}^{K \times N_{\text{voc}}}$ and has the same number of parameters as `FlyVec` ($2KN_{\text{voc}}$), since for each complex parameter two real numbers are required.

With this modified input, an immediate extension of eq. (1) with the Hermitian inner product $\langle \cdot, \cdot \rangle_H$ is not directly applicable, since for a sentence the quantity

$$\sum_{l \in L} \langle \mathbf{W}_\mu, \mathbf{z}_l \rangle_H = \sum_{l \in L} \mathbf{W}_\mu^T \overline{\mathbf{z}}_l \ \in \mathbb{C} \tag{3}$$

is complex and $\mathbb{C}$ is unordered, hence finding the $\arg\max$ as in eq. (2) is not possible. Thus, we adapt E to find a coherent maximally activated neuron $\widehat{\mu}$.

Adapting E. Due to the complex-valued nature of the Hermitian inner product, an alternative extension must be considered. If we disregard word repetition, we can think of eq. (3) as $\langle \mathbf{W}_\mu, \tilde{\mathbf{z}} \rangle_H$, where $\tilde{\mathbf{z}} := \sum_{l \in L} \mathbf{z}_l$ is a multi-hot encoded version of the input (similar to `FlyVec`). We can decompose $\langle \tilde{W}_\mu, \tilde{\mathbf{z}} \rangle_H$ into two steps, element-wise multiplication and aggregation, and yield real values that are proportional to neuron activation and can be sorted:

- *Element-wise multiplication*: For every word and neuron, we perform a complex multiplication, which translates to scaling absolutes and subtracting phases. The former indicates the correlation between the input *intensity* and the neuron's output, while the latter gives us a measure of the distance (*temporal difference*) between the word in neuron $\mathbf{W}_\mu$ and the sentence.

- *Aggregation*: Simply adding these complex numbers does not retain positional difference information, since $e^{i\theta} = e^{i2\pi\theta}$ for any $\theta \in \mathbb{R}$. Hence, we instead sum the magnitudes and absolute values of phase differences to capture both the effect of maximum word activation (as in the `FlyVec` model) and the word position distance.

Equation (4) reflects these modifications to the energy function E. Note that the energy of the original `FlyVec` model (see eq. (1)) is a special case of our approach when $\mathbf{W}$ is real and the inputs are preprocessed as in fig. 1. We slightly abuse notation by writing L and $\{1, \ldots, L\}$ interchangeably, and define

$$E := -\sum_{\mathbf{z} \in Data} \left(\frac{\sum_{l \in L} \left| \langle \mathbf{W}_{\widehat{\mu}}, \mathbf{z}_l / \mathbf{p}_{s_l} \rangle_H \right|}{\langle \mathbf{W}_{\widehat{\mu}}, \mathbf{W}_{\widehat{\mu}} \rangle_H^{1/2}} \right.$$
$$\left. + \sum_{l \in L} \left| \text{Arg} \langle \mathbf{W}_{\widehat{\mu}}, \mathbf{z}_l / \mathbf{p}_{s_l} \rangle_H \right| \right), \tag{4}$$

where

$$\widehat{\mu} := \underset{\mu \in \{0, \ldots, K-1\}}{\arg\max} \sum_{l \in L} \left| \langle \mathbf{W}_\mu, \mathbf{z}_l \rangle_H \right| + \left| \text{Arg} \langle \mathbf{W}_\mu, \mathbf{z}_l \rangle_H \right|, \tag{5}$$

Arg is the function that extracts the phases from the complex numbers, and s_l is the index in the vocabulary of the lth word in a sentence $[s_0, \ldots, s_{L-1}]$. Equation (4) favors learning complex weights for words in one half of the complex plane, since the absolute angle difference $|\text{Arg} \cdot|$ is maximized, and its maximum is at π. If we replaced the additions in both eqs. (4) and (5) with subtractions, the synapses would be learned on the same half-plane as the input vectors $\mathbf{z}$. Either option is of no consequence for the model and we choose the additive expression for our experiments. Furthermore, we highlight that the two summands in eq. (5) are not of the same scale; the former, $|\langle \mathbf{W}_\mu, \mathbf{z}_l \rangle_H|$, is in $[0, \infty)$, whereas the latter, $|\text{Arg}\langle \mathbf{W}_\mu, \mathbf{z}_l \rangle_H|$, is in $[0, \pi]$. To consider this, we experimented with replacing addition (+) by multiplication (·) in eq. (4) and eq. (5). This however resulted in instability leading to model overfitting (multiple neurons learn the same sequences) and, more prominently, would not constitute a generalization of the original energy presented in eq. (1). Nevertheless, in the case of inference, we also test the multiplicative approach.

Note that, as depicted in fig. 2, we limit the phases in the input $\mathbf{z}$ to lie in $[0, \pi)$ to enforce learning phases in $\mathbf{W}$ that also only span half of the complex plane. This resolves the ambiguity of having distinct positions in word phases that would be mapped equivalently in the phase distance component of our energy function, e.g., a word appearing in $\frac{-\pi}{2}$ and $\frac{\pi}{2}$ and the weight of that word in $\mathbf{W}_\mu$ with phase 0.

To summarize, the semantics and operations of complex numbers naturally give us a framework to generalize the objective of minimizing E, while capturing positional information.
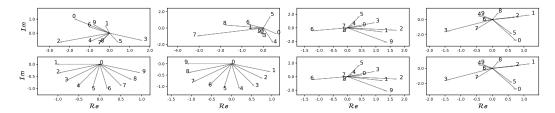
Figure 3: Toy example of training with two sequences $1, 2, \ldots, 9$ and $9, 8, \ldots, 1$ modified as in fig. 2 with phases in $[0, \pi)$. Top row: The randomly initialized complex weights of a model with four Kenyon cells. Bottom row: Weights after learning with eq. (4). The loss makes the model memorize the sequences in the lower half of the complex plane (phases in $[-\pi, 0)$) in the first two neurons. The other neurons remain unchanged.

Toy example. We illustrate the mechanics of the loss presented in eq. (4) with a toy example that uses only four KCs ($K = 4$) and ten words (digits) ($N_{\text{voc}} = 10$) for two sentences (sequences): $1, 2, \ldots, 9$ and $9, 8, \ldots, 1$. The model is trained with backpropagation following eq. (4). Figure 3 shows the initial state in the complex plane for each neuron, as well as the converging state after learning. Both sequences are explicitly learned in the network in the opposite half-plane of the input (that is, with angles in $[-\pi; 0)$). The 1-WTA activation (eq. (4)) successfully creates a sparse representation of the two sequences *without interference*. This can be observed since the remaining neurons are unchanged and the two sequences are learned (*imprinted*) in two distinct neurons.

Binary hashes. We construct binary feature vectors for `Comply` from the order of activations of the KCs. Specifically, as in [Ryali et al., 2020], the output vector $\mathbf{h} \in \{0, 1\}^K$ of an input $\mathbf{z}$ is defined for all $\mu \in \{1, \ldots, K\}$ as

$$\mathbf{h}_\mu := \begin{cases} 1, & \text{if } \sum_{l \in L} \left| \langle \mathbf{W}_\mu, \mathbf{z}_l \rangle_H \right| + \left| \text{Arg} \langle \mathbf{W}_\mu, \mathbf{z}_l \rangle_H \right| \\ & \quad \text{is in the top-}k \text{ of all KC activations,} \\ 0 & \text{otherwise,} \end{cases} \tag{6}$$

where $k \in \mathbb{N}$, the hash length, is a hyperparameter. Since the two summands are, as noted earlier, of different scales, we also experiment with a version we call `ComplyM` that does multiplication ($\cdot$) instead of addition ($+$) in eq. (6) above.

4 Experiments

We evaluate our methods in sentence semantic similarity tasks, as well as in a text-based Reinforcement Learning (RL) environment.

4.1 Baselines

We focus on `FlyVec` since it is the architecture we extend. For the sentence similarity tasks, we also compare against `BERT` embeddings which are contextual representations based on the Transformer architecture. We use the `bert-base-uncased` weights publicly available on `Hugging Face`.[3] Likewise, we use the publicly available weights of `FlyVec`.[4] Following the evaluation in [Yuchen et al., 2021], we constrain the tokenizer of `BERT` to the top 20,000 word indices for a fair comparison against `FlyVec` and our models.

In the RL evaluation, we compare against a `FlyVec` model that we pretrain on PubMed (`P-FlyVec`), and the best performing Transformer-based encoder model in [Winter et al., 2024], denoted as `Transformer-MLM`. This architecture provides a policy with an auxiliary language modeling objective that is crucial to stabilize Transformer-encoder learning in online RL settings. We use this approach for both `P-FlyVec` and `Comply`.

---

[3] `https://huggingface.co/google-bert/bert-base-uncased`
[4] `https://pypi.org/project/flyvec/`

4.2 Data and model training

We use large text corpora to train the models for our experiments. Sentences are the largest sequence structure that we model with `Comply` (see section 3.2) and we create batches accordingly. Although the sliding window approach is only relevant for `FlyVec` (see fig. 1), we use it in our implementation only to construct batches of the same shape (phases of the input are independent of this), and guarantee that `Comply` sees tokens as often as `FlyVec`. We pretrain our models on two different text corpora depending on the domain of the target task:

Open Web Corpus (OWC) [Gokaslan and Cohen, 2019]. For the sake of comparability with [Yuchen et al., 2021], we use the OWC dataset to train `Comply`. It consists of roughly eight million web documents filtered to be longer than 128 tokens in English, resulting in about 40GB of text files. We preprocess and encode these text data with the `FlyVec` tokenizer provided in [Yuchen et al., 2021]. It maps every word to a single integer in a vocabulary of 20,000 IDs, resulting in approximately 6.5B tokens. This model is evaluated against the semantic similarity tasks.

PubMed. We construct a dataset from the PubMed database[5] keeping only abstracts of articles as in [Gu et al., 2021]. Their model, `PubMedBERT`, highlights that in-domain abstracts are sufficient to achieve competitive results in biomedical tasks. We filter for abstracts in English and use this tokenizer to yield approximately 6.9B tokens. We pretrain both `P-FlyVec` and `Comply` with these data. The resulting models have the same number of parameters and we use them in the RL evaluation.

4.3 Semantic similarity evaluation

We emphasize that `Comply` creates representations for sentences. We are interested in evaluating the pretrained embeddings learned in an unsupervised manner *without further fine-tuning*. Hence, we select the subset of tasks compiled in the Massive Text Embedding Benchmark (MTEB) [Muennighoff et al., 2023] that targets sentence embeddings without additional models (classification heads or clustering algorithms). Furthermore, we focus on tasks comprising English sentences. This results in the majority of the Semantic Textual Similarity (STS), as well as Pair Classification tasks.

The STS tasks are the SemEval workshop tasks STS12-17 in addition to BIOSESS, SICK-R, and STSBenchmark. In these tasks, the similarity between two sentences is assessed based on the embeddings produced by a model. For `FlyVec` and `Comply`, we construct their respective contextual feature hashes and for `BERT` we use the average of all token embeddings. We report Spearman rank correlation based on cosine similarity as in [Reimers et al., 2016].

In the Pair Classification (PC) tasks, a model predicts whether two sentences are duplicates in a binary setting. The metric reported for these PC tasks is the average precision based on cosine similarity. These tasks are SprintDuplicateQuestions, TwitterSemEval2015, and TwitterURLCorpus.

Finally, we include the Words in Context (WiC) [Pilehvar and Camacho-Collados, 2019] task for comparability with the accuracy scores reported in [Yuchen et al., 2021]. Here, the goal is to disambiguate, in a binary sense, whether a specific word appearing in two sentences conveys the same meaning, given their context. We closely follow the evaluation protocol explained in [Yuchen et al., 2021] when evaluating our models.

For all tasks, we conduct a 5-fold cross-validation analysis. Following [Yuchen et al., 2021], we use only one fold to determine the hyperparameters for testing on the remaining four folds. For the MTEB tasks, we optimize the hash length k. For WiC, we optimize the same hyperparameters as in [Yuchen et al., 2021].

4.4 Reinforcement Learning (RL) evaluation

In practice, learning policies for RL environments require a significantly large number of environment interactions and training iterations. We argue that `P-FlyVec` and `Comply` are computationally efficient, and thus, well-suited for this scenario. RL involves simulation environments phrased as Markov Decision Processes (MDP) that involve a very large number of learning iterations. For our evaluation, we use DDxGym [Winter et al., 2024], which is an RL environment focused on

---

[5]https://pubmed.ncbi.nlm.nih.gov/download/

differential diagnosis – the process of elimination that doctors employ to determine the diagnoses of patients. This environment generates sequences of text describing the state of a patient. Here, an agent learns a policy to choose *diagnosis* or *treatment* actions to cure a patient. The *diagnosis* actions uncover state information (e.g., symptoms) about the patient, while the *treatment* actions treat them. Since not all the information about the state (symptoms, severity, and overall health condition) is present in each observation, this is a partially observable MDP (POMDP). The environment is based on a knowledge graph of 111 diseases and an action space of 330 actions. We compare the *mean reward* and *mean episode length* of all models in this environment. Both metrics show the level of success to diagnose and treat a simulated patient given a textual representation of their state.

All evaluated models use the auxiliary language modeling objective described in [Winter et al., 2024]. In the case of `P-FlyVec` and `Comply`, this objective is based on the energy in eq. (1) and eq. (4), respectively. For the action and value function prediction, we use a projection layer on the hashes, ignoring gradient computation for $\mathbf{W}$. This means that $\mathbf{W}$ is only fine-tuned by the auxiliary language modeling objective. To compare with the results in [Winter et al., 2024], we use `IMPALA` [Espeholt et al., 2018] to optimize our policy for a maximum of 80M steps. This algorithm is a state-of-the-art actor-critic method to parallelize the optimization of a policy across multiple workers (GPUs).

4.5 Hyperparameters and implementation

When compared, we use the same number of parameters for `FlyVec` and `Comply` in all experiments. Namely, 400 Kenyon cells ($K = 400$) and their respective tokenizers: 20,000-word indices for OWC and 30,522 for PubMed, i.e., $N_{\text{voc}} = 20{,}000$ and $N_{\text{voc}} = 30{,}522$, respectively. We follow `FlyVec` to pretrain our models for 15 epochs with a learning rate of 4×10^{-4}, which is linearly annealed towards 0.

Our models are developed with `PyTorch` [Paszke et al., 2019]. For multiprocess parallelization and RL, we use `Ray` [Moritz et al., 2018] and `RLlib` [Liang et al., 2018]. For the WiC task, we conduct hyperparameter tuning using the `hyperopt` [Bergstra et al., 2013] scheduler. For training, we use `Adam` [Kingma and Ba, 2015] as optimizer. The sparse matrix multiplications in our model are mainly indexing operations, which we favor over the sparse[6] abstractions in `PyTorch`, due to a much lower memory footprint.

For pretraining, we use eight A100 GPUs and a batch size of 0.8 *million* samples. In contrast, the sentence similarity evaluation on `FlyVec` and `Comply` is run on 64 CPU cores to compute all hashes. For `BERT` we use one V100 GPU and a batch size of 8 in the evaluations. In the RL evaluation, for `P-FlyVec` and `Comply` we use four P100 GPUs and the same learning rate as in pretraining. For `Transformer-MLM` we report the results stated in [Winter et al., 2024].

5 Results and Discussion

In the following sections, we present the results of our experiments in both semantic similarity and Reinforcement Learning (RL) tasks.

5.1 Semantic similarity

Table 1 presents our results for the semantic similarity tasks.

Task scores. The scores show that on seven out of 13 datasets, `Comply` outperforms all other methods. Except for STS12, `Comply` consistently outperforms `FlyVec` by a large margin, signaling that for these tasks the added positional information is beneficial. `BERT` outperforms on five out of 13 tasks. However, in two of these, TwitterSemEval2015 and STSBenchmark, `Comply` is very close in second place. `FlyVec` outperforms in STS12, where `Comply` follows closely. Similarly, `ComplyM` outperforms `FlyVec` and `BERT` in six tasks, however falls short of `Comply` on all tasks except for STS13. Nevertheless, these results show that `ComplyM` is an additional viable method to compute sentence hashes.

---

[6]`https://pytorch.org/docs/stable/sparse.html`

Table 1: Five-fold cross-evaluation of Semantic Textual Similarity (STS) (top), Pair Classification (middle), and Words in Context (WiC) result for `BERT`, `FlyVec`, `Comply` and `ComplyM` with the best hash length k for each task. Except for STS12, our method outperforms `FlyVec` using the same number of parameters. `Comply` also outperforms `BERT` in seven out of the 13 tasks. In terms of evaluation time, `FlyVec`, `Comply` and `ComplyM` perform similarly and broadly outperform `BERT` even though the latter was evaluated on a GPU. The best score per task is highlighted in **bold**, and the best times are <u>underlined</u>. The results with an asterisk* are taken from [Yuchen et al., 2021].

	Score mean ± std (↑)				Evaluation time [s] mean ± std (↓)			
Task	BERT	FlyVec	Comply	ComplyM	BERT	FlyVec	Comply	ComplyM
BIOSSES	**44.4±4.64**	18.4±4.07	28.7±7.49	26.7±8.11	<u>0.28±0.01</u>	0.49±0.01	0.45±0.02	0.76±0.11
SICK-R	51.4±1.78	42.0±1.97	**53.9±1.59**	52.7±1.87	11.32±0.26	<u>3.43±0.11</u>	3.68±0.35	4.25±0.35
STS12	33.8±7.04	**46.3±7.49**	45.5±7.33	42.4±7.10	4.27±0.19	1.59±0.18	1.67±0.08	<u>1.59±0.16</u>
STS13	48.8±3.76	42.1±2.13	51.8±1.70	**52.7±1.99**	2.56±0.50	1.19±0.07	1.23±0.08	<u>1.13±0.09</u>
STS14	44.3±3.52	42.1±2.61	**53.6±2.25**	51.9±2.68	6.69±1.24	<u>1.72±0.16</u>	1.83±0.06	1.74±0.13
STS15	57.3±2.88	53.5±2.76	**60.4±1.24**	59.5±0.90	5.54±0.41	<u>1.57±0.10</u>	1.73±0.07	1.69±0.09
STS16	**60.2±6.21**	45.5±6.30	49.7±4.53	47.1±4.35	2.26±0.17	1.08±0.08	1.10±0.09	<u>1.04±0.04</u>
STS17	64.5±2.26	52.3±6.93	**67.7±1.45**	59.5±12.04	<u>0.23±0.01</u>	0.74±0.06	0.63±0.07	0.95±0.07
STSBenchmark	**45.4±3.19**	33.8±4.90	43.3±2.32	41.0±2.71	1.83±0.02	<u>1.14±0.07</u>	1.27±0.10	1.42±0.45
SprintDuplicateQuestions	31.6±0.90	40.4±0.70	**56.2±1.02**	55.0±1.09	11.98±0.18	<u>7.78±0.11</u>	8.22±0.10	8.07±0.35
TwitterSemEval2015	**51.3±1.57**	38.4±1.74	51.2±2.66	50.8±2.00	9.90±0.13	<u>2.75±0.08</u>	2.98±0.19	3.03±0.17
TwitterURLCorpus	71.1±1.08	66.0±0.93	**75.4±1.14**	74.6±1.23	84.89±5.15	<u>7.24±0.84</u>	8.00±0.54	7.64±0.23
WiC	**61.2±0.22***	57.7±0.27*	57.08±1.10	58.16±1.59	-	-	-	-

It is remarkable that `Comply` manages to bridge the performance gap and improve on `BERT` with only 14.56% of the parameters, and that this is achieved solely by incorporating positional information. Note that only unsupervised pretraining is involved and no post-hoc methods of model compression [Xu and McAuley, 2023] are employed. In practice, the latter is the most popular avenue to achieving significantly smaller models with comparable performance.

Task evaluation time. The evaluation times of `FlyVec` and `Comply` on CPUs are significantly smaller than those of `BERT` on GPU. BIOSSES and STS17 are the only tasks for which `BERT` shows a comparatively shorter time. We argue that this is solely due to the small number of samples in these tasks. This effect is confirmed with datasets like TwitterURLCorpus, where the number of samples is significantly larger. Here, the efficiency benefits of `FlyVec` and `Comply` variants become evident (only ≈7 compared to ≈85 seconds of evaluation time). We argue that the differences in efficiency observed in the evaluation time of `FlyVec`, `Comply`, and `ComplyM` are mainly due to tokenization, hardware optimization, and implementation details. Still, in terms of the hash computation both `Comply` and `ComplyM` are mathematically more complex as can be inferred from the times of larger tasks, such as TwitterURLCorpus.

This is reinforced when considering pretraining time. `Comply` takes approximately 15 minutes per epoch, which is significantly higher than the reported time for `FlyVec` on the OWC corpus – eight minutes on less powerful hardware ($8 \times$ A100 vs. $3 \times$ V100) [Yuchen et al., 2021]. This difference comes from our choice of implementation. We favor an automatic differentiation framework, while [Yuchen et al., 2021] computes analytically a learning rule to fit the parameters. Even in this setting, our training resources of ≈4 hours on $8\times$A100 are significantly inferior to the ≈96 hours of pretraining of `BERT` on 16 TPUs reported in [Kenton and Toutanova, 2019]. When we pretrain `P-FlyVec` for the RL evaluation with the PubMed corpus with our implementation (optimizing eq. (1)), we report that `P-FlyVec` and `Comply` take roughly 12 and 15 minutes per epoch, respectively, highlighting the added complexity of our energy proposed in eq. (4).

Hash length. Additionally, we survey the relationship between the mean test performance in the MTEB tasks and the chosen hash length k. We present this in fig. 4 comparing `FlyVec` and `Comply`. Notably, the scores of `Comply` are not only higher, but also more regular and concave, allowing for a hash length k that is optimal with sparsity in the neighborhood of 150 KCs. In contrast, `FlyVec` tends to have an optimal k either with very low or high values (very high and low sparsity). Minimizing the energy of `Comply` in eq. (4) can be interpreted as memorizing sequences. Hence, we argue that the relationship between the scores and k is due to these sequences being partially distributed across multiple neurons. Although the energy optimized in `FlyVec` could accomplish the same (due to the sliding window, and as confirmed with the tasks that perform better with very dense hashes), we argue that presenting the additional positional information explicitly in $\mathbb{C}$ creates more distinct hashes. We
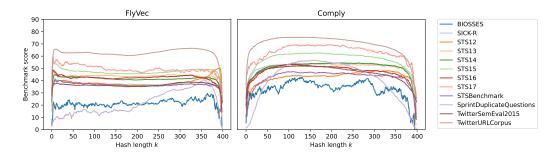
Figure 4: Test metric averaged over five folds with respect to the hash length k on the STS and PC tasks. `Comply` shows a better and more regular performance with respect to k, plateauing at around $k = 150$, while `FlyVec` tends to perform better on either sparse (low k) or dense (high k) hashes for these tasks.
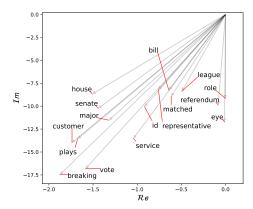

Figure 5: Highest activated neuron on the input sentence "*Senate majority leader discussed the issue with the members of the committee*" as in [Yuchen et al., 2021]. The neuron shows signs of memorizing (overlapping) of short sequences like [house, representative], [plays, major, role], and [senate, bill], among others. These are meaningful to the context of the input and likely to appear in the pretraining corpus.

observe these partially distributed sequences in `Comply` and present a qualitative example in fig. 5 of the most activated neuron conditioned on a specific sentence taken from [Yuchen et al., 2021]. This neuron shows contextually relevant short sequences that can be extracted due to the relative position preserved in the phases of the words. Examples of such sequences can be found in the caption.

5.2   Reinforcement Learning (RL)

In fig. 6, we present the results of the evaluation on DDxGym [Winter et al., 2024]. Both mean episode reward and mean episode length estimate how well the models learn to solve this environment. For the Transformer baseline both metrics are taken from [Winter et al., 2024]. `Comply` outperforms the Transformer baseline roughly after 15M environment steps, and it continues to improve until the 80M mark. The Transformer converges to a mean reward of 400 after roughly 10M steps. Since the theoretical maximum for the reward in DDxGym is at about 1200, there is significant room for improvement. The mean episode length shows a similar behavior – here a lower value means that the episode ends successfully with the diagnosis both uncovered and treated. While the Transformer converges at around 18 steps per episode, `Comply` improves until the experiment stops at 80M episodes with around 16 steps per episode. Although `P-FlyVec` initially reaches a reward of almost 0, it fails to improve further and shortly collapses in performance. This further confirms that capturing positional information significantly improves the performance of `Comply`, and makes the method more competitive to larger sequence-to-sequence models like Transformers.
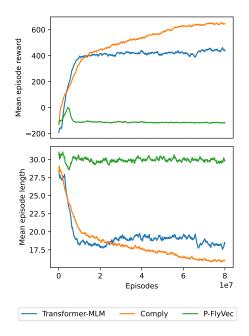
Figure 6: Mean episode reward (top, higher is better) and mean episode length (bottom, lower is better) in DDxGym. `Comply` outperforms and continues to improve, while the Transformer baseline converges/peaks at around 15M episodes.

6 Conclusions

We present `Comply`, an approach to improve learning of sequences by introducing positional information to a biologically-motivated model of the mushroom body of a fruit fly. We achieve this by a novel loss that leverages representations in the complex field $\mathbb{C}$ and generalizes `FlyVec` [Yuchen et al., 2021] to learning sentences. Our experiments show that `Comply` improves sequence representations without adding more parameters. In sentence similarity tasks we significantly bridge the gap between these biologically-inspired models and larger Transformer encoders like `BERT`. We show that the resource efficiency of our method makes it suitable for challenging settings like Reinforcement Learning (RL), where it outperforms a Transformer baseline. Our analysis expands on the mechanism of the loss we introduce and the interpretability of the learned feature representations.

7 Limitations and future work

Tasks and modalities. We limit our experiments to pretraining our model on large general-purpose corpora. Including the objective of sequence similarity as in sentence Transformers [Reimers and Gurevych, 2019] would expand the comparison to such models. Although our work presents results on the textual modality for comparability with previous work, we believe that our method is applicable to different modalities. Thus a more extensive evaluation of, e.g., different RL environments or time-series tasks is needed.

Loss and model design. Word repetition is supported in our model in a limited way. Namely, the phases of repeated words are averaged by the aggregation (see eq. (4)) during training. Improvements in our method could stem from incorporating word repetition as a factor when learning sub-sequences across neurons as we show qualitatively in fig. 5. Furthermore, for the sake of comparability, we do not extend our evaluation to larger models (larger K). Scaling induces problems such as *dead neurons*, so strategies like the GABA switch as proposed by [Bricken et al., 2023] or similar extensions could be needed. Thus, future work would involve a study on the effect of scale on our approach. Finally, the inherent benefit of adding positional information into a learned representation yields the potential for sequence generation. We do not explore this in our work, but we see making the model generative as a future direction of research.

11

Acknowledgments and Disclosure of Funding

References

[Andersson et al., 2011] Andersson, M. N., Binyameen, M., Sadek, M. M., and Schlyter, F. (2011). Attraction modulated by spacing of pheromone components and anti-attractants in a bark beetle and a moth. *Journal of Chemical Ecology*, 37:899–911. 2

[Baker et al., 1998] Baker, T., Fadamiro, H., and Cosse, A. (1998). Moth uses fine tuning for odour resolution. *Nature*, 393(6685):530–530. 2

[Bergstra et al., 2013] Bergstra, J., Yamins, D., and Cox, D. D. (2013). Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *SciPy*, pages 13–19. scipy.org. 8

[Bricken et al., 2023] Bricken, T., Davies, X., Singh, D., Krotov, D., and Kreiman, G. (2023). Sparse distributed memory is a continual learner. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. 11

[DasGupta and Waddell, 2008] DasGupta, S. and Waddell, S. (2008). Learned odor discrimination in drosophila without combinatorial odor maps in the antennal lobe. *Current Biology*, 18(21):1668–1674. 2

[de Belle and Heisenberg, 1994] de Belle, J. S. and Heisenberg, M. (1994). Associative odor learning in <i>drosophila</i> abolished by chemical ablation of mushroom bodies. *Science*, 263(5147):692–695. 2

[Espeholt et al., 2018] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR. 8

[Gers et al., 2000] Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471. 3

[Gokaslan and Cohen, 2019] Gokaslan, A. and Cohen, V. (2019). Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`. 7

[Gu et al., 2021] Gu, Y., Tinn, R., Cheng, H., Lucas, M., Usuyama, N., Liu, X., Naumann, T., Gao, J., and Poon, H. (2021). Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare (HEALTH)*, 3(1):1–23. 7

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780. 3

[Honegger et al., 2011] Honegger, K. S., Campbell, R. A., and Turner, G. C. (2011). Cellular-resolution population imaging reveals robust sparse coding in the drosophila mushroom body. *Journal of neuroscience*, 31(33):11772–11785. 2

[Hopfield, 1991] Hopfield, J. (1991). Olfactory computation and object perception. *Proceedings of the National Academy of Sciences*, 88(15):6462–6466. 2

[Islam et al., 2020] Islam, M. A., Jia, S., and Bruce, N. D. B. (2020). How much position information do convolutional neural networks encode? In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. 3

[Kenton and Toutanova, 2019] Kenton, J. D. M.-W. C. and Toutanova, L. K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, page 2. Minneapolis, Minnesota. 1, 3, 9

[Kingma and Ba, 2015] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR (Poster)*. 8

[Krotov and Hopfield, 2019] Krotov, D. and Hopfield, J. J. (2019). Unsupervised learning by competing hidden units. *Proceedings of the National Academy of Sciences*, 116(16):7723–7731. 3

[Laurent, 2002] Laurent, G. (2002). Olfactory network dynamics and the coding of multidimensional signals. *Nature reviews neuroscience*, 3(11):884–895. 2

[Lee et al., 2022] Lee, C., Hasegawa, H., and Gao, S. (2022). Complex-valued neural networks: A comprehensive survey. *IEEE/CAA Journal of Automatica Sinica*, 9(8):1406–1426. 3

[Liang et al., 2018] Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M., and Stoica, I. (2018). Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062. PMLR. 8

[Lin et al., 2014] Lin, A. C., Bygrave, A. M., De Calignon, A., Lee, T., and Miesenböck, G. (2014). Sparse, decorrelated odor coding in the mushroom body enhances learned odor discrimination. *Nature neuroscience*, 17(4):559–568. 2

[Lin et al., 2017] Lin, Z., Feng, M., dos Santos, C. N., Yu, M., Xiang, B., Zhou, B., and Bengio, Y. (2017). A structured self-attentive sentence embedding. *CoRR*, abs/1703.03130. 3

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. 1, 3

[Moritz et al., 2018] Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., et al. (2018). Ray: A distributed framework for emerging {AI} applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, pages 561–577. 8

[Muennighoff et al., 2023] Muennighoff, N., Tazi, N., Magne, L., and Reimers, N. (2023). Mteb: Massive text embedding benchmark. 7

[Nikonov and Leal, 2002] Nikonov, A. A. and Leal, W. S. (2002). Peripheral coding of sex pheromone and a behavioral antagonist in the japanese beetle, popillia japonica. *Journal of chemical ecology*, 28:1075–1089. 2

[Olsen et al., 2007] Olsen, S. R., Bhandawat, V., and Wilson, R. I. (2007). Excitatory interactions between olfactory processing channels in the drosophila antennal lobe. *Neuron*, 54(1):89–103. 2

[Papadopoulou et al., 2011] Papadopoulou, M., Cassenaer, S., Nowotny, T., and Laurent, G. (2011). Normalization for sparse encoding of odors by a wide-field interneuron. *Science*, 332(6030):721–725. 2

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32. 8

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543. 1, 3

[Pilehvar and Camacho-Collados, 2019] Pilehvar, M. T. and Camacho-Collados, J. (2019). Wic: the word-in-context dataset for evaluating context-sensitive meaning representations. 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 1267–1273. 7

[Raman et al., 2010] Raman, B., Joseph, J., Tang, J., and Stopfer, M. (2010). Temporally diverse firing patterns in olfactory receptor neurons underlie spatiotemporal neural codes for odors. *Journal of Neuroscience*, 30(6):1994–2006. 2

[Reimers et al., 2016] Reimers, N., Beyer, P., and Gurevych, I. (2016). Task-oriented intrinsic evaluation of semantic textual similarity. In Matsumoto, Y. and Prasad, R., editors, *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 87–96, Osaka, Japan. The COLING 2016 Organizing Committee. 7

[Reimers and Gurevych, 2019] Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In Inui, K., Jiang, J., Ng, V., and Wan, X., editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3980–3990. Association for Computational Linguistics. 11

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986. *Biometrika*, 71(599-607):6. 3

[Ryali et al., 2020] Ryali, C., Hopfield, J., Grinberg, L., and Krotov, D. (2020). Bio-inspired hashing for unsupervised similarity search. In *International conference on machine learning*, pages 8295–8306. PMLR. 6

[Saha et al., 2013] Saha, D., Leong, K., Li, C., Peterson, S., Siegel, G., and Raman, B. (2013). A spatiotemporal coding mechanism for background-invariant odor recognition. *Nature neuroscience*, 16(12):1830–1839. 2

[Sehdev et al., 2019] Sehdev, A., Mohammed, Y. G., Triphan, T., and Szyszka, P. (2019). Olfactory object recognition based on fine-scale stimulus timing in drosophila. *iScience*, 13:113–124. 2

[Sinha et al., 2021] Sinha, K., Jia, R., Hupkes, D., Pineau, J., Williams, A., and Kiela, D. (2021). Masked language modeling and the distributional hypothesis: Order word matters pre-training for little. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t., editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2888–2913, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics. 3

[Stopfer et al., 2003] Stopfer, M., Jayaraman, V., and Laurent, G. (2003). Intensity versus identity coding in an olfactory system. *Neuron*, 39(6):991–1004. 2

[Su et al., 2024] Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., and Liu, Y. (2024). Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063. 3

[Szyszka et al., 2012] Szyszka, P., Stierle, J. S., Biergans, S., and Galizia, C. G. (2012). The speed of smell: Odor-object segregation within milliseconds. *PLOS ONE*, 7(4):1–4. 2

[Turner et al., 2008] Turner, G. C., Bazhenov, M., and Laurent, G. (2008). Olfactory representations by drosophila mushroom body neurons. *Journal of neurophysiology*, 99(2):734–746. 2

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc. 3

[Wang et al., 2020] Wang, B., Zhao, D., Lioma, C., Li, Q., Zhang, P., and Simonsen, J. G. (2020). Encoding word order in complex embeddings. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. 3

[Winter et al., 2024] Winter, B., Figueroa Rosero, A. G., Loeser, A., Gers, F. A., Figueroa Rosero, N. K., and Krestel, R. (2024). DDxGym: Online Transformer Policies in a Knowledge Graph Based Natural Language Environment. In Calzolari, N., Kan, M.-Y., Hoste, V., Lenci, A., Sakti, S., and Xue, N., editors, *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 4438–4448. ELRA and ICCL. 6, 7, 8, 10

[Witzgall and Priesner, 1991] Witzgall, P. and Priesner, E. (1991). Wind-tunnel study on attraction inhibitor in male coleophora laricella hbn.(lepidoptera: Coleophoridae). *Journal of chemical ecology*, 17:1355–1362. 2

[Xu and McAuley, 2023] Xu, C. and McAuley, J. (2023). A survey on model compression and acceleration for pretrained language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 10566–10575. 9

[Yuchen et al., 2021] Yuchen, L., Ryali, C., Hoover, B., et al. (2021). Can a fruit fly learn word embeddings? In *Proceedings of the 9th International Conference on Learning Representations, Austria*. 1, 2, 3, 4, 6, 7, 9, 10, 11