ADAPTING PRE-TRAINED LANGUAGE MODELS FOR QUANTUM NATURAL LANGUAGE PROCESSING

Anonymous authors
Paper under double-blind review

ABSTRACT

The emerging classical-quantum transfer learning paradigm has brought a decent performance to quantum computational models in many tasks, such as computer vision, by enabling a combination of quantum models and classical pre-trained neural networks. However, using quantum computing with pre-trained models has yet to be explored in natural language processing (NLP). Due to the high linearity constraints of the underlying quantum computing infrastructures, existing Quantum NLP models are limited in performance on real tasks. We fill this gap by pre-training a sentence state with complex-valued BERT-like architecture, and adapting it to the classical-quantum transfer learning scheme for sentence classification. On quantum simulation experiments, the pre-trained representation can bring 50% to 60% increases to the capacity of end-to-end quantum models.

1 INTRODUCTION

Quantum computing combines quantum mechanics and computer science. The concepts of superposition and entanglement bring inherent parallelism between *qubits*, the basic computational element, which endow enormous computational power to quantum devices. Classical–quantum transfer learning (Mari et al., 2020) has emerged as an appealing quantum machine learning technique. As shown in Fig. 1, in a classical–quantum transfer learning pipeline, the pre-trained input features are encoded to a multi-qubit state, transformed and measured in a quantum circuit. The output probabilities are projected to the task label space. The losses are backpropagated to update the parameters in the pipeline with classical algorithms. This transfer learning scheme combines the representation power of state-of-the-art (SOTA) pre-trained models and the computational power of quantum computing, yielding decent accuracy on various image classification tasks (Lloyd et al., 2020; Mogalapalli et al., 2021; Mari et al., 2020; Oh et al., 2020).

However, combining pre-trained models and quantum computing remains unexplored in NLP, where large-scale pre-trained models have dramatically improved language representation (Devlin et al., 2019; Radford & Sutskever, 2018). Current Quantum NLP (QNLP) models (Zeng & Coecke, 2016; Coecke et al., 2020; Meichanetzidis et al., 2020; Lorenz et al., 2021; Lloyd et al., 2020; Kartsaklis et al., 2021b) mainly construct quantum circuits from a certain kind of tensor network that aggregates word vectors to sentence representations (Coecke et al., 2020), and the parameters in the network are randomly initialized and learned end-to-end. Since a quantum circuit can be seen as a linear model in the feature space (Schuld & Petruccione, 2021), these models are highly restricted in scalability and effectiveness. One attempt to resolve this issue is hybrid classical-quantum models (Li et al., 2022), where certain layers of models are implemented on a quantum device, and the intermediate results are sent to classical models for non-linear operations. However, the frequent switching between classical and quantum processing units significantly drags the speed of training and inference, and limits the applicability of the model.

We posit that classical-quantum transfer learning paradigm is a sound fit for QNLP models, especially in the current noisy intermediate-scale quantum (NISQ) era. The pre-trained language features can lead to strong performance in downstream natural language understanding tasks Devlin et al. (2019), even with simple models. Furthermore, it is crucial to introduce robust quantum encodings to mitigate the errors caused by the noisy quantum device, and pre-trained language model is a promising approach to this aim due to its high robustness Qiu et al. (2020). Finally, pre-trained mechanism

can contribute to scalable QNLP models by avoiding tensor product of all token vectors and using fixed-dimensional representations for arbitrarily long sentences.

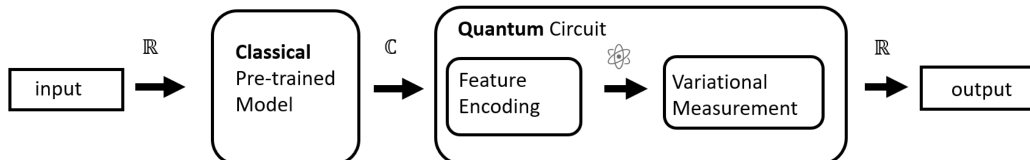

Figure 1: Classical-quantum transfer learning pipeline Mari et al. (2020).

We are motivated to pre-train language representations compatible with quantum computing models. Due to the crucial role of complex numbers in quantum computing, we build a complex-valued pre-trained language model (PLM) for classical-quantum transfer learning. Complex-valued neural networks (NNs) have been long studied (Georgiou & Koutsougeras, 1992; Nitta, 2002; Hirose, 2011; Trabelsi et al., 2018; Xiang et al., 2020; Yang et al., 2020) with various NN building blocks including RNN (Wisdom et al., 2016), CNN (Guberman, 2016) and Transformers (Yang et al., 2020; Wang et al., 2020; Tay et al., 2019; Zhang et al., 2021), and have shown advantages in enhanced representation capacity (Wisdom et al., 2016; Arjovsky et al., 2016; Trabelsi et al., 2018; Wang et al., 2020; Trouillon et al., 2016), faster learning speed (Arjovsky et al., 2016; Danihelka et al., 2016), and increased model robustness (Danihelka et al., 2016; Yeats et al., 2021; Xiang et al., 2020). Despite these advances, complex numbers are not used in pre-trained language models. It remains unknown whether complex-valued NN building blocks can be integrated into high-quality pre-trained models.

To adapt the complex-valued pre-trained models to QNLP, we impose numerical constraints to the network components. For feature encoding, we unit normalize the hidden vectors of the [CLS] token so that the sentence representation can be mapped to a quantum state all throughout the network. We also re-implement the next sentence prediction (NSP) head to simulate variational measurement. At fine-tuning, we train an authentic task-related variational measurement process by parameterizing the involved unitary transformation. Despite the imposed numerical constraints, the quantum-compatible pre-trained language model performs in par to a real-valued BERT of comparable size. More importantly, the pre-trained sentence state encoding brings remarkable performance gain to end-to-end quantum models on various classification datasets, with an relative accuracy improvement of around 50% to 60%.

We contribute the first approach to introduce the classical-quantum transfer learning for QNLP models, and pre-train language representations compatible with quantum computing models. Apart from the remarkably improved performance, our model is more scalability then existing NLP models and can tackle longer sentences.

2 Related work

2.1 Complex-valued Neural Networks

Complex values have been used in various NNs across domains (Arjovsky et al., 2016; Danihelka et al., 2016; Wisdom et al., 2016; Trouillon et al., 2016; Hirose, 2011; Trabelsi et al., 2018; Xiang et al., 2020; Yang et al., 2020; Guberman, 2016; Wang et al., 2019). Arjovsky et al. (2016); Wisdom et al. (2016); Wolter & Yao (2018) studied complex numbers in recurrent NNs. Arjovsky et al. (2016) systematically studied variants of CNNs with complex-valued inputs and weights, which led Trabelsi et al. (2018) to build a complex-valued NN that achieved SOTA performance in audio-related tasks. Wang et al. (2019); Yang et al. (2020); Zhang et al. (2021); Tay et al. (2019) obtained promising results on sequence-to-sequence (seq2seq) tasks with a complex-valued Transformer. Complex-valued NNs have also been used in privacy detection (Xiang et al., 2020) and knowledge graph completion (Trouillon et al., 2016; Trouillon & Nickel, 2017).

Apart from effectiveness gains (Wisdom et al., 2016; Arjovsky et al., 2016; Trabelsi et al., 2018; Wang et al., 2020; Trouillon et al., 2016), complex-valued NNs also contribute to faster learning (Arjovsky et al., 2016; Danihelka et al., 2016) and increased model robustness (Danihelka et al., 2016; Yeats

et al., 2021; Xiang et al., 2020). However, these properties have been found only in end-to-end tasks. The impact of complex values to pre-trained models remains unexplored.

2.2 QUANTUM NATURAL LANGUAGE PROCESSING (QNLP)

QNLP aims to build NLP models compatible with quantum hardware. Current QNLP models are limited in their architecture and application (Coecke et al., 2020; Meichanetzidis et al., 2020; Lorenz et al., 2021; Lloyd et al., 2020); they are based on a compositional model (Coecke et al., 2010), which represents words as tensors in spaces dependent on their grammatical roles, and performs tensor operations to encode syntactic relations. Sentence representations are built by bottom-up aggregating individual word tensors. This process is translated to quantum circuits, followed by a quantum measurement component to produce classification labels. Preliminary studies have successfully implemented and simulated QNLP models for sentence classification (Meichanetzidis et al., 2020; Lorenz et al., 2021). By quantum simulation in the feedforward pass and performing backpropagation on a classical computer, the model can learn from data and outperform random labeling.

Like other quantum machine learning models (Lloyd et al., 2020; Jerbi et al., 2021), QNLP models encode words to different *qubits*, and design the quantum circuit routine (a.k.a *ansatz*) to derive the sentence representation before feeding it to a measurement layer. In its mathematical form, the model encodes each word to a unit complex vector and it has an all-linear structure up to the measurement outcome. Therefore, they have a low capacity and suffer from the scalability issue. Recently, a hybrid classical-quantum scheme (Li et al., 2022) is introduced to alleviate the limitations. In this quantum self-attention neural network (QSANN), a quantum process is introduced, with parameterized unitary rotations and Pauli measurement, to compute the query, key and value vectors, and they are sent to classical computer to perform non-linear attentions. Due to the introduced non-linearity, QSANN beats the QNLP model in (Lloyd et al., 2020). However, running the network requires switching between quantum and classical hardware at each self-attention layer, which is too inefficient to be practical. We therefore posit that the classical-quantum transfer learning paradigm (Mari et al., 2020) is a more promising solution for alleviating the low non-linearity issue for QNLP models.

3 BACKGROUND

Complex number. A complex number z is written as $z = a + bi$ in the rectangular form or $z = re^{i\theta} = r(\cos\theta + i\sin\theta)$ in the polar form. a, b are the real and imaginary parts of z, written as $\mathfrak{Re}$(z) and $\mathfrak{Im}$(z). $r = |z| \in [0, +\infty)$ and $\theta \in [-\pi, \pi)$ are the *modulus* (or *amplitude*) and *argument* (or *phase*). The conjugate of a complex number $z = a + bi$ is $\overline{z} = a - bi$, which could be extended for vectors and matrices. The Hermitian of a complex matrix A is its conjugate transpose, written as $A^H = \overline{A^T}$. A is an Hermitian matrix when $A = A^H$. An orthogonal or unitary complex matrix U satisfies $U^H U = I$. We use the typical complex-valued fully-connected (dense) layer to illustrate complex additions and multiplications. A complex dense layer is parameterized by a complex matrix $\boldsymbol{W} = \boldsymbol{A} + i\boldsymbol{B}$ and a complex bias $\boldsymbol{b} = \boldsymbol{c} + i\boldsymbol{d}$. For a complex input vector $\boldsymbol{X} = \boldsymbol{x} + i\boldsymbol{y}$, the output is a complex-valued multiplication of the weight and the input vector, plus the bias value:

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{X} + \boldsymbol{b} = (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{y} + \boldsymbol{c}) + i(\boldsymbol{B}\boldsymbol{x} + \boldsymbol{A}\boldsymbol{y} + \boldsymbol{d}) \tag{1}$$

The **mean and variance** of a set of complex numbers $\{z_j\}_{j=1}^n$ are given below:

$$\begin{aligned} \bar{\mathbf{z}} &= \frac{\sum_{j=1}^n z_i}{n} \\ \sigma_z^2 &= \frac{\sum_{j=1}^n (z_j - \bar{\mathbf{z}})\overline{(z_j - \bar{\mathbf{z}})}}{n}. \end{aligned} \tag{2}$$

Quantum Computing. We present the basic concepts of quantum computing, see (Nielsen & Chuang, 2010) for more. The basic computing unit is a *qubit*, the quantum analog of a classical bit. A qubit describes the state $|\psi\rangle$[1] in a 2-dim Hilbert space. The *basis states* $|0\rangle$ and $|1\rangle$ are orthonormal vectors that form the basis of the space. A general state $|\psi\rangle$ is a *superposition* of the basis states, i.e. $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α, β are complex numbers with $|\alpha|^2 + |\beta|^2 = 1$.

[1] In Dirac's Notations, $|\psi\rangle$ and $\langle\psi|$ refer to a complex-valued unit row and column vector, respectively.

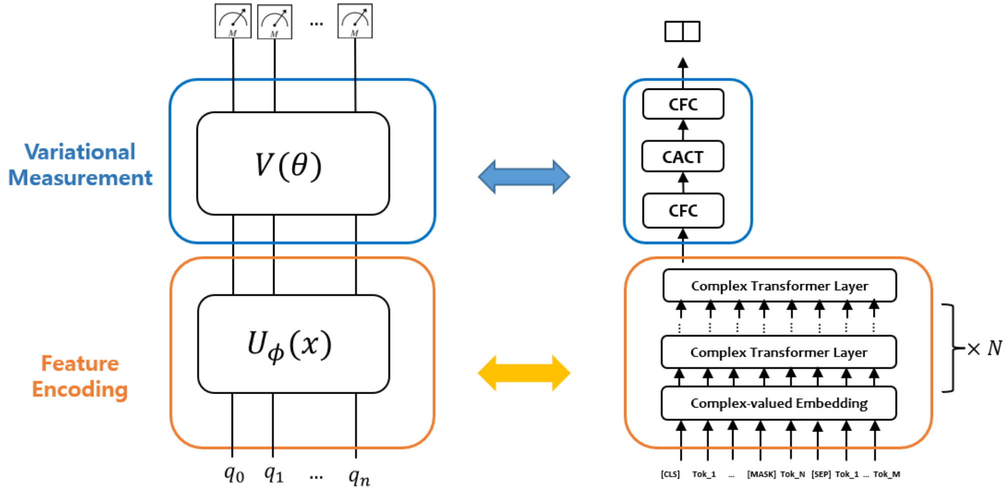

Figure 2: Mapping the classical-quantum transfer learning scheme to our complex-valued PLM. The MLM prediction head for PLM and the linear projection layer of the quantum model are omitted.

One can apply *measurement* to a qubit to check its probabilities of outcomes 0 and 1 by Born's rule (Born, 1926), i.e, $P(i) = |\langle\psi|i\rangle|^2$, so $P(0) = |\alpha|^2$ and $P(1) = |\beta|^2$ for the state above. The probabilities of all outcomes always sum to 1. For multiple qubits, their joint space is the tensor product of each qubit space, hence of dimension 2^n for n qubits, and the basis states are denoted by $\{|a_1a_2...a_n\rangle, a_i \in \{0,1\}\}$. A state transformation is mathematically a unitary map or a complex unitary matrix U, such that $|\psi'\rangle = U|\psi\rangle$ for state $|\psi\rangle$. Quantum circuits are physical implementations of quantum computing models. The basic units of quantum circuits are quantum gates, which are unitary maps that play similar roles to logic gates in classical computers. The combination of different quantum states allows us to implement any unitary transformation before the final measurement step.

Classical BERT. Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2019) takes as input a concatenation of two segments (sequences of tokens). The inputs are passed through an embedding layer that adds positional embedding, token embedding and segment embedding. The embeddings are then fed into a stack of N transformer layers, and each layer has a multi-head attention module to enact token-level interactions. The last hidden units of each sequence are used to perform Mask Language Model (MLM) and Next Sentence Prediction (NSP). The MLM objective $\mathcal{L}_{MLM}$ is a cross-entropy loss on predicting the randomly masked tokens in the sequence, while the NSP loss $\mathcal{L}_{NSP}$ produces binary cross-entropy loss on predicting whether the two segments follow each other in the original text. The overall objective of BERT is $\mathcal{L}_{BERT} = \mathcal{L}_{MLM} + \mathcal{L}_{NSP}$. BERT is pre-trained on large text corpora, e.g., BOOKCORPUS and the English WIKIPEDIA (Devlin et al., 2019). The model is fine-tuned on different text classification and natural language inference datasets, such as the famous GLUE benchmark (Wang et al., 2018). The effectiveness on these datasets indicates the performance of the pre-trained model.

4 Our quantum-compatible pre-trained language model

4.1 Overall Architecture

A typical quantum circuit for classification is composed of a feature encoding module and a variational measurement component. The feature encoding unit encodes an input data point x, such as a sentence, to any n-qubit state $|\psi(x)\rangle$. Basically, it applies a unitary transformation $U_\phi(x)$ to the n-qubit basis state, i.e. $|\psi(x)\rangle = U_\phi(x)|00...0\rangle$. Next, the encoded state $|\psi(x)\rangle$ is fed to a variational measurement unit, which consists of a task-related unitary transformation $V(\theta)$ and a measurable observable along the basis states $\{|e_j\rangle\}$ of the n-qubit system. The output probabilities on all basis states $p_j = |\langle e_j|V(\theta)|\psi(x)\rangle|^2$ are aggregated as the circuit output. They are further projected onto a low-dimensional space to produce the task label.

We establish that the above schema can be structurally mapped to a pre-trained language model, as shown in Fig. 2. The [CLS] token vector is usually viewed as the sequence representation. The multi-layer Transformer encoder encodes its input to a quantum state $|\boldsymbol{\psi_{[CLS]}(x)}\rangle$, playing the role of quantum feature encoding in a quantum circuit. Essentially, the unitary transformation $U_\phi(x)$ is parameterized such that the transformed state $U_\phi(x)\,|00...0\rangle = |\psi_{[CLS]}(x)\rangle$. Furthermore, the NSP prediction head that maps [CLS] token to classification label is analogous to the variational measurement component that directs $|\boldsymbol{\psi_{[CLS]}(x)}\rangle$ to the classification label.

Since complex values are vital to quantum models, we build a pre-trained LM with complex values, namely QBERT, to support the mapping above. We follow the multi-layer bidirectional Transformer architecture of classical BERT, and adjust the implementations of each network component to support complex representations. For feature encoding, we unit normalize the hidden vectors of the [CLS] token so that the sentence representation can be attributed to a quantum state at each layer of the network. We also re-implement the NSP prediction head to simulate variational measurement in both pre-training and fine-tuning phase.

4.2 QBERT BUILDING BLOCKS

Embedding layer. In classical BERT, token embeddings, segment embeddings and position embeddings are summed up at a per-token level. They are each extended to the complex domain in QBERT. Essentially, the complex-valued token embeddings, segment embeddings and position embeddings are summed up as the output of the embedding layer for each token.

Multi-head attention. A complex transformer layer has a multi-head attention component at its core. It computes query-key affinity scores, and linearly combines them with value vectors to produce a contextual vector for each element. The attention scores for one head are computed by

$$\text{ComplexAttention}(Q, K, V) = f(\frac{QK^H}{\sqrt{d_k}})V, \tag{3}$$

where $f(\cdot)$ is a softmax-like activation function applied on the query-key complex inner (i.e. Hermitian) products. Since the inputs to this function are complex matrices, we extend the real softmax function to the complex domain.

A straightforward approach to this aim is to apply softmax to real and imaginary parts of the inner product separately. Suppose the Hermitian product is denoted by $\{\sigma(q, k)\}$ for a pair of query-key elements (q, k), the formula of this *split activation function* is given by

$$f_{\text{split}}(q, k) = \frac{e^{\mathfrak{Re}(\sigma(q,k))}}{\sum_{k'} e^{\mathfrak{Re}(\sigma(q,k'))}} + i\frac{e^{\mathfrak{Im}(\sigma(q,k))}}{\sum_{k'} e^{\mathfrak{Im}(\sigma(q,k'))}}, \tag{4}$$

where the summation iterates over all key elements k' in the sequence. The split softmax function normalizes both real and imaginary parts of the affinity scores to sum up to 1 for each key. When the scores are taken to linearly combine the value vectors $\{v'\}$, the summation can be decomposed into

$$\begin{aligned} h = &\sum_{k'} (\mathfrak{Re}(f_{\text{split}}(q, k')) + i\mathfrak{Im}(f_{\text{split}}(q, k'))(\mathfrak{Re}(v') + i\mathfrak{Im}(v')) \\ &= \sum_{k'} (\mathfrak{Re}(f_{\text{split}}(q, k'))\mathfrak{Re}(v') - \mathfrak{Im}(f_{\text{split}}(q, k'))\mathfrak{Im}(v')) \\ &+ i\sum_{k'} (\mathfrak{Re}(f_{\text{split}}(q, k'))\mathfrak{Im}(v') + \mathfrak{Re}(f_{\text{split}}(q, k'))\mathfrak{Im}(v')), \end{aligned}$$

and a negative sign exists in the real part of the summation due to the $i^2 = -1$. However, an ideal weighted combination should be a convex combination of the value vectors with all non-negative weights. This motivates us to modify the normalization function f so that real-valued affinity scores are produced from the complex inputs, because they indicate a convex combination in both the real and imaginary channels. We propose to apply softmax normalization to the modulus part of the complex numbers, as shown in the equations below.

$$f_{\text{mod}}(q,k) = \frac{e^{|\sigma(q,k)|}}{\sum_{k'} e^{|\sigma(q,k')|}} \tag{5}$$

Feed-forward network. The main component of feed-forward networks is the fully-connected layer. We employ Eq. 1, Sec. 3 for The implementation of a complex fully-connected layer. For an input vector $\boldsymbol{X} \in \mathbb{C}^{d_i}$, a fully-connected layer projects it to a d_o-dim vector $z \in \mathbb{C}^{d_o}$, with weight matrix $\boldsymbol{W} \in \mathbb{C}^{d_o \times d_i}$ and a bias term $\boldsymbol{b} \in \mathbb{C}^{d_o}$.

Prediction heads. In order to be consistent with the quantum classification model, we re-implement the NSP head to simulate a variational measurement, which consists of unitary training and measurement. Due to the computational cost of training a unitary matrix, we replace the unitary transformation with a dense layer followed by unit-normalization in the pre-training phase Chen et al. (2021). Forr the measurement, two pure states are for the NSP task, and the squared Hermitian product between the input state and each state is computed and linearly re-scaled to discrete probabilities. As per Lorenz et al. (2021), the probabilities are used to compute the binary cross-entropy loss against the true binary label. We further remove the non-linear activation function from the NSP prediction head.

For MLM head, a complex-valued feed-forward network is used to project the encoder-output tokens to MLM logits, which are then converted to real values by taking the moduli of complex numbers.

Activation function. Classical BERT typically adopts Rectified Linear Unit (ReLU) (Nair & Hinton, 2010) or the Gaussian Error Linear Unit (GeLU) (Hendrycks & Gimpel, 2016) as the activation function for hidden units. To extend it to the complex domain, we simply employ **split-GeLU**, which activates the real and imaginary parts of the input with a GeLU function:

$$\text{split-GeLU}(z) = \text{GeLU}(\mathfrak{Re}(z)) + i\text{GeLU}(\mathfrak{Im}(z)) \tag{6}$$

Layer normalization. For a real vector, standard layer normalization rescales the elements to zero mean and unit variance, and applies an affine transformation to the rescaled values. Similarly, one can directly compute the mean $\bar{z}$ and variance σ_z for a set of complex numbers $\boldsymbol{z} = \{z_j\}_{j=1}^n$ by Eq. 2. The complex layer normalization function becomes

$$\text{complex-LN}(z) = \frac{z - \bar{z}}{\sigma_z} \times a + b. \tag{7}$$

Complex-LN is slightly different from applying layer normalization respectively to real and imaginary channels. They both normalize the mean value of inputs to zero, but bring different variances to the normalized values, and hence lead to different outputs. To ensure the [CLS] token is a legal quantum state, we unit-normalize the hidden vector of the [CLS] token and apply complex-LN to the remaining tokens.

4.3 NETWORK TRAINING

Optimization. Most recent works (Li et al., 2019; Yang et al., 2020; Tay et al., 2019) implement complex-valued NNs with double-sized real networks, and apply classical backpropagation to update their real and imaginary parts simultaneously. However, because of non-holomorphic functions (Hirose, 2003), this can yield wrong gradients of complex weights, and the effectiveness metrics of a complex-valued NN may not reflect its true performance. Therefore, we use the Wirtinger Calculus (Kreutz-Delgado, 2009) to update complex-valued parameters, which explicitly computes the gradient with respect to each complex weight. We are the first to adapt AdamW (Loshchilov & Hutter, 2017), the most popular optimizer, for complex weights. AdamW computes the second raw moment (i.e., uncentered variance) of the gradient by averaging the squared gradients in the real case. In the complex domain, however, the variance should be computed by *multiplying the gradient with its conjugate*. We modify AdamW accordingly to fix this problem and get the correct second raw moment for complex gradients. We highlight the difference between the real and complex AdamW optimizers in Alg.1.

Weight Initialization. By default, we initialize the real and imaginary parts of complex weights with a normal distribution at a mean value of zero and a variance of 0.01.

Algorithm 1 AdamW for real numbers and AdamW for for complex numbers

1: **Given** $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $\lambda \in \mathbb{R}$
2: **Initialize** time step $t \leftarrow \mathbf{0}$, parameter $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, parameter $\boldsymbol{\theta}_{t=0} \in \mathbb{C}^n$, first moment $\mathbf{m}_{t=0} \leftarrow \mathbf{0}$, second moment $\mathbf{v}_{t=0} \leftarrow \mathbf{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
3: **while** stopping criteria is not met **do**
4: $\quad t \leftarrow t + 1$
5: $\quad \bigtriangledown f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$
6: $\quad \mathbf{g}_t \leftarrow f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$
7: $\quad \mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t$
8: $\quad \mathbf{v}_t \leftarrow \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)\mathbf{g}_t \odot \mathbf{g}_t$
9: $\quad \mathbf{v}_t \leftarrow \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)\mathbf{g}_t \odot \bar{\mathbf{g}}_t$
10: $\quad \hat{\mathbf{m}}_t \leftarrow \mathbf{m}_t/(1 - \beta_1^t)$
11: $\quad \hat{\mathbf{v}}_t \leftarrow \mathbf{v}_t/(1 - \beta_2^t)$
12: $\quad \eta_t \leftarrow \text{SetScheduleMultiplier}(t)$
13: $\quad \boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t(\alpha \hat{\mathbf{m}}_t/(\sqrt{\hat{\mathbf{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1})$
14: **end while**
15: **Return** $\boldsymbol{\theta}_t$

4.4 Measurement as Classification

With the pre-trained quantum encoding $|\psi_{[CLS]}(\boldsymbol{x})\rangle$, we train a task-related variational measurement for text classification. The state is passed to an authentic unitary layer, parameterized by a complex-valued square matrix W as follows:

$$H = \frac{W + W^H}{2}, \quad U = e^{iH}, \tag{8}$$

where $e^{(\cdot)}$ stands for matrix exponential. The resulting U is guaranteed to be a square unitary matrix, supporting a trainable unitary transformation of [CLS] state $|\psi'_{[CLS]}(\boldsymbol{x})\rangle = U\,|\psi_{[CLS]}(\boldsymbol{x})\rangle$. Finally, $|\psi'_{[CLS]}(\boldsymbol{x})\rangle$ is measured along each basis state, and the output probability vector is linearly transformed to produce the target class labels. The variational measurement head is trained with the previous transformer layers in a fine-tuning task.

At this stage, the model operates in a hybrid classical-quantum fashion: the multi-layer Transformer encoder is executed in a classical computer to obtain the encoded state $|\psi_{[CLS]}(\boldsymbol{x})\rangle$. The state is then passed to a quantum device to compute the classification probabilities for each class. Finally, we compute the cross-entropy loss against true class labels as the loss function, and the network weights are updated accordingly with the **CAdamW** optimizer in a classical computer. Compared to QSANN (Li et al., 2022), this model only requires switching once between quantum and classical hardware, so it is far more practical for hybrid classical-quantum training.

5 Experiment

We pre-train QBERT on BOOKCORPUS and English WIKIPEDIA and evaluated them on the GLUE benchmark, following the standard practice. To examine its effectiveness, we compare it with the classical BERT (a.k.a **BERT-base**) and a complex-valued BERT (a.k.a **CVBERT-base**). **CVBERT-base** has the same settings as *QBERT-base*, except that the [CLS] token is layer normalized by *complex-LN*, and the NSP head is not replaced by quantum-compatible structures in pre-training and fine-tuning. We plot their learning curves during pre-training, and compute their effectiveness on GLUE datasets. Following the established practice, **GLUE scores** are calculated by averaging the performance values on all GLUE datasets Wang et al. (2018).

To ensure a fair comparison, we align the parameter number of all three models. The models have a 12-layer Transformer structure with 12 heads in each layer. **BERT-base** has a model dimension $d_{model} = 768$ and a hidden size of $d_{hidden} = 3072$. Since a complex-valued NN has twice the number of parameters as its real-valued counterpart, the complex-valued models **CVBERT-base** and **QBERT-base** have halved hidden dimension $d_{hidden} = 1536$ and same model dimension $d_{model} = 768$. In this way, a 768-dim unit complex vector is pre-trained as the quantum encoding $|\psi_{[CLS]}(\boldsymbol{x})\rangle$. This means that the quantum classification model can be implemented as a 10-qubit

circuit[2]. We further remove the query projection and output projection layers W^Q, W^O from all its transformer layers, and tie the input embedding lookup table with the MLM projection matrix. As shown in Tab. 1, the real, complex and quantum-compatible BERT models are comparable in size.

To empirically check the gain in representation capacity brought about by the classical-quantum transfer learning paradigm, we compare the fine-tune performance of **QBERT-base** with **DisCoCat** Lorenz et al. (2021), the architecture for QNLP models. However, due to the scalability issue, **DisCoCat** can only work at a low dimensionality to be able to handle long sentences in the datasets. We follow the original setting in Lorenz et al. (2021), which uses at most 3 qubits to represent each token. We also implement two other end-to-end *quantum-like* [3] models to simulate the performance of quantum models at a comparable scale to **QBERT-base**. **QCLS-end2end** model embeds each word to a complex-valued vector and normalizes the average of word vectors as the sequence encoder. The sequence state is passed to a variational measurement to produce classification labels. The **QCLS-transformer** model has a similar structure to **QBERT-base** but is directly trained on text classification dataset with randomly initialized complex-valued token embeddings. We try $N \in \{3, 6, 12\}$ transformer layers in the **QCLS-transformer** architecture. In the absence of the source code of **QSANN** Li et al. (2022), the performance of **QCLS-transformer** can serve as an estimate of **QSANN**, since they have similar attention mechanisms.

Table 1: Results of **QBERT-base** on GLUE in comparison to classical BERT models and end-to-end QNLP models. The evaluation metric for each dataset is shown in the parentheses below the dataset name. Performance values on the development set are reported. We report the relative differences between **QBERT-base** and each model in the parentheses in the last column.

Name	d_{model}	d_{hidden}	Size	MNLI (Acc)	QNLI (Acc)	QQP (F1)	RTE (Acc)	SST (Acc)	MRPC (F1)	CoLA (Matthews Correlation)	STS (Pearson Correlation)	Avg (GLUE score)
BERT-base	768	3072	133.54M	82.1/83.0	86.6	88.8	64.6	90.1	85.6	46.5	87.9	79.8 (+4.5%)
CVBERT-base	768	1536	135.10M	81.6/81.7	86.2	88.1	63.3	90.7	88.2	52.8	88.5	80.1 (+4.9%)
QBERT-base	768	1536	135.06M	78.9/79.7	84.3	86.9	60.3	89.0	80.0	41.9	84.8	76.2 (+0.0%)
QCLS-end2end	768	1536	48.07M	40.6/41.6	65.9	67.6	48.0	80.5	74.8	0.0	-4.5	47.1 (-61.8%)
QCLS-transformer-3L	768	1536	71.30M	55.0/55.1	65.8	72.6	45.1	80.6	65.8	3.3	2.4	49.8 (-53.0%)
QCLS-transformer-6L	768	1536	92.57M	57.2/57.4	65.9	73.4	52.0	81.0	65.9	7.3	11.3	50.8 (-50.0%)
QCLS-transformer-12L	768	1536	135.06M	59.6/59.8	66.6	74.4	48.0	80.0	74.1	13.4	13.3	51.4 (-48.2%)
DisCoCat Lorenz et al. (2021)								50.9		-2.0		

DisCoCat is implemented with lambeq Kartsaklis et al. (2021a), an open-source, modular, extensible high-level Python library for experimental Quantum Natural Language Processing (QNLP). We apply **DisCoCat** to SST and CoLA, the text classification datasets in GLUE, and both training and prediction are simulated classically, with CAdamW and a batch size of 32. See App. B for the source code. The remaining models are pre-trained on 8 Tesla V100 GPU cards, at a batch size of 512 and an initial learning rate of 1e-4. Fine-tuning models are executed on a single Tesla V100 card, at a batch size of 128 and an initial learning rate of 1e-3. All models are implemented in PyTorch 1.9.0.

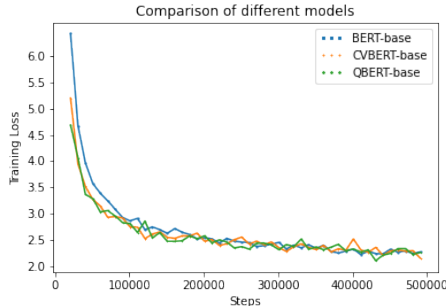

Figure 3: Learning curves of real, complex and quantum BERT models.

[2]Since the dimensionality is not a power of 2, the capacity of qubits is not fully exploited. We aim at conducting a fair comparison with the other BERT models.

[3]They are not strict quantum models, since the sentence encoding in **QBERT-base** and attention components in the **QCLS-transformers** must be implemented on a classical computer. However, they are good estimates of how a quantum model of comparable size to **QBERT-base** performs on text classification.

Overall Result. Fig. 3 presents the learning curves of the three BERT models in pre-training, while Tab. 1 compares the accuracy performance of all above-mentioned models on GLUE dataset. **QBERT-base** has a close learning curve to **CVBERT-base** and **BERT-base** and a small 4.5% and 4.9% relative drop over the two models, indicating that the quantum-compatible settings on the network implementations have little harm to the performance. More importantly, remarkable gaps in GLUE scores appear between **QBERT-base** and all end-to-end quantum classification models. By GLUE score, **QBERT-base** outperforms **QCLS-end2end** by 61.8% and **QCLS-transformer** with a minimum gap of 48.2%. Due to its low dimensionality, **DisCoCat** is even inferior to QCLS models by a large margin, and it is unfair to compare it with **QBERT-base**. However, the margin between **QBERT-base** and QCLS models does indicate the enormous benefit to the representation capacity of QNLP models brought about by the pre-trained quantum encoding.

CAdamW vs. RAdamW. We compare the complex-valued AdamW optimizer with the original AdamW optimizer (RAdamW) for real numbers. For a fair comparison, we use the two optimizers to pre-train the same complex-valued language model. As shown in Fig. 4, CAdamW converges faster than RAdamW and to a lower loss in pre-training. Therefore, the improved AdamW optimizer is indeed superior to RAdamW for training complex-valued models in practice.

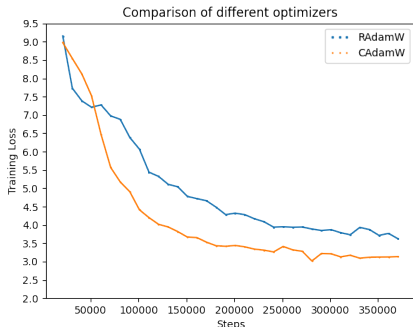

Figure 4: Learning curves of real and complex Adam optimizers to train the same complex-valued model.

Quantum Simulation. The experiment results are acquired by classical simulation. However, as demonstrated in App. A, the fine-tuning network can be converted to a quantum circuit can be implemented by the qiskit[4] toolbox, and both networks have identical behaviours. This implies that the reported values in the table can be obtained by hybrid classical-quantum training. The authentic classical-quantum hybrid simulation of QBERT is left for future work.

6 Conclusion

We have presented the first classical-quantum transfer learning scheme for quantum natural language processing (QNLP). By delicately designing the pre-trained model architecture, we managed to leverage the strong pre-trained language models for enhancing the capacity of QNLP. Empirical evaluation suggests that 50% to 60% improvement in effectiveness can be brought about by the pre-trained quantum language encoding. Besides the main finding, we proposed the first AdamW optimizer for training complex-valued BERT models, and we believe it will be beneficial for training other complex-valued neural networks.

The applicability of our model is limited in that current quantum technology cannot support authentic classical-quantum hybrid training for fine-tuning QBERT on downstream NLP tasks. However, we believe that the classical-quantum transfer mechanism and pre-trained models is necessary for scalable QNLP models, and this work has made the crucial first step by demonstrating the enormous potential of pre-trained quantum encodings. We expect that future advances on quantum technologies will make our approach feasible on real quantum computers.

[4]https://qiskit.org/

REFERENCES

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on International Conference on Machine Learning*, 2016.

Max Born. Zur Quantenmechanik der Sto\s svorgänge. *Zeitschrift für Physik*, 37(12):863–867, December 1926. ISSN 0044-3328. doi: 10.1007/BF01397477.

Yiwei Chen, Yu Pan, and Daoyi Dong. Quantum language model with entanglement embedding for question answering. *IEEE Transactions on Cybernetics*, pp. 1–12, 2021. doi: 10.1109/TCYB.2021.3131252.

Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen J. Clark. Mathematical foundations for a compositional distributional model of meaning. *Linguistic Analysis*, 36(1):345–384, 2010.

Bob Coecke, Giovanni de Felice, Konstantinos Meichanetzidis, and Alexis Toumi. Foundations for near-term quantum natural language processing, 2020.

Ivo Danihelka, Greg Wayne, Benigno Uria, Nal Kalchbrenner, and Alex Graves. Associative long short-term memory. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1986–1994, New York, New York, USA, 20–22 Jun 2016. PMLR.

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

George M Georgiou and Cris Koutsougeras. Complex domain backpropagation. *IEEE transactions on Circuits and systems II: analog and digital signal processing*, 39(5):330–334, 1992.

Nitzan Guberman. On complex valued convolutional neural networks. *arXiv preprint arXiv:1602.09046*, 2016.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2016. URL `https://arxiv.org/abs/1606.08415`.

Akira Hirose. *Complex-Valued Neural Networks: Theories and Applications*. World Scientific, 2003. ISBN 978-981-279-118-4.

Akira Hirose. Nature of complex number and complex-valued neural networks. *Frontiers of Electrical and Electronic Engineering in China*, 6(1):171–180, 2011.

Sofiene Jerbi, Lukas J. Fiderer, Hendrik Poulsen Nautrup, Jonas M. Kübler, Hans J. Briegel, and Vedran Dunjko. Quantum machine learning beyond kernel methods, 2021.

Dimitri Kartsaklis, Ian Fan, Richie Yeung, Anna Pearson, Robin Lorenz, Alexis Toumi, Giovanni de Felice, Konstantinos Meichanetzidis, Stephen Clark, and Bob Coecke. lambeq: An Efficient High-Level Python Library for Quantum NLP. *arXiv preprint arXiv:2110.04236*, 2021a.

Dimitri Kartsaklis, Ian Fan, Richie Yeung, Anna Pearson, Robin Lorenz, Alexis Toumi, Giovanni de Felice, Konstantinos Meichanetzidis, Stephen Clark, and Bob Coecke. lambeq: An Efficient High-Level Python Library for Quantum NLP. Technical Report arXiv:2110.04236, arXiv, October 2021b. URL `http://arxiv.org/abs/2110.04236`. arXiv:2110.04236 [quant-ph] type: article.

Ken Kreutz-Delgado. The Complex Gradient Operator and the CR-Calculus. Technical Report arXiv:0906.4835, arXiv, June 2009. URL `http://arxiv.org/abs/0906.4835`. arXiv:0906.4835 [math] type: article.

Guangxi Li, Xuanqiang Zhao, and Xin Wang. Quantum self-attention neural networks for text classification, 2022. URL `https://arxiv.org/abs/2205.05625`.

Qiuchi Li, Benyou Wang, and Massimo Melucci. Cnm: An interpretable complex-valued network for matching. 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)*, pp. 4139–4148, 2019.

Seth Lloyd, Maria Schuld, Aroosa Ijaz, Josh Izaac, and Nathan Killoran. Quantum embeddings for machine learning. *arXiv preprint arXiv:2001.03622*, 2020.

Robin Lorenz, Anna Pearson, Konstantinos Meichanetzidis, Dimitri Kartsaklis, and Bob Coecke. Qnlp in practice: Running compositional models of meaning on a quantum computer. *arXiv preprint arXiv:2102.12846*, 2021.

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.

Andrea Mari, Thomas R. Bromley, Josh Izaac, Maria Schuld, and Nathan Killoran. Transfer learning in hybrid classical-quantum neural networks. *Quantum*, 4:340, Oct 2020. ISSN 2521-327X. doi: 10.22331/q-2020-10-09-340.

Konstantinos Meichanetzidis, Stefano Gogioso, Giovanni De Felice, Nicolò Chiappori, Alexis Toumi, and Bob Coecke. Quantum natural language processing on near-term quantum computers, 2020.

Harshit Mogalapalli, Mahesh Abburi, B. Nithya, and Surya Kiran Vamsi Bandreddi. Classical–Quantum Transfer Learning for Image Classification. *SN Computer Science*, 3(1):20, October 2021. ISSN 2661-8907. doi: 10.1007/s42979-021-00888-y. URL https://doi.org/10.1007/s42979-021-00888-y.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. doi: 10.1017/CBO9780511976667.

T. Nitta. On the critical points of the complex-valued neural network. In *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP '02.*, volume 3, pp. 1099–1103, 2002.

Seunghyeok Oh, Jaeho Choi, and Joongheon Kim. A Tutorial on Quantum Convolutional Neural Networks (QCNN). Technical Report arXiv:2009.09423, arXiv, September 2020. URL http://arxiv.org/abs/2009.09423. arXiv:2009.09423 [quant-ph] type: article.

Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. Pre-trained models for natural language processing: A survey. *CoRR*, abs/2003.08271, 2020. URL https://arxiv.org/abs/2003.08271.

Alec Radford and Ilya Sutskever. Improving language understanding by generative pre-training. In *arxiv*, 2018.

Maria Schuld and Francesco Petruccione. Quantum Models as Kernel Methods. In Maria Schuld and Francesco Petruccione (eds.), *Machine Learning with Quantum Computers*, Quantum Science and Technology, pp. 217–245. Springer International Publishing, Cham, 2021. ISBN 978-3-030-83098-4. doi: 10.1007/978-3-030-83098-4_6. URL https://doi.org/10.1007/978-3-030-83098-4_6.

Yi Tay, Aston Zhang, Anh Tuan Luu, Jinfeng Rao, Shuai Zhang, Shuohang Wang, Jie Fu, and Siu Cheung Hui. Lightweight and Efficient Neural Natural Language Processing with Quaternion Networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 1494–1503, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1145. URL https://www.aclweb.org/anthology/P19-1145.

Chiheb Trabelsi, Olexa Bilaniuk, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, Joao Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J Pal. Deep complex networks. In *International Conference on Learning Representations*, 2018.

Théo Trouillon and Maximilian Nickel. Complex and holographic embeddings of knowledge graphs: A comparison. *CoRR*, abs/1707.01475, 2017.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Eric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2071–2080, New York, New York, USA, 20–22 Jun 2016. PMLR.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 353–355, 2018.

Benyou Wang, Qiuchi Li, Massimo Melucci, and Dawei Song. Semantic hilbert space for text representation learning. In *The World Wide Web Conference*, pp. 3293–3299, 2019.

Benyou Wang, Donghao Zhao, Christina Lioma, Qiuchi Li, Peng Zhang, and Jakob Grue Simonsen. Encoding word order in complex embeddings. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Scott Wisdom, Thomas Powers, John R. Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 4887–4895, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Moritz Wolter and Angela Yao. Complex gated recurrent neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Liyao Xiang, Hao Zhang, Haotian Ma, Yifan Zhang, Jie Ren, and Quanshi Zhang. Interpretable complex-valued neural networks for privacy protection. In *Eighth International Conference on Learning Representations*, 2020.

Muqiao Yang, Martin Q. Ma, Dongyu Li, Yao-Hung Hubert Tsai, and Ruslan Salakhutdinov. Complex transformer: A framework for modeling complex-valued sequence. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4232–4236, 2020.

Eric C Yeats, Yiran Chen, and Hai Li. Improving gradient regularization using complex-valued neural networks. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 11953–11963. PMLR, 18–24 Jul 2021.

William Zeng and Bob Coecke. Quantum algorithms for compositional natural language processing. *Electronic Proceedings in Theoretical Computer Science*, 221:67–75, Aug 2016. ISSN 2075-2180. doi: 10.4204/eptcs.221.8.

Aston Zhang, Yi Tay, Shuai Zhang, Alvin Chan, Anh Tuan Luu, Siu Cheung Hui, and Jie Fu. Beyond Fully-Connected Layers with Quaternions: Parameterization of Hypercomplex Multiplications with $1/n$ Parameters. *arXiv:2102.08597 [cs]*, February 2021. URL `http://arxiv.org/abs/2102.08597`. arXiv: 2102.08597.

A DEMONSTRATION OF IMPLEMENTING THE QBERT FINE-TUNING STRUCTURE WITH A QUANTUM CIRCUIT

We examine whether our QBERT fine-tuning structure can be implemented with a quantum circuit. First, we build the fine-tuning network in PyTorch, which contains a unitary layer, a measurement layer and a linear projection layer. We then feed the model with random pure states and take the output logits. Next, we design a quantum circuit with the network weights using the qiskit toolkit. The same input states for the classical model are taken to initialize the quantum circuit state. With the circuit design and initial state, we are able to compile the circuit and run quantum simulation. The statistics of the output state is taken over a large number of simulations. The obtained occurrences are converted to probabilities of each basis state, and the probabilities are multiplied by the classical linear transformation matrix to produce the predicted logits. We compute the differences between the logits under both networks to judge if they produce identical results.

We run a simulation with $n = 3$ qubits, and $k = 2$ classes. That means our QBERT fine-tuning structure involves an 8-dim unitary transformation layer, a measurement along the 8 basis states $\{|000\rangle, ..., |111\rangle\}$, and a linear projection with a 8-by-2 matrix to produce the logits. We randomly initialize a total number of M=16 pure states. The quantum circuit is simulated for a total number of N=100000 times. We compute the mean squared error (MSE) between the outputs under classical and quantum implementations. As a result, we obtain an MSE value of $e = 4.29 \times 10^-10$. This proves that the QBERT fine-tuning head can be precisely converted to a quantum circuit. In another word, the whole fine-tuning procedure can potentially be instrumented by a hybrid classical-quantum network, which will be explored in future works.

Below are the main body of the codes for implementing the classical and quantum networks as well as running the simulation on random examples.

```
# Quantum Implementation
class ParamCircuit(object):
    """
    This class provides a simple interface for interaction
    with the quantum circuit
    """
    def __init__(self, n_qubits,
                 backend,
                 unitary_matrix,
                 tranformation_matrix,
                 shots=1000):

        self.n_qubits = n_qubits
        self.all_qubits = [i for i in range(n_qubits)]
        self.unitary_gate = UnitaryGate(unitary_matrix)
        self.backend = backend
        self.shots = shots

    def run(self, initial_states):
        all_probs = []
        for state in initial_states:
            self._circuit = qiskit.QuantumCircuit(self.n_qubits)
           self._circuit.initialize(state, self._circuit.qubits)
           self._circuit.append(self.unitary_gate,self.all_qubits)
            self._circuit.measure_all()
            t_qc = transpile(self._circuit,
                             self.backend)
            qobj = assemble(t_qc,
                    shots=self.shots)
            job = self.backend.run(qobj)
            result = job.result().get_counts()
            counts = [result[k] for k in sorted(result)]
            # Compute probabilities for each state
            probs = np.array(counts) / self.shots
            all_probs.append(probs)
        # Get state expectation
```

```
        return all_probs

# Classical Implementation
from torch.nn import CrossEntropyLoss, MSELoss

import torch.nn as nn
from layers.bert_dense_factory import Unitary
class QBertCLSHead(nn.Module):
    def __init__(self, dim,num_labels=2):
        super(QBertCLSHead, self).__init__()
        self.num_labels = num_labels
        self.dim=dim
        self.pooler = Unitary(self.dim,
                    init_mode='normal',
                         real_weight=False)

        self.classifier = nn.Linear(self.dim, self.num_labels, bias=False
                                                )
        self.pooler.init_params(std=0.001)
        nn.init.normal_(self.classifier.weight, mean=0, std=0.001)

    def forward(self, input_state, labels=None):

        pooled_output = self.pooler(input_state)

        logits = self.classifier(pooled_output.abs().pow(2))

        if labels is not None:
            if self.num_labels == 1:
                #  We are doing regression
                loss_fct = MSELoss()
                loss = loss_fct(logits.view(-1), labels.view(-1))
            else:
                loss_fct = CrossEntropyLoss()
                # print(logits)
                loss = loss_fct(logits.view(-1, self.num_labels), labels.
                                                    view(-1))
            return loss, logits
        return logits

# Running Simulation on Toy Examples
from models.qiskit.param_circuit import ParamCircuit
from models.qiskit.qcls_head import QBertCLSHead
import qiskit
import torch
import torch.nn.functional as F
import numpy as np
from torch.nn import MSELoss
def run_simulation(qbits=3, batch_size=16, nclasses = 2,shots=1000000):

    #qbits = 5
    model_dim = pow(2,qbits)
    qbert_head = QBertCLSHead(model_dim, nclasses)
    r = torch.rand(batch_size, model_dim,dtype = torch.cdouble)
    initial_states = F.normalize(r,p=2, dim=-1)
    outputs = qbert_head(initial_states.to(torch.cfloat))
    unitary_matrix = qbert_head.pooler.compute_unitary().detach()
    backend = qiskit.Aer.get_backend('aer_simulator')
    circuit = ParamCircuit(qbits,backend,unitary_matrix,shots)
    res = circuit.run(initial_states.numpy())
    res = torch.Tensor(np.stack(res,axis=0))
    output = torch.matmul(res,qbert_head.classifier.weight.t())
    print(MSELoss()(output, outputs))
```

B CODES FOR TRAINING DISCOCAT ON SST WITH LAMBEQ

```
# -*- coding: utf-8 -*-

import torch
import pickle
import os
from discopy import Dim
from datasets import load_dataset
from lambeq import AtomicType, SpiderAnsatz, PytorchTrainer, Dataset,
                                    PytorchModel, BobcatParser

print('running discocat classification on sst...')
BATCH_SIZE = 32
EPOCHS = 10
LEARNING_RATE = 1e-3


# Otherwise there will be weird bugs in converting the diagrams to tensor
                                     networks
model_dim = 2
nb_classes = 2

SEED = 0

max_train_samples = 10000
max_val_samples = 10000

dataset = load_dataset('glue','sst2')

max_train_samples = min(max_train_samples, len(dataset['train']['sentence
                                    ']))
max_val_samples = min(max_val_samples, len(dataset['validation']['
                                    sentence']))

parser = BobcatParser(verbose='text')

train_diagrams = parser.sentences2diagrams(dataset['train']['sentence'][:
                                    max_train_samples])
val_diagrams = parser.sentences2diagrams(dataset['validation']['sentence'
                                    ][:max_val_samples])

train_labels=  [[float(y), float(1-y)] for y in dataset['train']['label']
                                    [:max_train_samples]]
val_labels = [[float(y), float(1-y)] for y in dataset['validation']['
                                    label'][:max_val_samples]]

max_train_samples = min(max_train_samples, len(train_diagrams))
max_val_samples = min(max_val_samples, len(val_diagrams))

new_train_diagrams = []
new_train_labels = []
for i, (diagram, label) in enumerate(zip(train_diagrams, train_labels)):
    if i >=max_train_samples:
        break
    # Some sentences are not grammatically correct,
    # and the parser will return None result
    if diagram is not None:
        new_train_diagrams.append(diagram)
        new_train_labels.append(label)
del train_diagrams
del train_labels
```

```
new_val_diagrams = []
new_val_labels = []
for i, (diagram, label) in enumerate(zip(val_diagrams, val_labels)):
    if i >=max_val_samples:
        break
    if diagram is not None:
        new_val_diagrams.append(diagram)
        new_val_labels.append(label)

del val_diagrams
del val_labels

# Convert the diagrams to circuits
ansatz = SpiderAnsatz({AtomicType.NOUN: Dim(model_dim),
                      AtomicType.SENTENCE: Dim(nb_classes),
                       AtomicType.PREPOSITIONAL_PHRASE: Dim(model_dim)})

train_circuits = []
val_circuits = []
legal_ids = []
for i, diagram in enumerate(new_train_diagrams):
    # Some diagrams cannot be converted to circuits,
    # and None will be returned
    try:
        circuit = ansatz(diagram)
        train_circuits.append(circuit)
        legal_ids.append(i)
    except:
        continue
train_labels = [new_train_labels[x] for x in legal_ids]
legal_ids = []
for i, diagram in enumerate(new_val_diagrams):
    try:
        circuit = ansatz(diagram)
        val_circuits.append(circuit)
        legal_ids.append(i)
    except:
        continue
val_labels = [new_val_labels[x] for x in legal_ids]

all_circuits = train_circuits + val_circuits
model = PytorchModel.from_diagrams(all_circuits)

def accuracy(y_hat, y):
    return sum(torch.argmax(y_hat,dim=-1) == torch.argmax(y, dim=-1))/len
        (y)

eval_metrics = {"acc": accuracy}

# Create Trainer and datasets from the pre-processed circuits
trainer = PytorchTrainer(
        model=model,
        loss_function=torch.nn.CrossEntropyLoss(),
        optimizer=torch.optim.AdamW,
        learning_rate=LEARNING_RATE,
        epochs=EPOCHS,
        evaluate_functions=eval_metrics,
        evaluate_on_train=True,
        verbose='text',
        seed=SEED)


print('creating dataset...')
train_dataset = Dataset(
            train_circuits,
```

```
            train_labels,
            batch_size=BATCH_SIZE)

val_dataset = Dataset(val_circuits, val_labels, shuffle=False)

# Training the model
print('training...')
trainer.fit(train_dataset, val_dataset, evaluation_step=1, logging_step=1
                                     )


import matplotlib.pyplot as plt
fig1, ((ax_tl, ax_tr), (ax_bl, ax_br)) = plt.subplots(2, 2, sharey='row',
                                      figsize=(10, 6))

ax_tl.set_title('Training set')
ax_tr.set_title('Development set')
ax_bl.set_xlabel('Epochs')
ax_br.set_xlabel('Epochs')
ax_bl.set_ylabel('Accuracy')
ax_tl.set_ylabel('Loss')

colours = iter(plt.rcParams['axes.prop_cycle'].by_key()['color'])
ax_tl.plot(trainer.train_epoch_costs, color=next(colours))
ax_bl.plot(trainer.train_results['acc'], color=next(colours))
ax_tr.plot(trainer.val_costs, color=next(colours))
ax_br.plot(trainer.val_results['acc'], color=next(colours))

# print validation accuracy
test_output = model(val_circuits)
test_labels = torch.tensor(val_labels)
if type(test_output) == tuple:
    test_output, legal_idx = test_output
    test_labels = test_labels[legal_idx]
test_acc = accuracy(test_output, test_labels)
with open('acc_discocat_sst2.txt','w') as writer:
    writer.write(str(test_acc.item()))
```