Elementwise Language Representation

Anonymous authors
Paper under double-blind review

Abstract

We propose a new technique for computational language representation called elementwise embedding, in which a material (semantic unit) is abstracted into a horizontal concatenation of lower-dimensional element (character) embeddings. While elements are always characters, materials are arbitrary levels of semantic units so it generalizes to any type of tokenization. To focus only on the important letters, the nth spellings of each semantic unit are aligned in n^{th} attention heads, then concatenated back into original forms creating unique embedding representations; they are jointly projected thereby determining own contextual importance. Technically, this framework is achieved by passing a sequence of materials, each consists of v elements, to a transformer having $h = v$ attention heads. As a pure embedding technique, elementwise embedding replaces the w-dimensional embedding table of a transformer model with 256 c-dimensional elements (each corresponding to one of UTF-8 bytes) where $c = w/v$. Using this novel approach, we show that the standard transformer architecture can be reused for all levels of language representations and be able to process much longer sequences at the same time-complexity without "any" architectural modification and additional overhead. BERT trained with elementwise embedding outperforms its subword equivalence (original implementation) in multilabel patent document classification exhibiting superior robustness to domain-specificity and data imbalance, despite using 0.005% of embedding parameters. Experiments demonstrate the generalizability of the proposed method by successfully transferring these enhancements to differently architected transformers CANINE and ALBERT.

1 Introduction

We understand texts from various levels of semantics but current language representation strategies leverage tokenization which relies on a certain level of semantics exclusively, fully ignoring the hierarchical structures of natural languages. Text is encoded to a sequence of integers then projected into fixed-size latent embeddings. These types of expressions result in a recursive trade-off between different levels of language representations: (sub)word-level models indirectly recover characters (Itzhak & Levy, 2021) but it is not always sufficient for spelling-sensitive tasks, character-level models need much longer sequences to reach comparable performance to word-level models thus amplifying the computational complexity of self-attention. Some recently proposed studies (Clark et al., 2022; Godey et al., 2022; Tay et al., 2021) attempt to solve this by downsampling long character sequences into an acceptable length, however, they share the same limitation as pure character-level models because their valid downsampling rates are constrained to relatively small values mainly due to the smoothing and overhead issues.

Instead, we propose elementwise embedding, a language representation technique for addressing this trade-off in which a set of lower-dimensional character embeddings called *elements* are horizontally concatenated into a single latent embedding called *material* that mimics a semantic unit such as a word, phrase, sentence and etc. Using this method, models with higher-dimensional hidden representations create each semantic unit (i.e., a material) by concatenating a greater numbers of characters (i.e., elements), which implies that larger models can process longer sequences than smaller ones at the same computational complexity. This means that the acceptable sequence length scales with the size of a transformer model, but the complexity is fixed as that of its attention. Assuming that a character-level GPT-3 [processing 2048 12,288-dimensional token embeddings with 96 attention heads; Brown et al. (2020)] is trained with elementwise embedding, it aligns a sequence of $2,048 \times 96 = 296,608$ characters which is 96x longer at the same $O(N\sqrt{N})_{N=2048}$ complexity.

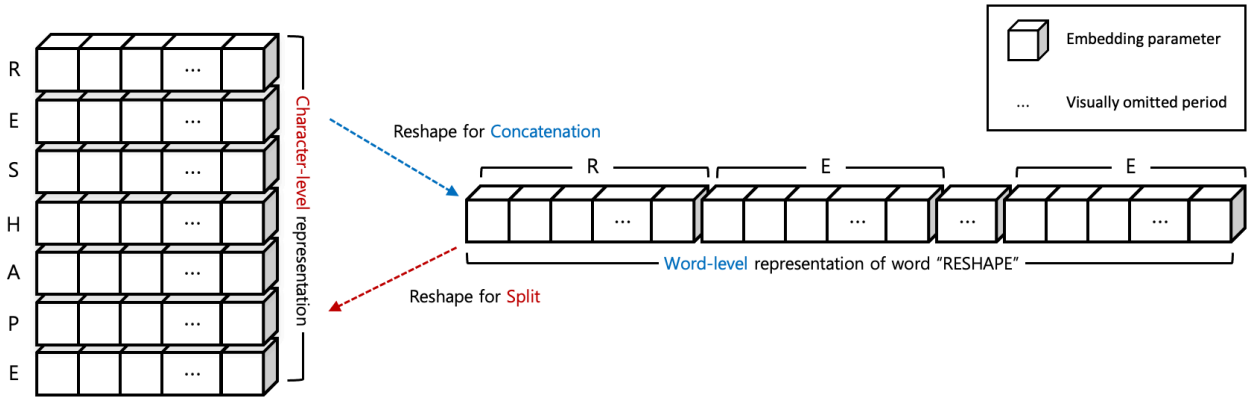

Figure 1: Visualization of the proposed elementwise language representation. A material (word "RESHAPE" here) is abstracted into a horizontal concatenation of elements (spellings [R, E, S, H, A, P, E] here).

The proposed methodology follows the two-step framework of *"reshape, then focus"*. First, the given text is encoded as a sequence of uv UTF-8 bytes and projected into a (uv, c) embedding matrix in which each row is a c-dimensional element; it's *"reshaped"* into a (u, w) embedding matrix in which each row is a w-dimensional material (e.g., a word), where $c = w/v$. As a result, one material consists of v elements so that we can align uv elements at the $O(u^2)$ complexity using multihead self-attention (Vaswani et al., 2017) with v attention heads. Each ith column of this (u, w) material matrix is a sequence of the ith elements of all u materials, so i^{th} attention head aligns i^{th} elements. This operation is most straightforward when a material is a v letters word: i^{th} spellings of all u words are aligned in i^{th} attention head, then concatenated back creating unique embedding representations where $i \in [1, v]$. Each attended i^{th} spelling is referred as *focus* because it is quite similar to that we often read text inferring the meanings of words by *"focusing"* on a few important letters. The contextual importance of each word is determined jointly via linear transformation. Theoretically, this can be understood as lowering the entropy of character sequences concentrating distributed probabilities into several important spellings. Technically, it is just to pass a (u, w) word embedding matrix in which each row is a horizontal concatenation of v c-dimensional character embeddings as input to a transformer model with w-dimensional hidden layers. It's identical to aligning words using character-level semantics and vice versa.

In practical implementation, focus is performed by multihead attention of the parent (any transformer model) by setting the number of attention heads to $h = v$, so applying elementwise embedding is simply to replace the embedding table of parent model with a set of 256 c-dimensional character embeddings (each mapping to one of UTF-8 bytes; elements) and a following tensor reshaping operation. Neither structural modification of neural networks nor additional operations such as up/downsampling that entail unnecessary engineering efforts and overheads are required. Fig 1 offers an intuitive visualization of elementwise embedding.

2 Research Objectives

In this study, we suggest the new elementwise language representation and demonstrate its validity.

Theoretically, we propose the first generalized language representation:

- applying with all levels of tokenization strategies
- aligning longer sequences proportional to the model's size
- based on information theory rather than the distributional hypothesis

Empirically, we demonstrate the practical contributions of the proposed methodology by:

- reusing BERT (Devlin et al., 2018) for various levels of language representation
- improving BERT to be more robust to domain-specific and imbalanced training examples
- improving BERT to process longer sequences at the same $O(N^2)$ computational complexity

without any architectural modification and additional overhead.

We generalize these contributions to different transformers CANINE (Clark et al., 2022) and ALBERT (Lan et al., 2019). Through experiments, we validate the clear superiority in robustness to domain-specificity and dataset imbalance of the proposed method by comparing transformers, trained with elementwise embedding from scratch without pretraining, with their original implementations for multilabel patent classification.

This is the first part of our two-paper study discussing:

1. Theoretical and practical advantages of elementwise language representation
2. Unsupervised language modeling strategy for elementwise language representation

Though these were originally topics to be covered in one paper, we divided them into two parts mainly due to the lack of computational resources for pretraining and demonstrating language models in various scales at the time of this study.

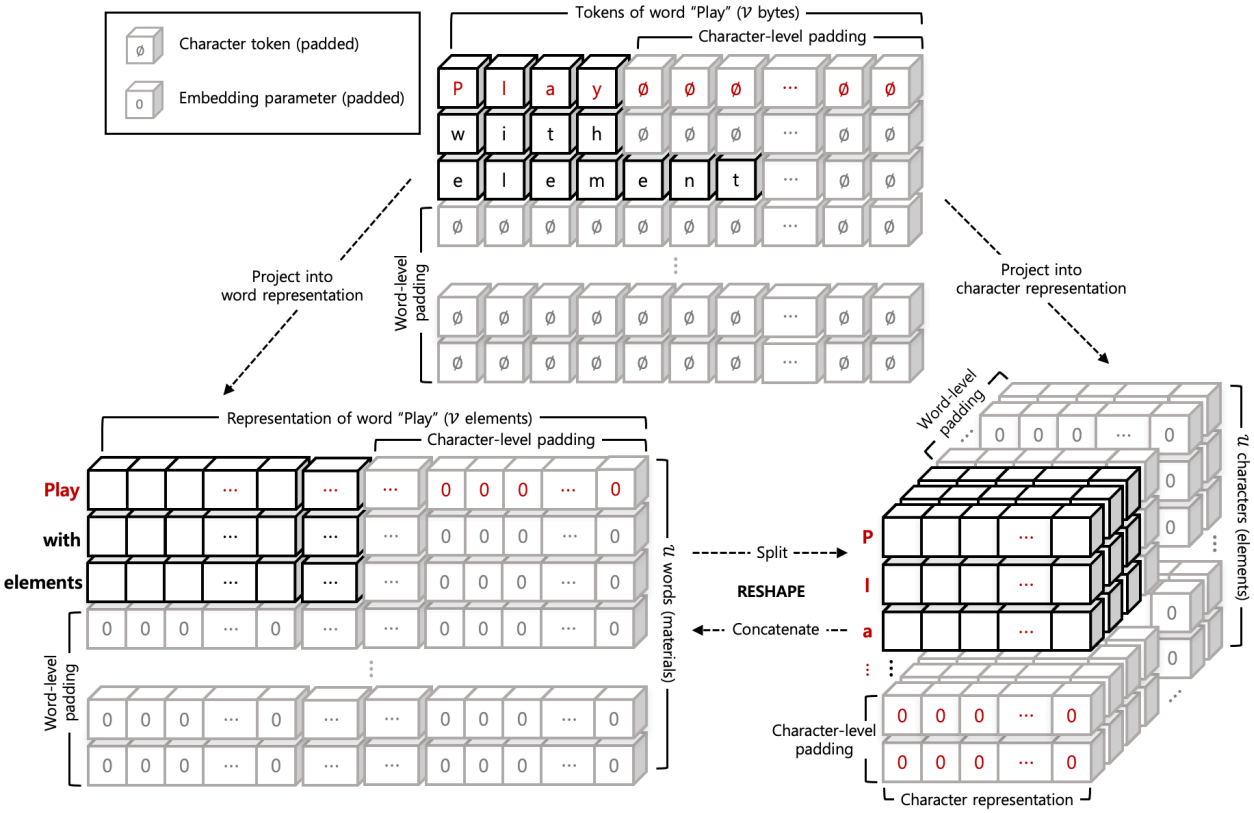

Figure 2: Overall framework of elementwise language representation applied with whitespace tokenization. First, the given text is tokenized into a sequence of u words based on whitespaces. Each word is encoded to a sequence of v UTF-8 bytes resulting in a uv bytes sequence; each byte is 4 greater than its original value for reserving 4 special tokens [CLS], [SEP], [PAD] and [MASK]; the token [MASK] is reserved for unsupervised pretraining with elementwise embedding to be introduced in our follow-up study. Words shorter than v are padded with integer zeros, longer ones are truncated. Sequences shorter than u are padded with embeddings filled with v zeros. uv bytes are projected into a (uv, c) character embedding matrix, and then reshaped into a (u, w) word embedding matrix in which each row comprises of horizontally concatenated v c-dimensional character embeddings, where $c = w/v$. Transformation from (sub)word-level to character-level representation and vice versa are always possible via reshaping operation. This framework can be generalized to all kinds of tokenization; just split text into u tokens, encode each token to v bytes, project, then reshape.

3 Related Work

3.1 Character-level Models

Most of the past and current state-of-the-arts and impactful studies in the field of natural language processing (Devlin et al., 2018; Radford et al., 2019; Lan et al., 2019; Yang et al., 2019; Brown et al., 2020; Clark et al., 2020; Raffel et al., 2020; Reed et al., 2022; Taylor et al., 2022) relies on subword-level tokenization (Sennrich et al., 2015; Wu et al., 2016; Kudo & Richardson, 2018). These pervasive choices are due to the reasonable trade-off between robustness and efficiency of subword tokenization, but their limitations in several special environments wherein the data is domain-specific and/or its distribution shifts frequently are problems that have to be addressed at some point. While some subsequent studies have suggested improved techniques for subword-level tokenization (Provilkov et al., 2019; He et al., 2020; Hiraoka et al., 2021; Wang et al., 2021), many of them involve significant increases in computational costs and engineering efforts.

Character-level modeling has long been proposed as a promising alternative to the (sub)word-level language representations. Although the chronic long-range dependence issues of pure character-level models (Sutskever et al., 2011; Graves, 2013; Zhang et al., 2015) solved by the adoption of the transformer architecture (Vaswani et al., 2017) as demonstrated in (Belouadi & Eger, 2022; Xue et al., 2022), the quadratic time-complexity of self-attention became a new bottleneck for character-level language representation. Some recently proposed studies try to address this problem by downsampling the long character sequences to an acceptable length: Tay et al. (2021) and Clark et al. (2022) utilize convolutional and non-parametric mean-pooling respectively, Godey et al. (2022) leverages non-parametric max-pooling. All of them requires additional computations for explicit downsampling and character-level enrichment, thus causing corresponding overheads. The proposed elementwise embedding achieves the similar using a trivial tensor reshaping operation so does not degrades the inference speed of its backbone transformer architecture while processing much longer sequences.

3.2 Efficient Transformers

The major challenge of the transformer architecture Vaswani et al. (2017) is to mitigate its quadratic self-attention complexity. Liu et al. (2018) proposed to enhance this complexity by computing attentions within partitioned embedding matrices. Child et al. (2019) estimated the full attention by mixing a sparse number of local attentions. Beltagy et al. (2020) extended this idea using technique called dilated sliding window to cover a wider range of attention. Zaheer et al. (2020) achieved the similar by attending three different types of attentions: global, windowed and random. Kitaev et al. (2020) alleviated the memory complexity by applying locality sensitive hashing and reversible residual layers Gomez et al. (2017). Wang et al. (2020) improved overall attention complexity to be linear time by performing dimensionality reduction on the length axis, Katharopoulos et al. (2020) enhanced the computational complexity to linear time using kernel-based formulation and causal masking.

Approaches through other structural modifications have also been proposed. Lan et al. (2019) reduced the size of its BERT (Devlin et al., 2018) backbone extremely smaller by sharing weight parameters between every attention layer. The concept of knowledge distillation (Hinton et al., 2015) has been demonstrated to be useful by (Jiao et al., 2019; Sanh et al., 2019; Tang et al., 2019). Trials for improving the inference-time efficiency in which unimportant attention heads are pruned away (Michel et al., 2019; Voita et al., 2019); or "blocks" are pruned instead (Lagunas et al., 2021); were made. Some recent studies proposed to enhance computational complexity by downsampling input sequences to an acceptable length (Godey et al., 2022; Tay et al., 2021). Elementwise embedding is quite similar to these approaches in terms of increasing the efficiency of transformer architecture, but is fundamentally different in that it does not require any downsampling and architectural modification of the transformer model to work with. What it does is that simply projects uv bytes sequence into a (uv, c) character embedding matrix, reshapes it into a $(u, w = vc)$ embedding matrix, and pass it to a transformer as input features; thus making it to align uv sequences at the $O(u^2)$ complexity. As elementwise embedding is designed as a pure embedding technique that does not modify any part of the transformer architecture, it can be potentially utilized with all the above methods in conjunction.

3.3 Patent Classification

Patent classification is an interesting subfield of text classification that targets to automize the categorization of patent documents. Although this research field has not been actively studied compared to other compelling applied-ml areas like medicine and robotics, it deserves attention because it is essential for the data-driven patent analysis (Lee et al., 2009; Kim & Lee, 2017). Patents filed during a specific period are often closely associated with technological trends at that time which implies a big flow of capital, and modern machine learning methods as large language models (Brown et al., 2020; Taylor et al., 2022) and graph-powered neural networks (Sanchez-Gonzalez et al., 2020; Stokes et al., 2020; Jumper et al., 2021) are powerful enough for extracting meaningful patterns from those textual/tabular data to lead high-impact decision makings.

In addition to the purpose of patent analysis, patents are also valuable for general-purpose machine learning research. Patent documents are large amounts of multi-modal data that consist of texts, graphs, images and their organized structure makes it easier to preprocess than dealing with raw data such as randomly crawled web corpora. Furthermore, patent data can be used to benchmark learning algorithms because its extremely imbalanced distribution and a wide range of domain-specific lexicons hinder the models from convergence. This is the main reason that we utilize patent classification for evaluating improved robustness of our models trained with the proposed elementwise embedding.

The technical requirement of patent classification is largely threefold. First, the classifier should be possible to capture the unique properties (i.e., the classification symbols) of each patent that is computed by Precision and must be able to distinguish between different patent documents (i.e., the difference between classification symbols of two separate patent documents) that is calculated by Recall. Second, the classifier has to do well on both Precision and Recall, and every classification symbol should have equal importance (all technical categories are potentially important even if it is not currently popular). The former can be achieved by F_1 measure which is the harmonic mean of Precision and Recall, and the latter is satisfied by computing the micro-averaged scores for these three metrics. Third, the first-listed classification symbol must best indicate the invention of each patent and later ones offer additional information regardless of their relative positions (i.e., the meaning of a symbol differs by its order listed). To the best of our knowledge, however, this guide has not been reflected in known previous studies on multilabel patent classification (Lim & Kwon, 2017; Li et al., 2018; Yadrintsev et al., 2018; Lee & Hsiang, 2020; Haghighian Roudsari et al., 2022). They considered two different patents having classification symbols [G06Q, G06Q, A01B], [A01B, G06Q, A01B] as identical, one-hot encoding their labels as [A01B: 1, G06Q: 1]. This skewed labeling no longer provide classifiers with the correct evaluation criteria. To fix this problem, we performed simple relabeling that conserves the order of class labels (see Section 5.3) and compared the experimental results with our own baselines. Section 5.5 offers the equations of the above three metrics used in the following experiments.

4 Methodology

Before explaining the detailed implementation of elementwise embedding, we define mathematical notations to be used in this section. We denote the sequence of u semantic units (i.e., the given text), as an embedding matrix $e_u \in \mathbb{R}^{u \times w}$ and its ith row (e.g., the ith word in a sentence) by $e^{(i)}$. The jth character in each ith semantic unit (e.g., the jth letter of ith word) is denoted by $e^{(i)[j]} \in \mathbb{R}^{1 \times c}$, where $c = w/v$. Focus embeddings $f^{(i)} \in \mathbb{R}^{1 \times w}$ (local) and $g^{(p)} \in \mathbb{R}^{1 \times c}$ (global) are added to $e^{(i)}$ and $e^{(i)[j]}$ respectively, by elementwise addition $\oplus$, where $p = (i \times j) + j$. Operator $\mathrm{Reshape}_{(a \times b)}$ reshapes the given tensor to be of the shape $(a \times b)$.

4.1 Elementwise Embedding

This section describes the detailed implementation of elementwise embedding, the technique for elementwise language representation. Consider a word with a missing spelling *App_le*. Missing spelling has low entropy since we can easily infer that it will be "*l*". In the case of a sentence with a missing word "*_ brought a basket of apples to the front yard.*", the entropy of missing word becomes higher since its spellings vary depending on what word the subject becomes. For the same reason, the entropy of a missing sentence in a paragraph will skyrocket. Based on this intuition, we can assume that the entropy is lowest at the character-level and grows in higher semantic levels; the entropy of a semantic unit is proportional to the number of its spellings.

Assumption. *For a v letters semantic unit x,* $\mathrm{H(x)} \propto v$

where H is the *Shannon entropy* $\mathrm{H(x)} = -\mathbb{E}_{\mathrm{x}\sim p}[\log P(x)]$ and $v \in \mathbb{R}$. Because low entropy means there are fewer cases to encode, it is natural to represent a character as a much lower-dimensional latent embedding. Assuming a character as one of UTF-8 bytes [1] and each semantic unit (i.e., a token) consists of v characters, a neural network with w-dimensional hidden layers will have 256 c-dimensional character embeddings as its embedding table when $c = w/v$. A semantic unit is abstracted into a horizontal concatenation of v character embeddings. Larger meanings are just concatenations of smaller ones and this hierarchical expression allows neural networks to explicitly recognize characters while learning at any complex-level of semantics. We name this pair of $(256, c)$ embedding table and the following concatenation operation as elementwise embedding, referring 256 character embeddings to *elements* and their concatenated meanings to *materials*.

As entropy of any v letters semantic unit increases proportionally with v, we need a way to reduce it again. One reasonable approach is to concentrate the probabilities of v spellings to several important letters. Using self-attention (Vaswani et al., 2017) with v attention heads, we can align $\{e^{(i)[n]}\}_{i=1}^{u}$, the sequence of the nth letters of u semantic units, thereby assigning higher probabilities to more important characters. It is similar to that we often catch the meanings of words by focusing only some morphologically noticeable spellings, so we call this operation *focus*. By setting the number of attention heads to $h = v$, n^{th} attention focuses on to n^{th} important letters when $n \in [1, v]$. For example, when the input sequence is encoded to a sequence [*Focus*, *on*, *the*, *elements*], 1st attention head attend to 1st spellings [*F*, *o*, *t*, *e*] focusing on the most important letters e.g., [*F*, *e*], 2nd head attends to [*o*, *n*, *h*, *l*], and so on (see Fig 5 in Appendix A).

Proposition. *The entropy of an important semantic unit $e^{(i)}$ in the given text e_u can be minimized using v-headed self-attention, where $e^{(i)}$ consists of horizontally concatenated v c-dimensional character embeddings.*

Proof. In forward propagation, each nth letter in i^{th} semantic unit $e^{(i)[n]}$ is assigned a probability by softmax function in n^{th} attention head, then concatenated back ($n \in [1, v], i \in [1, u]$). All uv characters are jointly projected by a position-wise feed-forward layer. This allows neural networks to jointly attend once to spelling of each $e^{(i)}$ and once again to the entire uv characters in e_u. Probability of each $e^{(i)}$ is determined by the alignment of its spellings $\{e^{(i)[n]}\}_{n=1}^{v}$. Cost is computed by an arbitrary objective function and errors are backpropagated to each $e^{(i)[n]}$. Networks and elementwise embedding are updated to assign higher probability to more crucial $e^{(i)}$, based on its letters $\{e^{(i)[n]}\}_{n=1}^{v}$, so that the given objective is minimized. □

Note that by matching the number of attention heads with v, we can restrict the subspace of each attention head to the latent space of c-dimensional character embeddings (i.e., elements), when attention layers expect $vc = w$-dimensional embeddings (i.e., materials) as input features. It can be interpreted as approximating the w-dimensional latent space with a closed set of $w/v = c$-dimensional vectors, and also as dividing the roles of embeddings and hidden layers: embeddings to encode character-level semantics and hidden layers to encode more complex-levels of semantics. This helps neural networks do not waste their limited expressiveness on encoding implicit information. Because the transformer architecture (Vaswani et al., 2017) is itself multiple layers of attention, we do not have to implement $focus$ operation by hand, so what we need for elementwise language representation is only a single-time tensor reshaping operation from (uv, c) to (u, w); which is the same as concatenating uv c-dimensional elements to be u w-dimensional materials. Following this framework, the standard transformer architecture can align uv characters at the $O(u^2)$ computational complexity fully ignoring the value of v. The acceptable length of the input sequence scales with the size of the hidden layers so larger models can process longer sequences than the smaller ones at the same attention complexity; this is more natural than the current transformers that no matter how large the model is, the length of the sequence it can process in a reasonable amount of time does not change at all.

4.2 Implementation Details

In practical implementation, focus for aligning important elements is performed by the mutli-head attention of parent architecture (i.e., a transformer model to work with) having $h = v$ attention heads, so elementwise embedding is just a set of lookup table containing 256 c-dimensional element embeddings and the following

[1] While only ASCII characters can be expressed in 1 byte, we used this analogy for ease of explanation.

tensor reshaping operation. In other words, applying elementwise embedding is simply to replace the existing embedding table of parent model with elementwise embedding. While token embeddings are usually trained with neural networks, they can be always detached fully independently of the network architecture so that we do not regard replacing embedding table as a structural modification. This feature is clearly different from the previous studies on character-aware language representation that additional computational components (e.g., non-parametric mean/min/max pooling, shallow convolutional/recurrent/transformer layers) are required for enriching character-level information or up/downsampling input sequences as leveraged in (Ma et al., 2020; Tay et al., 2021; Clark et al., 2022; Godey et al., 2022).

Technically, elementwise embedding is implemented by a single-time reshaping operation as,

$$e^{(i)} \leftarrow \text{Reshape}_{(1,w)}[\{e^{(i)[j]} \oplus g^{(p)}\}_{j=1}^{v}] \oplus f^{(i)}$$

We add two kinds of position embeddings $g^{(p)}$ and $f^{(i)}$ called *focus embeddings* to $e^{(i)[j]}$ and $e^{(i)}$ respectively, to manually encode the focusable positions: the former describes global positions (e.g., position of a character $e^{(i)[j]}$ in the entire sentence e_u) and the latter directs the local positions (e.g., position of a spelling $e^{(i)[j]}$ in the ith word $e^{(i)}$). Though elementwise embedding works well without focus embeddings, we found that they help models trained with elementwise embedding a lot to converge more stable and to perform better as explained in Section 6.1. Before being passed to as input to the parent model, dropout (Srivastava et al., 2014) and normalization (Ba et al., 2016) can be applied to embeddings for better generalization performance.

Elementwise embedding applies with any type of tokenization by following the framework:

1. Divide given text into u tokens
2. Encode each token into v integers (UTF-8 bytes)
3. Project integers into a (uv, c) element embedding matrix
4. Reshape element embedding matrix into a (u, w) material embedding matrix

where $w = vc$ (see Fig 2).

5 Experimental Setup

5.1 Dataset

All models used in the following experiments were trained on patent documents published by the USPTO (United States Patent and Trademark Office) from 2006 to 2014 and then tested on the two test set splits 2015_{A} and 2015_{B} that consist of patents in the first- and second-half of 2015, respectively. We leveraged each patent document as a single training example: its claim texts as input features and the classification symbols (i.e, CPC codes) assigned to it as labels. Among the five hierarchical levels of Section, Class, Subclass, Main Group, Subgroup, we used the slices from Section to Subclass (i.e., subclass-level CPC codes) as labels; label of the CPC code A01N 53/12 [Section: A, Class: 01, Subclass: N, Main Group: 53/00, Subgroup: 53/12] is A01N which is the concatenation of slice [Section: A, Class: 01, Subclass: N]. We concatenated the first 20 claims into a single input text, instead of using the first claim only as in Lee & Hsiang (2020) to prevent the classifier from overfitting on meaningless training examples that are too short to describe each patent's own invention. Since the entire number of class labels are doubled from 664 to 1,328 by the two position attributes *First* and *Later* after relabeling, standard deviation of each dataset split decreases (see Section 5.3) which means that the imbalances between class labels are quite mitigated. Because we used patent data to benchmark the robustness of classification models to domain-specificity and long-tailed distribution, we did not adjust the dataset imbalance further. We collected utility type patents only filed in the United States as training data from the BigQuery table called Google Patents Public Data [2]. Table 2 provides a statistical summary of the dataset.

5.2 Model Architectures

As mentioned repeatedly in previous sections, applying elementwise embedding is to replace the embedding table of the parent transformer architecture. We trained three different transformers BERT (Devlin et al.,

[2] `https://github.com/google/patents-public-data`

Table 1: Configurations of all six models used in the following experiments. Hyperparameters u and v denote the number of materials and of elements per a material. w and c refer to the size of materials and of elements, respectively. w is always a multiple of v and c. Hyperparameter v is also used for ORIG models for intuitive comparison; ORIG models represent a material using one element. h denotes the number of attention heads.

Model	**Parameters**		$\boldsymbol{u}$	$\boldsymbol{v}$	$\boldsymbol{w}$	$\boldsymbol{c}$	$\boldsymbol{h}$
	Total	**Embedding**					
BERT_{EWE}	87M (0.8x)	12k (0.005x)	128	$16(=h)$	768	$48(=w/v)$	$16(=v)$
$\text{BERT}_{\text{ORIG}}$	110M (1x)	23M (1x)	128	1	$768(=c)$	768	12
$\text{ALBERT}_{\text{EWE}}$	9M (0.7x)	2k (0.005x)	128	$16(=h)$	128	$8(=w/v)$	$16(=v)$
$\text{ALBERT}_{\text{ORIG}}$	12M (1x)	4M (1x)	128	1	$128(=c)$	128	12
$\text{CANINE}_{\text{EWE}}$	110M (0.8x)	1M (0.05x)	128	$16(=h)$	768	$48(=w/v)$	$16(=v)$
$\text{CANINE}_{\text{ORIG}}$	130M (1x)	25M (1x)	128	1	$768(=c)$	768	12

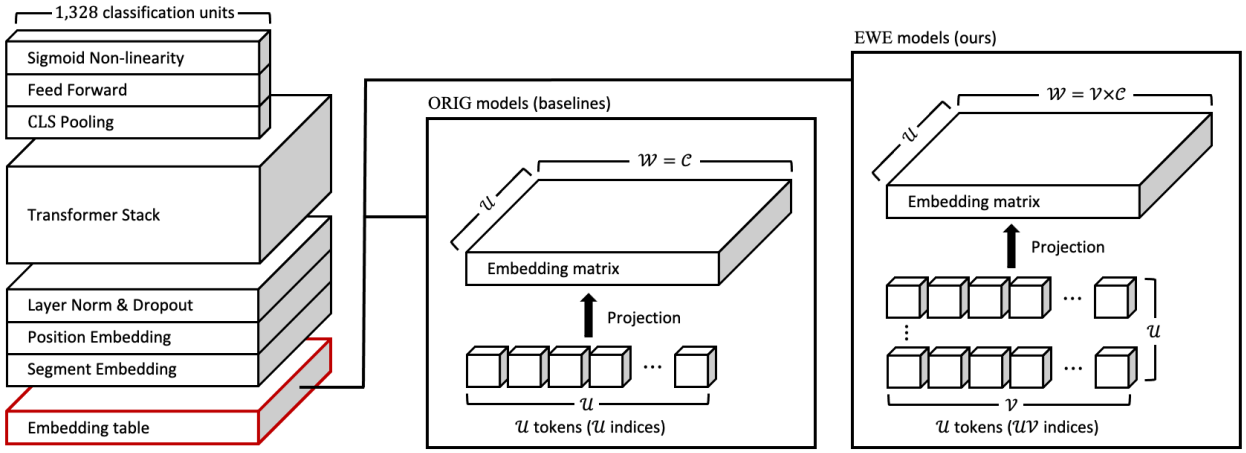

Figure 3: Visualization of the model architectures used in the experiments. As shown in the above graphic, applying elementwise embedding is simply to replace the embedding table of any transformer-based model. Unique structures of CANINE and ALBERT and specific implementation of the standard transformer stack were omitted for visual brevity.

2018), ALBERT (Lan et al., 2019), CANINE (Clark et al., 2022) with elementwise embedding for multilabel patent classification and compared with their original implementations to demonstrate the idea of elementwise language representation. We denote three transformers with elementwise embedding by EWE models such as BERT_{EWE}, $\text{ALBERT}_{\text{EWE}}$, $\text{CANINE}_{\text{EWE}}$ and their original versions, the baselines, as ORIG models like $\text{BERT}_{\text{ORIG}}$, $\text{ALBERT}_{\text{ORIG}}$, $\text{CANINE}_{\text{ORIG}}$. In both theoretical and practical implementations, elementwise embedding does not modify any part of its parent model so that EWE models always have exactly the same architectures as their ORIG equivalences. Every model used in our experiments follow the configuration of $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2018) wherein a transformer encoder has 12 768-dimensional attention layers with 12 heads [3] each followed by a 3072-dimensional linear projection; every model processes $u = 128$ tokens at once but EWE models align v times longer sequence than ORIG models since each token is represented by v embeddings (see Fig 3). Table 1 shows the differences in between configurations of EWE and ORIG models.

5.3 Patent Relabeling

For the purpose of patent classification, the label of each patent document becomes a list of textual symbols classifying it to the corresponding technical categories. Among several classification schemes, the two most

[3]While EWE models use 16 attention heads, this is only to meet the theory of elementwise embedding so that is independent of their superior performances to the ORIG models as shown in the ablation study in Section 6.2

Table 2: A statistical summary of dataset used in the experiments. *Total* is the standard deviation between all CPC codes (class labels), *Majors* denotes the standard deviation between CPC codes that dominate over 90% of the entire labels in training examples. Standard deviations in all three dataset splits were significantly decreased after relabeling (see the second row of the table) which means that the imbalances between class labels were alleviated.

Labeling	**Train Samples**	**Total**	**Majors**	**2015_A Samples**	**Total**	**Majors**	**2015_B Total**	**Total**	**Majors**
Raw	1.9M	21k	35.4k	14.5k	1.8k	3k	15.4k	1.9k	3.2k
Relabeled		12.6k	21k		1.1k	1.9k		1.2k	2k

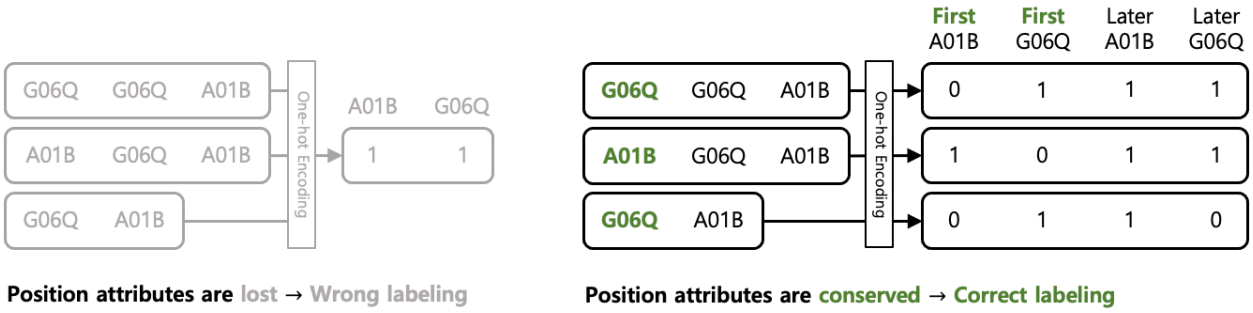

Figure 4: Visualization of correct labeling which separates the first-listed and later classification symbols. Our proposed relabeling strategy (right) conserves the order between symbols (First- and Later-CPC codes) even after being one-hot encoded that are ignored in previous literature (left).

frequently utilized from previous studies are IPC (International Patent Classification) and CPC (Cooperative Patent Classification). One interesting fact is that the assigned symbols have different meanings depending on the order in which they are listed: the first-listed symbol describe the invention of each patent document and the later symbols represent additional information, so that patent documents with classifications [G06Q, G06Q, A01B], [A01B, G06Q, A01B], [G06Q, A01B] should have three different labels. This is in accordance of their documentations (USPTO, 2022; WIPO, 2022), but to the best of our knowledge, previous studies on patent classification (Lim & Kwon, 2017; Li et al., 2018; Yadrintsev et al., 2018; Lee & Hsiang, 2020; Haghighian Roudsari et al., 2022) did not reflect this guide, hence one-hot encoding the above three patents to an identical label [A01B: 1, G06Q: 1] (see the left of Fig 4). This skewed labeling is quite undesirable for both practical patent classification and algorithm benchmarking since the metrics for evaluating the classifier will be messed up by the distorted one-hot encodings.

To address this problem, we simply relabeled our patent documents by attaching position attributes *First* and *Later* as prefixes to their classification symbols. We relabeled the above classifications [G06Q, G06Q, A01B], [A01B, G06Q, A01B], [G06Q, A01B] as [First-G06Q, Later-G06Q, Later-A01B], [First-A01B, Later-G06Q, Later-A01B], [First-G06Q, Later-A01B] so that to be one-hot encoded as [0, 1, 1, 1], [1, 0, 1, 1], [0, 1, 1, 0]; when the placeholder for one-hot encoding is [First-A01B, First-G06Q, Later-A01B, Later-G06Q] (see the right of Fig 4). Relabeled symbols fully satisfy the technical requirements for patent classification described in Section 3.3 and furthermore, the imbalanced distribution between class labels are significantly alleviated because the count of each classification symbol (CPC codes here) is halved by position attributes *First* and *Later* (See Table 2). As a result, all our models trained on relabeled data do well both on Precision and Recall (see Table 3), compared to the case of aforementioned studies that are biased against either score.

5.4 Training Details

This section provides specific configurations to train our models. Every model was trained during 10 epochs using binary cross-entropy loss with sigmoid activation ($threshold = 0.3$) for multilabel patent classification

Table 3: Superiority of transformers trained with elementwise embedding in multilabel patent classification.

Model	2015A F1	Precision	Recall	2015B F1	Precision	Recall
BERT_{EWE}	<u>64.30</u>	<u>66.02</u>	<u>62.66</u>	<u>63.94</u>	66.55	<u>61.53</u>
$\text{BERT}_{\text{ORIG}}$	63.68	65.59	61.82	63.35	<u>67.16</u>	59.95
$\text{CANINE}_{\text{EWE}}$	<u>64.30</u>	<u>65.86</u>	<u>62.82</u>	<u>63.95</u>	<u>66.43</u>	<u>61.64</u>
$\text{CANINE}_{\text{ORIG}}$	60.40	64.08	57.12	59.97	64.52	56.01
$\text{ALBERT}_{\text{EWE}}$	<u>63.18</u>	<u>65.84</u>	<u>60.73</u>	<u>62.91</u>	<u>66.47</u>	<u>59.71</u>
$\text{ALBERT}_{\text{ORIG}}$	63.15	65.82	60.70	62.79	66.36	59.59

from scratch without pretraining. We used AdamW (Loshchilov & Hutter, 2017) as optimizer ($\beta_1 = 0.9$ and $\beta_2 = 0.999$, $eps = 1e-8$) with L_2 regularization ($\lambda = 0.01$). Initial learning rate decays linearly from $2e-5$ without a warmup period and batch size was set to 32; we selected these small values to prevent overfitting due to imbalanced training examples. Larger memory-safe batch sizes did not show meaningful differences, only slowing convergence. All models were trained and tested using a single 24GB VRAM GPU (NVIDIA RTX TITAN) with FP16 mixed-precision.

5.5 Performance Measures

For evaluating models for multilabel patent classification, we utilize three metrics Precision, Recall and their harmonic mean F_1 measure. Because all CPC codes (technical categories of patents; class labels) are equally important (current technical categories draw long tail when counted by the number of the patent documents that they are classifying; only 30% of the categories contain over 90% of the entire patents, mainly due to the popularity of each field of technology; but all of them are potentially valuable) we compute micro-averaged scores (TP, TN, FP, FN) for all metrics by $\text{Score} = \sum_l \text{Score}_l$. Since we consider a total of 664 CPC symbols as class labels with two position attributes *First* and *Later*, $l \in [1, 1328]$ is established. As mentioned in Section 3.3, $\text{Precision} = \text{TP} \cdot (\text{TP} + \text{FP})^{-1}$ captures how well a model identifies the unique classifications of each patent document, $\text{Recall} = \text{TP} \cdot (\text{TP} + \text{FN})^{-1}$ implies how well a model distinguishes between different patent documents which is critical for patent search engine optimization. As Precision and Recall are equally important for patent classification, we utilize their $\text{F}_1 = 2 \cdot (\text{Precision}^{-1} + \text{Recall}^{-1})^{-1}$ as the main metric.

6 Results

This section presents the experimental results demonstrating the validity of the proposed method explained so far. Because there are no comparable state-of-the-arts in multilabel patent classification that comply the guide to the position attributes (see Section 3.3), we compare the results with our own baselines. Every EWE model was trained using whitespace tokenizer for a fair comparison with subword-level models ($\text{BERT}_{\text{ORIG}}$ and $\text{ALBERT}_{\text{ORIG}}$); we analyze tokenization-free EWE models in ablation study in Section 6.2. As shown in Table 3, all EWE models surpass their corresponding baselines in multilabel patent classification on all test-set splits 2015_{A} and 2015_{B}. Used patent dataset is highly imbalanced and contain massive amounts of unusual technical lexicons, so that the superior classification performances of EWE models on it show clear robustness to domain-specificity and long-tailed distributions improved by elementwise language representation. Note that all EWE models have much less embedding parameters than their original implementations.

$\text{ALBERT}_{\text{EWE}}$ and $\text{CANINE}_{\text{EWE}}$ are both improved by elementwise embedding while maintaining their own design choices: the shallow transformer layer for character-level encoding and 1D strided convolutional layers for sequence downsampling of CANINE (Clark et al., 2022), and the factorized embedding parameterization and parameter sharing of ALBERT (Lan et al., 2019). These empirically demonstrate the generalizability of elementwise language representation. Performance enhancement in $\text{ALBERT}_{\text{EWE}}$ is relatively smaller than in other EWE models and it's presumably because it uses much lower-dimensional embeddings as elements ($c = 8$) than others (with $c = 48$). Based on this observation, using c greater or equal than 8 is recommended. $\text{CANINE}_{\text{EWE}}$ shows the largest improvement among all EWE models, however, its classification performance

is at the same level as BERT_{EWE} that has 40% fewer parameters, so it is unclear whether additional sequence downsampling gives a meaningful benefit when elementwise embedding is already applied. All EWE models process $v = 16$ times longer sequences than their ORIG counterparts at the same $O(N^2)$ computational complexity. The overhead of the one-time tensor reshaping operation for elementwise language representation is negligible and it can even be removed by technical optimizations such as JIT (Just In Time) compilation, so there is no meaningful difference in inference speeds between EWE and ORIG models [4].

Table 4: Effect of focus embeddings on the convergence of transformers trained with elementwise embedding.

Model	**Ablation**	**2015A**			**2015B**		
		F1	**Precision**	**Recall**	**F1**	**Precision**	**Recall**
BERT_{EWE}	None	<u>64.30</u>	66.02	<u>62.66</u>	<u>63.94</u>	66.55	<u>61.53</u>
	Focus embeddings	63.22	<u>66.14</u>	60.55	62.91	<u>66.69</u>	59.54

6.1 Effect of Focus Embeddings

Originally, the main design goal of focus embedding was to stabilze the convergence of models trained with elementwise embedding. While the idea of elementwise embedding works well without focus embeddings (see Table 4), training parent models with it was somewhat tricky; the starting point of meaningful convergence differed randomly at each training trial, and therefore the reproducibility was not ensured. Focus embeddings guarantee the stable convergence of elementwise models by explicitly encoding the global and local positions of focusable characters. We found that focus embeddings also improve classification performance of parent transformer, so set them as the default component of elementwise embedding. In the case where the positional information is supplemented in some other way e.g, n-gram as in tokenization-free BERT_{EWE} (see Section 6.2) focus embedding did not provide a meaningful enhancement in both stability and performance, however, more research is needed on how their absence will affect other applications as sequence-to-sequence modeling. Table 4 shows the ablation study on focus embeddings BERT_{EWE} classifier.

Table 5: Effect of other tokenization on the convergence of transformers trained with elementwise embedding.

Model	**Tokenization**	**2015A**			**2015B**		
		F1	**Precision**	**Recall**	**F1**	**Precision**	**Recall**
BERT_{EWE}	None	60.01	63.77	56.67	59.80	64.49	55.75
	Gradient	64.14	65.91	62.45	63.75	66.41	61.29
	Whitespace	<u>64.30</u>	<u>66.02</u>	<u>62.66</u>	<u>63.94</u>	<u>66.55</u>	<u>61.53</u>

6.2 Effect of Tokenization Strategies

In this section, we generalize the idea of elementwise embedding to tokenization-free language representation. We implement two versions of tokenization-free BERT_{EWE}: one using pure UTF-8 byte-level tokenizer[5] and one using gradient-based tokenizer. Tokenization-free elementwise language representation follows the same framework described in Fig 2 and Section 4.2 [6]. First, BERT_{EWE} trained using raw UTF-8 bytes encoding (see the first row of Table 5) shows significantly poor performance compared to one trained using whitespace tokenizer (see the third row Table 5), but still equivalent to $\text{CANINE}_{\text{ORIG}}$ which is 1.3x larger. This version of BERT_{EWE} shares the same hyperparameters with whitespace version which set $n = v = 16$ and $u = 128$.

Notably, tokenization-free BERT_{EWE} recovers the same level of performance as one with whitespace tokenizer when trained using gradient-based tokenizer (see the second row of Table 5). In this implementation, each

[4] Although reshaping tensors in GPU entails physical arrangements so is relatively more expensive than in CPU, the cost of reshaping from (uv, c) to $(u, w = vc)$ is trivial and can be optimized well by technical tricks like asynchronous dispatch and JIT compilation of XLA; so elementwise embedding does not slower its parent transformer.

[5] UTF-8 bytes encoding can be achieved by `list(bytes("text/to/encode", "utf-8"))` in Python3.

[6] For tokenization-free elementwise language representation, text is encoded to a sequence of uv UTF-8 bytes, projected into a (uv, c) element embedding matrix, then reshaped into a (u, w) material embedding matrix directly without any tokenization.

element is replaced with softmax-weighted sum-pooled v-gram[7]:

$$e^{(i)[j]} \leftarrow \sum_{j=1}^{v} \alpha_j e^{(i)[j]}$$

where $\alpha_j = \mathrm{softmax}_j(e^{(i)[j]}s)$ and $s \in \mathbb{R}^c$ is the weight vector for linear projection $\mathbb{R}^c \mapsto \mathbb{R}$. Since aggregated v-grams implicitly encode the positions of focusable elements, focus embeddings are not used for this model. $n = v = 8$ attention heads are used. Operations for v-gram pooling cause overhead but is trivial, so that the resulting decrease in inference speed is negligible. This type of elementwise representation is expected to be useful for the tasks where consistent tokenization is difficult (e.g., multilingual applications which deal with heterogeneous language systems simultaneously).

7 Discussion

So far, we explored the new elementwise language representation from both theoretical and practical aspects.

This framework gives advantages of:

- being generalized to every level of language representation
- being able to process longer sequences at the same complexity as model scales
- being able to reuse existing transformer architectures for all levels of language representation

Neither architectural modification nor additional computational overheads occured.

This framework remains several challenges that:

- does not reflect various other linguistic components than semantics
- has not yet proposed an optimal strategy for unsupervised pretraining
- still requires separate embedding parameters for language representation

This framework suggests new research directions which can be extended as follows:

1. pretraining a single language model understanding all kinds of tokenization
2. pretraining a language model with multiple levels of semantics at the same time
3. integrating representations of all types of languages (text, image, graph, etc.) into bytes

Recent groundbreaking studies (Brown et al., 2020; Raffel et al., 2020; Jaegle et al., 2021; Reed et al., 2022) have successfully demonstrated that it is possible to pretraining multimodal, multitasking neural networks. By expanding their contributions with these research directions, computers will finally be able to understand the world solely based on their native language, bytes.

8 Conclusion

In this paper, we proposed elementwise embedding that is a technique for generalized language representation. To the best of our knowledge, this is the first case of computational language representation:

- applying with all levels of tokenization strategies
- aligning longer sequences proportional to the model size
- reusing existing transformers for all levels of language representations

without either any architectural modification or degradation in performance and inference speed.

We expand these contributions in our follow-up studies discussing:

- elementwise representation for other types of data as images and graphs
- unsupervised pretraining approach for elementwise language representation
- language modeling method leveraging multiple levels of semantics simultaneously

[7]This can be understood as the simplified version of *Block scoring network* proposed in Tay et al. (2021).

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Jonas Belouadi and Steffen Eger. Bygpt5: End-to-end style-conditioned poetry generation with token-free language models. *arXiv preprint arXiv:2212.10474*, 2022.

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

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Jonathan H Clark, Dan Garrette, Iulia Turc, and John Wieting. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91, 2022.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Nathan Godey, Roman Castagné, Éric de la Clergerie, and Benoît Sagot. Manta: Efficient gradient-based tokenization for robust end-to-end language modeling. *arXiv preprint arXiv:2212.07284*, 2022.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. *Advances in neural information processing systems*, 30, 2017.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Arousha Haghighian Roudsari, Jafar Afshar, Wookey Lee, and Suan Lee. Patentnet: multi-label classification of patent documents using deep learning based language understanding. *Scientometrics*, pp. 1–25, 2022.

Xuanli He, Gholamreza Haffari, and Mohammad Norouzi. Dynamic programming encoding for subword segmentation in neural machine translation. *arXiv preprint arXiv:2005.06606*, 2020.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Tatsuya Hiraoka, Sho Takase, Kei Uchiumi, Atsushi Keyaki, and Naoaki Okazaki. Joint optimization of tokenization and downstream model. *arXiv preprint arXiv:2105.12410*, 2021.

Itay Itzhak and Omer Levy. Models in a spelling bee: Language models implicitly learn the character composition of tokens. *arXiv preprint arXiv:2108.11193*, 2021.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pp. 5156–5165. PMLR, 2020.

Jeeeun Kim and Sungjoo Lee. Forecasting and identifying multi-technology convergence based on patent data: the case of it and bt industries in 2020. *Scientometrics*, 111(1):47–65, 2017.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. Block pruning for faster transformers. *arXiv preprint arXiv:2109.04838*, 2021.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Jieh-Sheng Lee and Jieh Hsiang. Patent classification by fine-tuning bert language model. *World Patent Information*, 61:101965, 2020.

Sungjoo Lee, Byungun Yoon, Changyong Lee, and Jinwoo Park. Business planning based on technological capabilities: Patent analysis for technology-driven roadmapping. *Technological Forecasting and Social Change*, 76(6):769–786, 2009.

Shaobo Li, Jie Hu, Yuxin Cui, and Jianjun Hu. Deeppatent: patent classification with convolutional neural networks and word embedding. *Scientometrics*, 117:721–744, 2018.

Sora Lim and YongJin Kwon. Ipc multi-label classification applying the characteristics of patent documents. In *Advances in Computer Science and Ubiquitous Computing: CSA-CUTE2016 8*, pp. 166–172. Springer, 2017.

Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Wentao Ma, Yiming Cui, Chenglei Si, Ting Liu, Shijin Wang, and Guoping Hu. Charbert: character-aware pre-trained language model. *arXiv preprint arXiv:2011.01513*, 2020.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. Bpe-dropout: Simple and effective subword regularization. *arXiv preprint arXiv:1910.13267*, 2019.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pp. 8459–8468. PMLR, 2020.

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

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958, 2014.

Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackermann, et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702, 2020.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 1017–1024, 2011.

Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. Distilling task-specific knowledge from bert into simple neural networks. *arXiv preprint arXiv:1903.12136*, 2019.

Yi Tay, Vinh Q Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. Charformer: Fast character transformers via gradient-based subword tokenization. *arXiv preprint arXiv:2106.12672*, 2021.

Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085*, 2022.

USPTO. Manual of patent examining procedure. `https://www.uspto.gov/web/offices/pac/mpep/s905.html`, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Xinyi Wang, Sebastian Ruder, and Graham Neubig. Multi-view subword regularization. *arXiv preprint arXiv:2103.08490*, 2021.

WIPO. Guide to international patent classification. `https://www.wipo.int/publications/en/details.jsp?id=4593&plang=EN`, 2022.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022.

Vasiliy Yadrintsev, Amir Bakarov, Roman Suvorov, and Ilya Sochenkov. Fast and accurate patent classification in search engines. In *Journal of Physics: Conference Series*, volume 1117, pp. 012004. IOP Publishing, 2018.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

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

A Appendix

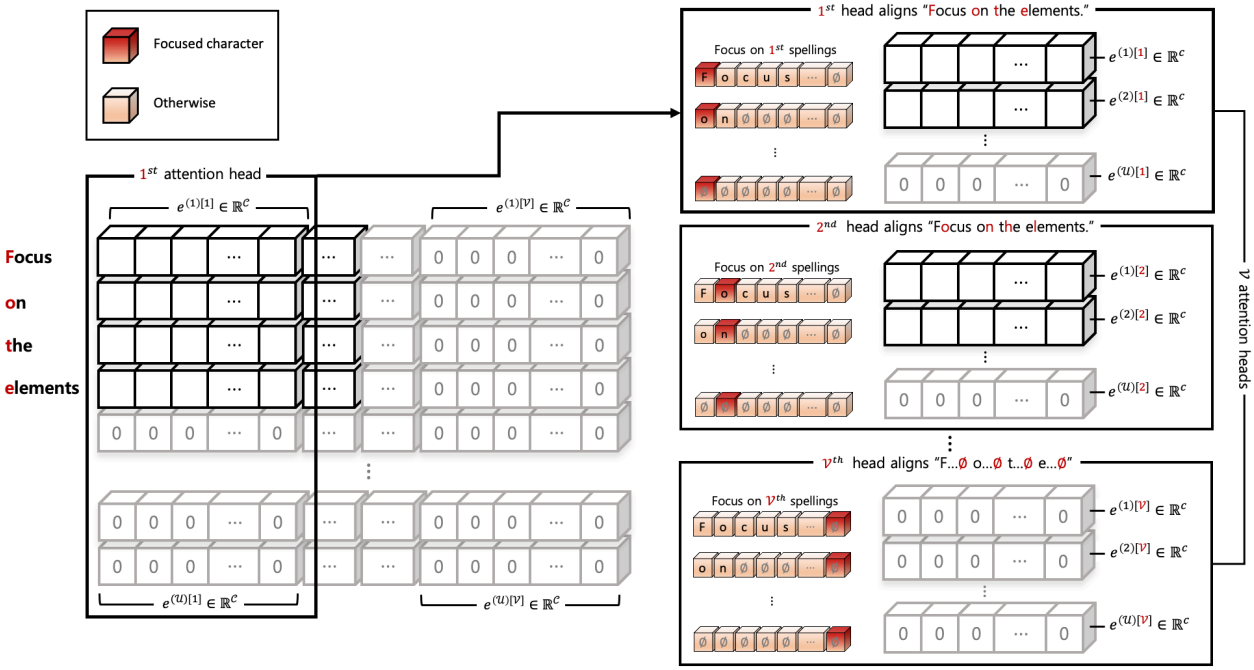

Figure 5: Visualization of the focus operation in multihead attention. n^{th} attention head aligns n^{th} characters (i.e., elements) hence focusing on the most important ones, when the number of attention heads is set to v.