NLP Neural Networks & ULMFiT

June 2019

Julius Hochmuth

Overview

- Python frameworks for neural networks (NN)
- Preprocessing for natural language processing (NLP) tasks
- A description of the neural network
- Various optimizations
- Universal Language Processing Fine Tuning (ULMFiT)

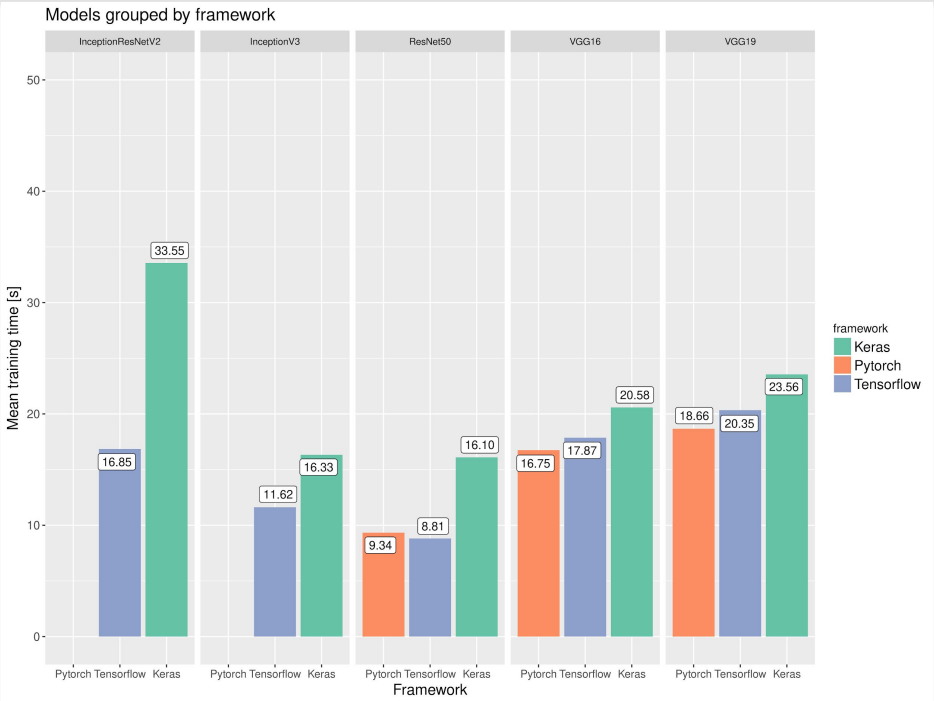
An Overview of Python NN Libraries

- Pytorch, Tensorflow, and Keras are the main python frameworks for NNs.
  - Pytorch is more “pythonic and intuitive” than tensorflow (according to the community).
  - Tensorflow is used more often, and therefore has greater community support.
  - Keras is a wrapper built on Tensorflow.
    - Makes building NNs extremely easy, but this comes at a cost to speed.
  - Keras is simpler to learn; Pytorch and Tensorflow are lower level environments which allow for greater experimentation.

```
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Simplicity of a Keras NN

Speed of Frameworks



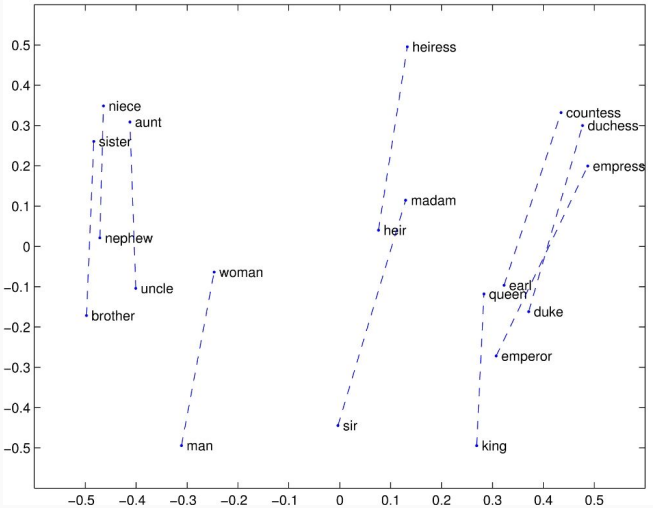
Steps to Using NNs for NLP Tasks

- Preprocess data
- Split data into folds
- For each fold:
  - Train/eval for *n* epochs
  - Generate predictions on test set
- Average predictions from each fold to get final prediction.

Preprocessing

Word Embeddings

- Represent words in n-dimensional matrices.
- Relationships of words to contexts are preserved.
- Created by various sets of models
- Pre-trained embeddings available online.
    - Different methods are used for the training of embeddings.
        - Word2vec is a predictive model
        - GloVe is a count-based model
    - Each embedding makes use of different preprocessing rules.



https://nlp.stanford.edu/projects/glove/

Word Embedding Example

A very simple example of an embedding file with vocabulary size of 4 and 3 dimensions:

Cat: [.2, .2, .2]

Feline: [.2, .2, .3]

Dog: [.7, .7, .7]

Canine: [.7, .7, .8]

Meta-Embeddings

- Meta-embeddings use multiple embeddings to improve results (ensembling technique).
- Ways to create meta-embeddings:
  - Average
  - Weighted average
  - Concatenation

Meta-embedding Research

- “We have presented an argument for averaging as a valid meta-embedding technique, and found experimental performance to be close to, or in some cases better than that of concatenation, with the additional benefit of reduced dimensionality.” (Coates and Bollegala, 2018)
- “The result seems counter-intuitive given that vector spaces in different source embeddings are not comparable and cannot be simply averaged.” (Coates and Bollegala, 2018)
- “If word embeddings can be shown to be approximately orthogonal, then averaging will approximate the same information as concatenation, without increasing the dimensionality of the embeddings.” (Coates and Bollegala, 2018)

Word Embeddings With Context

- Most words have multiple definitions, and some can be extremely different.
- One possible solution is to generate embeddings which incorporate information about the context that the word is found within.
  - Ie. use the average of the embedding for the word in question and its two neighbors.
- Sentence- and document-level embeddings are also options.

Preprocessing Text for Embeddings

- Preprocess your data to be as close to the embeddings as possible (apostrophes, digits, capitalization, etc.)
- Handling vocabulary words without a corresponding embedding:
  - If the missing word is misspelled, fix spelling.
  - Use the already-existent vector provided for a synonym of the missing word.
    - Eg. if missing an embedding for “feline”, but an embedding for “cat” is provided, use the latter as a replacement for all instances of the former.
  - Generate a new vector.
    - Create an n-dimensional matrix of 0s.
    - Create an n-dimensional matrix based on the distribution of the embedding file.

Other Preprocessing Guidelines

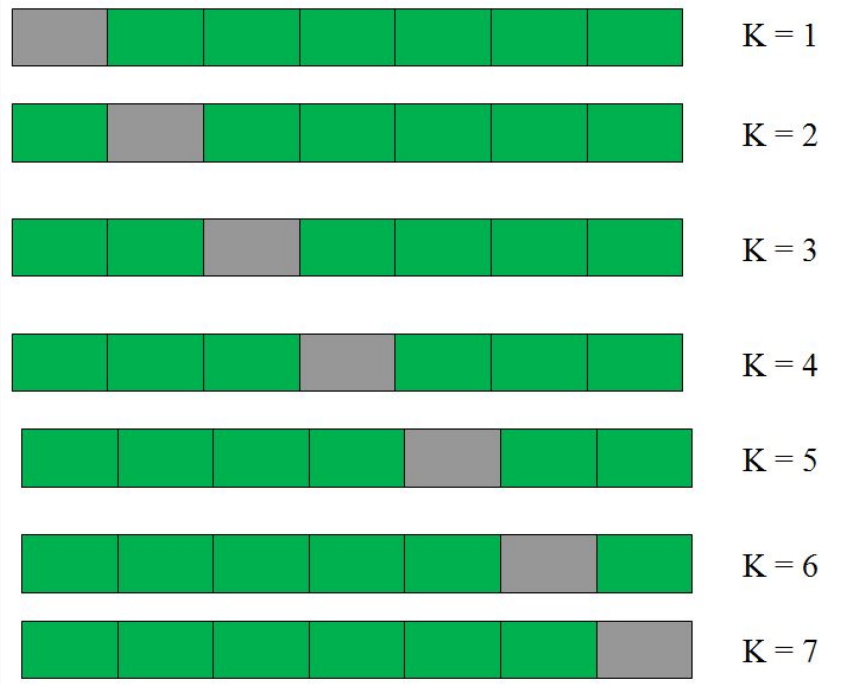
- Truncate or pad all training samples so that they are of the same length.
- Capitalization can be removed. However, if your embedding file contains a vector for the precise capitalization, you will probably want to use it because it may contain valuable information.
- Noise removal can be helpful, depending on the dataset.
- Stemming/lemmatization generally unhelpful for NNs.
- Stopword removal also not useful for NNs.

K-Fold Cross Validation

- Just like with other predictive methods, *k*-fold cross validation (KFCV) can be used to help prevent overfitting.
- The training data is divided into *k* sets (any number may be chosen, but 5/10 are common).
- A total of *k* NNs will be trained.
- During each fold, *k - 1* folds will be used as training data, and the unused fold will be used as a validation set.
    - Each fold will be used as part of the training set *k - 1* times, but only once as the validation set.
- After each NN is trained, they will be used to create predictions on the same test set.
- After all folds are completed, there will be *k* predictions for each test sample – one from each of the folds
- For each test sample, all *k* predictions are averaged together to create the final prediction.

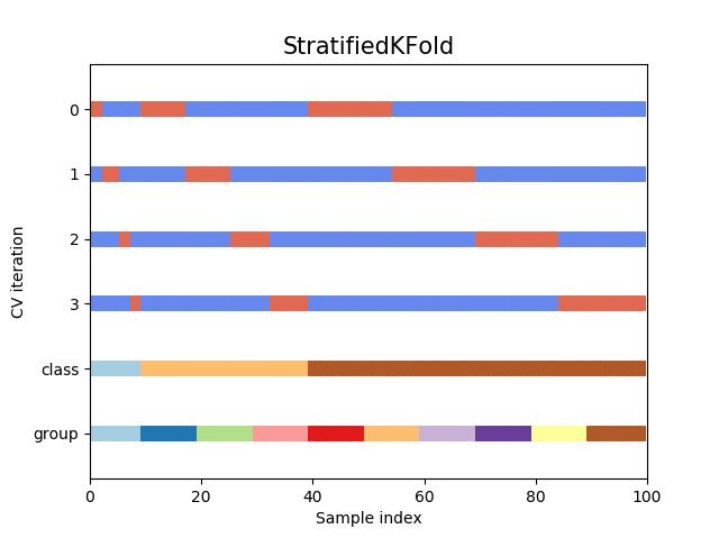
Visualization of K-Fold Cross Validation

7-folds



KFCV for Imbalanced Classes

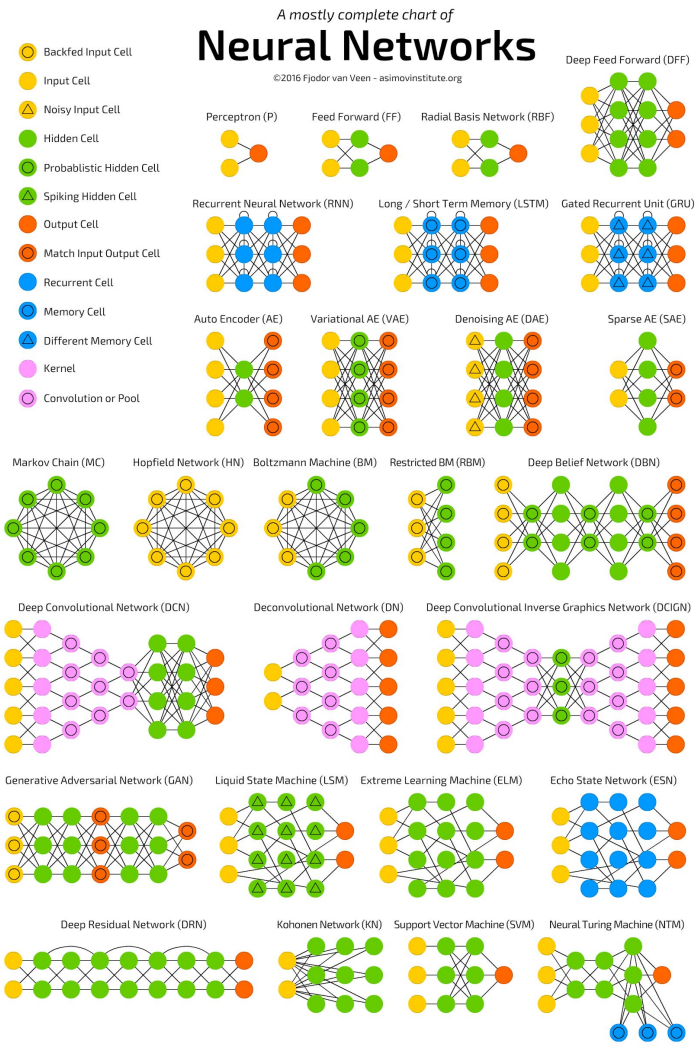
- For classification tasks, it is possible for each fold to not retain the proper ratios if no effort is made to preserve this information.
- Stratified *k*-folds can be used to preserve the ratio within each fold.



The Network

Neural Network Basics

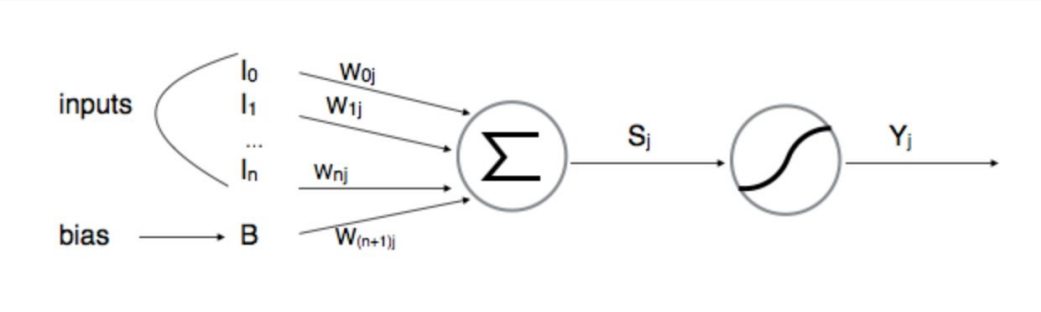
- Invented in 1958 by the US navy
  - Not useful until the past decade – AlexNet on ImageNet challenge - 2012
- Inspired by biological neural networks (ie. brains)
  - Inspired, but does not mimic!!
- An individual neural network is not an algorithm, but it will include many different algorithms
- Main types of NNs are convolutional neural networks (CNNs) and recurrent neural networks (RNNs)
  - “Generally speaking, CNNs are hierarchical and RNNs sequential architectures.” (Yin, et al., 2017)



https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

Neurons

Basic structure of a neuron:

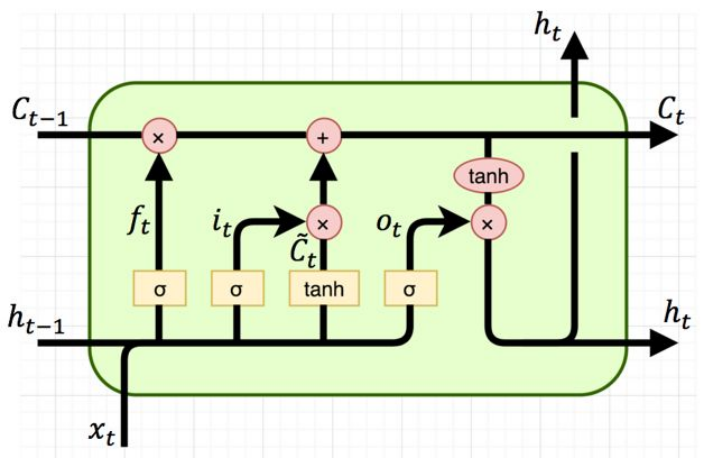


Neural Networks for NLP

- “Natural language processing has benefited greatly from the resurgence of deep neural networks, due to their high performance with less need of engineered features.” (Yin, et al., 2017)
- CNN or RNN?
  - “Which architecture performs better depends on how important it is to *semantically understand the whole sequence.*” (Yin, et al., 2017)
  - “We found that RNNs perform well and robust in a broad range of tasks except when the task is essentially a keyphrase recognition task as in some sentiment detection and question-answer matching settings.” (Yin, et al., 2017)

RNN Variations: LSTMs

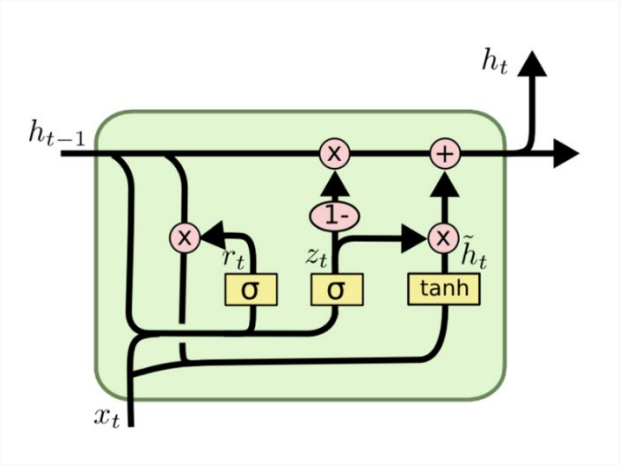
- Long short-term memory.
- “In a recurrent neural network, the gradient blows up or decays exponentially over time.” (Sundermeyer, et al., 2012)
- LSTMs were created to mitigate this problem (1997), but networks composed of LSTMs are more computationally expensive because of extra intermediate steps.
    - “Experiments suggest that the performance of standard recurrent neural network architectures can be improved by about 8% [on a language-modelling task].” (Sundermeyer, et al., 2012)



LSTM cell

RNN Variations: GRUs

- Gated recurrent unit.
- GRU units are inspired “by the LSTM unit but [are] much simpler to compute and implement.” (Cho, et al., 2014)
- While performance is mostly similar to LSTMs, there will be times when LSTMs outperform GRUs, and other times when the opposite will be true.



GRU cell

Optimization

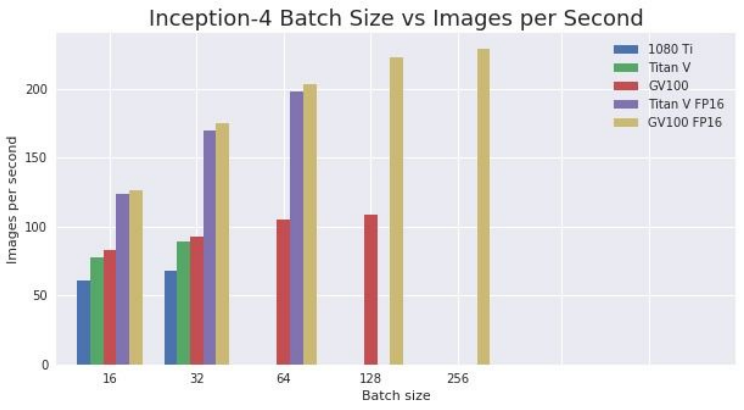
- Hyperparameter tuning: learning rate (LR), size of hidden layers, dropout, vocabulary size, batch size, etc.
- Optimizations algorithms: SGD, SGD w/ nesterov momentum, Adam, etc.
- LR schedulers:
  - Reduce LR every n epochs
  - Reduce LR on certain milestones
  - Reduce LR on plateau
  - Cyclic LR (CLR)

Epochs

- Unlike with other model types, the complete set of data is fed into the NN multiple times.
- An epoch has passed each time all of the data has been fed through the NN.
- Multiple epochs are necessary because gradient descent is an iterative process and causes the NN to slowly learn over time.
- The number of epochs is highly dependent on the dataset and the model structure.
    - Early stopping can be used when loss repeatedly does not improve for a certain number of epochs (known as patience).

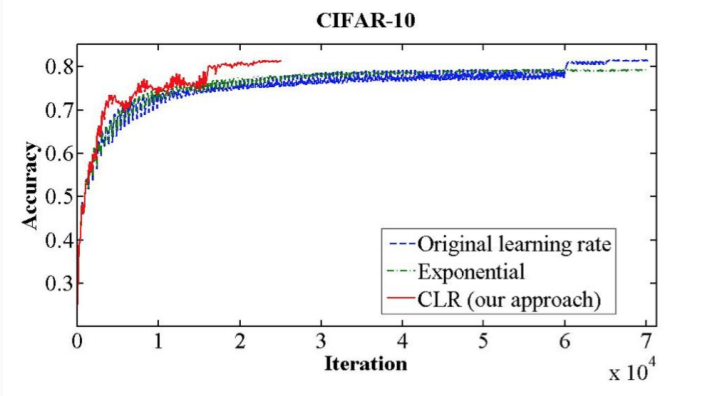
Batches

- Batch size is the number of training samples passed through the neural network before the internal parameters are updated.
- Changing the batch size does have an effect on the final accuracy of the model.
- Increasing the batch size can speed up training, but also requires more memory.
- Iterations are the number of batches per epoch.



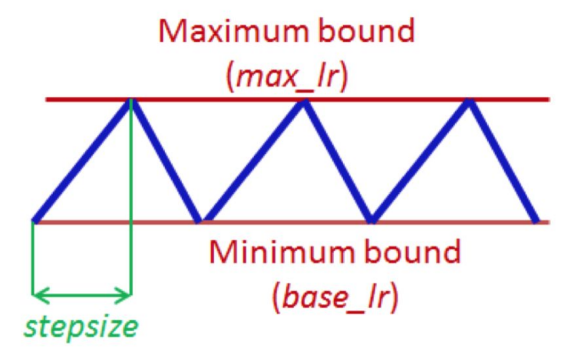
CLR

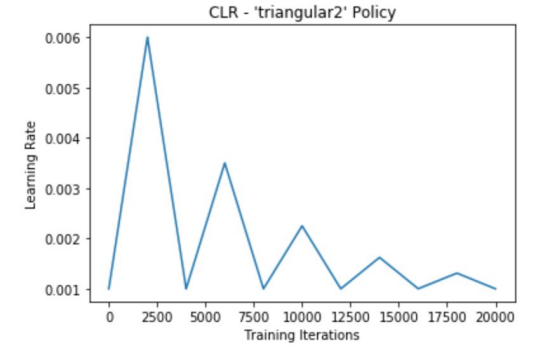
- “The essence of this learning rate policy comes from the observation that increasing the learning rate might have a short term negative effect and yet achieve a longer term beneficial effect.” (Smith, 2017)
- “It is possible to fully train the network using the CLR method instead of tuning.” (Smith, 2017)
- Can be used with adaptive learning rates.
- Increasing the learning rate can help the model “escape” from a local minima of the error.

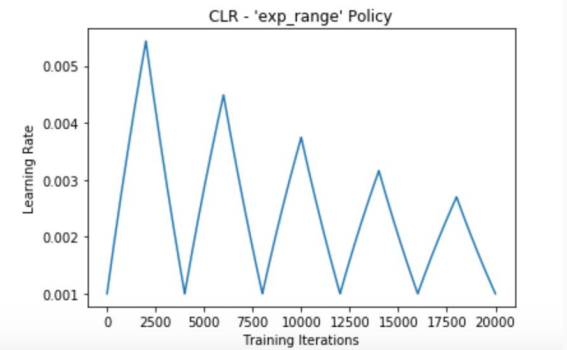


Performance of CLR vs. other strategies

CLR Shapes


Maximum bound
(max_lr)
Minimum bound
(base_lr)
stepsize


CLR - 'triangular2' Policy
Learning Rate
Training Iterations
0.001
0.002
0.003
0.004
0.005
0.006
0
2500
5000
7500
10000
12500
15000
17500
20000


CLR - 'exp_range' Policy
Learning Rate
Training Iterations
0.001
0.002
0.003
0.004
0.005
0
2500
5000
7500
10000
12500
15000
17500
20000

Miscellaneous Techniques

- Pseudo-labelling:
  - A method to artificially create more labelled data.
  - NN is trained using available labelled data, then predictions are made using unlabelled data. The NN is then retrained using labelled and pseudo-labelled data.
- Transfer learning:
  - Repurposing a model trained on one task to be used for another task.
  - Usually, more general knowledge is applied to a more specific task.
- Bidirectional RNNs:
  - A second RNN is trained on the reversed sequences.

ULMFiT

Overview

- Universal Language Model Fine Tuning state-of-the-art results for text-classification tasks. “Universal” because:
  - “It works across tasks varying in document size, number, and label type.”
  - “It uses a single architecture and training process.”
  - “It requires no custom feature engineering or preprocessing.”
  - ”It does not require additional in-domain documents or labels.” (Howard and Ruder, 2018)
- Makes use of transfer learning to improve performance. Because of this, it can be used on tasks when only a limited amount of labelled data is available: “It matches the performance of training from scratch on 100× more data.” (Howard and Ruder, 2018)
- **ULMFiT is a method rather than an architecture.**

Performance

Model	Error	Paper / Source	Code
ULMFiT (Howard and Ruder, 2018)	5.01	Universal Language Model Fine-tuning for Text Classification	Official
CNN (Johnson and Zhang, 2016) *	6.57	Supervised and Semi-Supervised Text Categorization using LSTM for Region Embeddings	Official
DPCNN (Johnson and Zhang, 2017)	6.87	Deep Pyramid Convolutional Neural Networks for Text Categorization	Official
VDCN (Alexis et al., 2016)	8.67	Very Deep Convolutional Networks for Text Classification	Non Official
Char-level CNN (Zhang et al., 2015)	9.51	Character-level Convolutional Networks for Text Classification	Non Official

AG News Dataset

Model	Error	Paper / Source	Code
ULMFiT (Howard and Ruder, 2018)	0.80	Universal Language Model Fine-tuning for Text Classification	Official
CNN (Johnson and Zhang, 2016)	0.84	Supervised and Semi-Supervised Text Categorization using LSTM for Region Embeddings	Official
DPCNN (Johnson and Zhang, 2017)	0.88	Deep Pyramid Convolutional Neural Networks for Text Categorization	Official
VDCN (Alexis et al., 2016)	1.29	Very Deep Convolutional Networks for Text Classification	Non Official
Char-level CNN (Zhang et al., 2015)	1.55	Character-level Convolutional Networks for Text Classification	Non Official

DBpedia Dataset

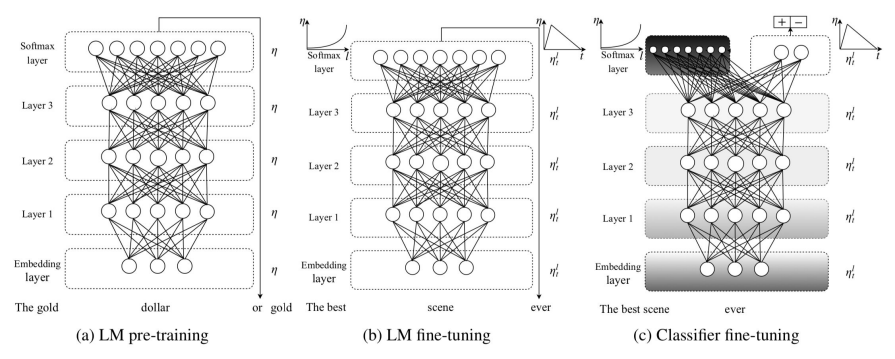
https://nlpprogress.com/english/text_classification.html

Problems Addressed by ULMFiT

- Previously, word embeddings were used as a simple way of including transfer learning.
  - Disadvantage: all other parameters in a neural network (eg. weights and biases) are randomly initialized.
- Attempts have been made previous to the creation of ULMFiT to apply generalized LMs to domain-specific tasks.
  - Disadvantage: poor performance and required large amounts of domain-specific data. Also, generalized LMs had a tendency towards catastrophic forgetting when trained on a domain-specific task.

Overview of Steps

1. Language model pretraining on general domain data (this step can be skipped if pretrained LM is used).
2. Language model fine-tuning on target data.
3. Classifier fine-tuning on target data.

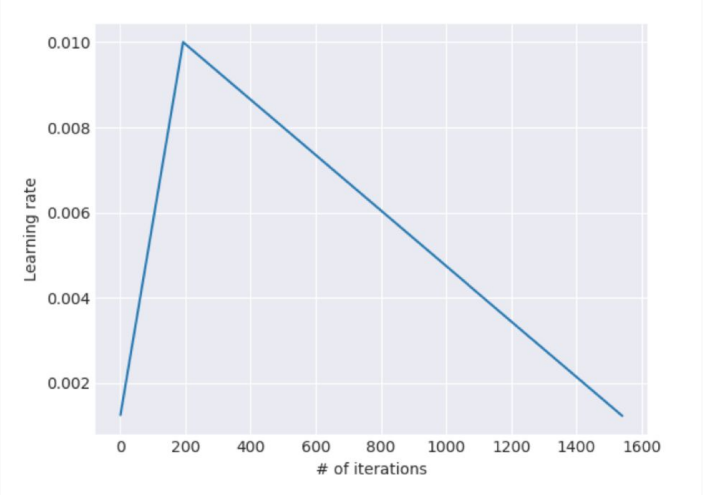


Language Model Pretraining

- Pretraining of the language models should be conducted on large datasets which “capture general properties of language.” (Howard and Ruder, 2018)
- This stage takes the most time, but pretrained models are available online.
    - These models were pretrained on the Wikitext-103 dataset, which consists of slightly less than 30,000 preprocessed wikipedia articles.
- “We leave the exploration of more diverse pretraining corpora to future work, but expect that they would boost performance.” (Howard and Ruder, 2018)
- Pretrained models are available for a variety of languages.

Language Model Fine-Tuning

- “No matter how diverse the general-domain data used for pretraining is, the data of the target task will likely come from a different distribution.” (Howard and Ruder, 2018)
- Even if the target-task dataset is small, the use of pretraining on a general corpus will offset the minimal size of the data.
- Discriminative fine-tuning:
  - “As different layers capture *different types of information*, they should be fine-tuned to *different extents*.” (Howard and Ruder, 2018)
  - Each layer of the NN is trained with a different LR.
- Slanted triangular learning rates (STLR)



Slanted Triangular LR

Classifier Fine-Tuning

- Language model is augmented with two additional linear layers.
  - First linear layer takes as input the pooled states of the last hidden layer.
- Discriminate fine-tuning and STLR are used here as well.
- Gradual unfreezing:
  - “Overly aggressive fine-tuning will cause catastrophic forgetting… too cautious fine-tuning will lead to slow convergence.” (Howard and Ruder, 2018)
  - “Gradually unfreeze the model starting from the last layer as this contains the *least general* knowledge.” (Howard and Ruder, 2018)
  - One more layer is unfrozen each epoch. All unfrozen layers are trained.

Journals Citations

Coates, J., & Bollegala, D. (2018). Frustratingly Easy Meta-Embedding – Computing Meta-Embeddings by Averaging Source Word Embeddings. *Association for Computational Linguistics*. Retrieved from https://www.aclweb.org/anthology/N18-2031.

Howard, J., & Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. *ArXiv*. Retrieved from https://arxiv.org/pdf/1801.06146v5.pdf.

Smith, L. (2017). Cyclical Learning Rates for Training Neural Networks. *ArXiv*. Retrieved from https://arxiv.org/pdf/1506.01186.pdf.

Sundermeyer, M., Schlüter, R., & Ney, H. (2012). LSTM Neural Networks for Language Modeling. *Interspeech*. Retrieved from https://www.isca-speech.org/archive/archive_papers/interspeech_2012/i12_0194.pdf.

Yin, W., Kann, K., Yu, M., & Schütze, H. (2017). Comparative Study of CNN and RNN for Natural Language Processing. *ArXiv*. Retrieved from https://arxiv.org/pdf/1702.01923.pdf.