Deepwrap

https://github.com/nageshsinghc4/deepwrap

**Deepwrap**, a low-code Python library that makes deep learning more accessible and easier to apply. As a wrapper to TensorFlow/Keras and many other libraries (e.g., transformers, scikit-learn etc), it is designed to make sophisticated, state-of-the-art deep learning models simple to build, train, inspect, and apply by both beginners and experienced practitioners. Featuring modules that support text data (e.g., text classification, sequence tagging, open-domain question-answering), vision data (e.g., image classification) and tabular data, **deepwrap** presents a simple unified interface enabling one to quickly solve a wide range of tasks in as little as three or four “commands” or lines of code.

**1. Introduction**

Deep learning workflows can be quite involved and challenging for newcomers to master. Consider the following steps.

**1) Model-Building**. The training data may reside in a number of different formats from files in folders to CSVs or pandas dataframes. If the data is large, it must be wrapped in a generator. Data must be pre-processed in specific ways depending on different factors such as the language of training texts (e.g., English vs. Chinese) and whether or not transfer learning is being employed. Learning rates, learning rate schedules, number of epochs, weight decay, and many other hyperparameters and settings must be selected or implemented.

**2) Model-Inspection.** Once trained, a model is inspected in terms of both its successes and failures. This may include classification reports on validation performance, easily identifying examples that the model is getting the most wrong, and Explainable AI methods to understand why mistakes were made.

**3) Model-Application.** Both the model and the potentially complex set of steps required to pre-process raw data into the format expected by the model must be easily saved, transferred to, and executed on new data in a production environment.

**deepwrap** is a Python library for deep learning with the goal of presenting a simple, unified interface to easily perform the above steps regardless of the type of data (i.e., text vs. images vs. graphs). Moreover, each of the three steps above can be accomplished in as little as three or four lines of code, which we refer to as “low-code” deep learning. **deepwrap** can be used with any deep learning model implemented in TensorFlow Keras (tf.keras). In addition, **deepwrap** currently includes out-of-the-box support for the following algorithms and tasks:

**Artificial Neural Network (ANN):**

* classification and regression on data stored in tables

#### **Convolutional Neural Network (CNN):**

#### Image Classification: auto-categorize images across various dimensions

#### Image Regression: predict numerical values (e.g., age of person) from photos

#### **Recurrent neural Network (RNN) and Long short-term memory (LSTM):**

#### • Text Classification: auto-categorize documents across different dimensions

#### • Text Regression: predict numerical values (e.g., prices) from textual descriptions

#### • Sequence Tagging: extract sequences of words that represent some concept of interest (e.g., Named Entity Recognition or NER)

#### • Unsupervised Topic Modelling: discover latent themes buried in large document sets

#### • Document Similarity with One-Class Learning: find and score new documents based on thematic similarity to a set of seed documents

#### • Document Recommendation: recommend or return documents that are semantically related to given text (i.e., semantic search)

#### • Text Summarization: generate short summaries of long documents

#### • Open-Domain Question-Answering: submit questions to a large text corpus and receive exact answers

Many of the tasks above allow users to either choose from a menu of state-of-the art models or employ a custom model. With respect to text classification, for example, available models include cutting-edge Transformer models like BERT (Devlin et al., 2018; Wolf et al., 2019) in addition to fast models such as fastText (Joulin et al., 2016) and NBSVM (Wang and Manning, 2012) that are amenable to being trained on a standard laptop CPU. Other features include a learning-rate-finder to estimate an optimal learning rate (Smith, 2018), easy-to-access learning rate schedules like the 1cycle policy (Smith, 2018) and Stochastic Gradient Descent with Restarts (SGDR) (Loshchilov and Hutter, 2016), state-of-the-art optimizers like AdamW (Loshchilov and Hutter, 2017), ability to easily inspect classifications through Explainable AI and other methods, and a simple prediction API for use in deployment scenarios.

**Deepwrap** is also bundled with pretrained, ready-to-use NER models for English, Chinese, and Russian. It is open-source, free to use under a permissive Apache license, and available on GitHub at: https://github.com/nageshsinghc4/deepwrap.

**2. Building Models**

Supervised learning tasks in **deepwrap** follow a standard, easy-to-use template, which we now describe.

**STEP 1: Load and Pre-process Data.**

This step involves loading data from different sources and pre-processing it in a way that is expected by the model. In the case of text, this may involve language-specific pre-processing (e.g., tokenization). In the case of images, this may involve auto-normalizing pixel values in a way that a chosen model expects. In the case of graphs, this may involve compiling attributes of nodes and links in the network (Data61, 2018). All pre-processing methods in **deepwrap** return a pre-processor instance that encapsulates all the pre-processing steps for a particular task, which can be employed when using the model to make predictions on new, unseen data.

**STEP 2: Create Model.**

Users can create and customize their own model using tf.keras or select a pre-canned model with well-chosen defaults (e.g., pretrained BERT text classifier (Devlin et al., 2018), models for sequence tagging (Lample et al., 2016), pretrained Residual Networks (He et al., 2015) for image classification). In the latter case, the model is automatically configured by inspecting the data (e.g., number of classes, multilabel vs. multi-classification). At this stage, both the model and the datasets are wrapped in a deepwrap.Learner instance, which is an abstraction to facilitate training.

**STEP 3: Estimate Learning Rate.**

Users can employ the use of a learning rate range test (Smith, 2018) to estimate the optimal learning rate given the model and data. Some models like BERT have default learning rates that work well, so this step is optional.

**STEP 4: Train Model.**

The **deepwrap** package allows one to easily try different learning rate schedules. For instance, the **fit\_onecycle** method employs a 1cycle policy (Smith, 2018). The autofit method employs a triangular learning rate schedule (Smith, 2018) with automatic early stopping and reduction of maximal learning rate upon plateau. Thus, specifying the number of epochs is optional in autofit. The fit method, when supplied with the **cycle\_len** parameter, decays the learning rate each cycle using cosine annealing. Users can easily experiment with what works best for a particular problem.

**Step 5: Evaluation**

**Step 6: Makes predictions**