Your Data, Your Rules: Pretraining Federated Text Models

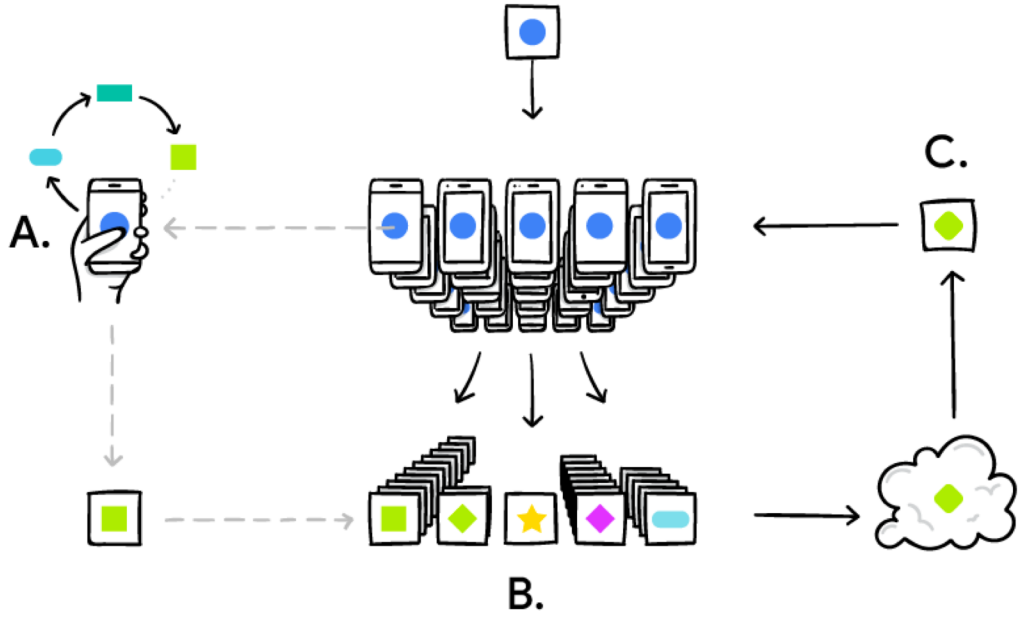
Students: Arjun Singh* & Joel Stremmel*

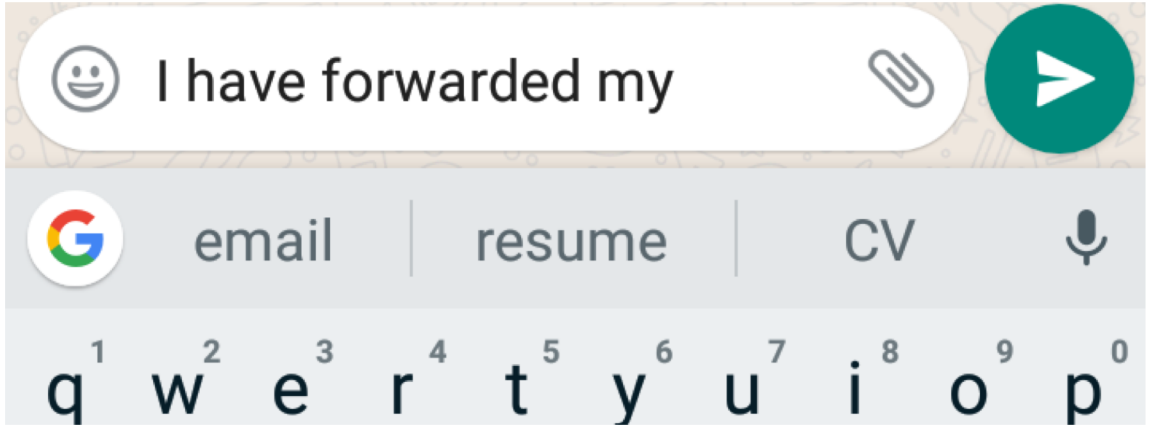
Mentors: Keith Rush and Peter Kairouz

Instructor: Megan Hazen

Federated Learning

- Federated Learning is a decentralized approach for training models on user devices, personalizing the process and ensuring privacy by summarizing local changes and only sending to the cloud a focused update from the local model
- FL trains language models on client devices without exporting sensitive user data to servers and has been successful at tasks like Next Word Prediction for mobile keyboards

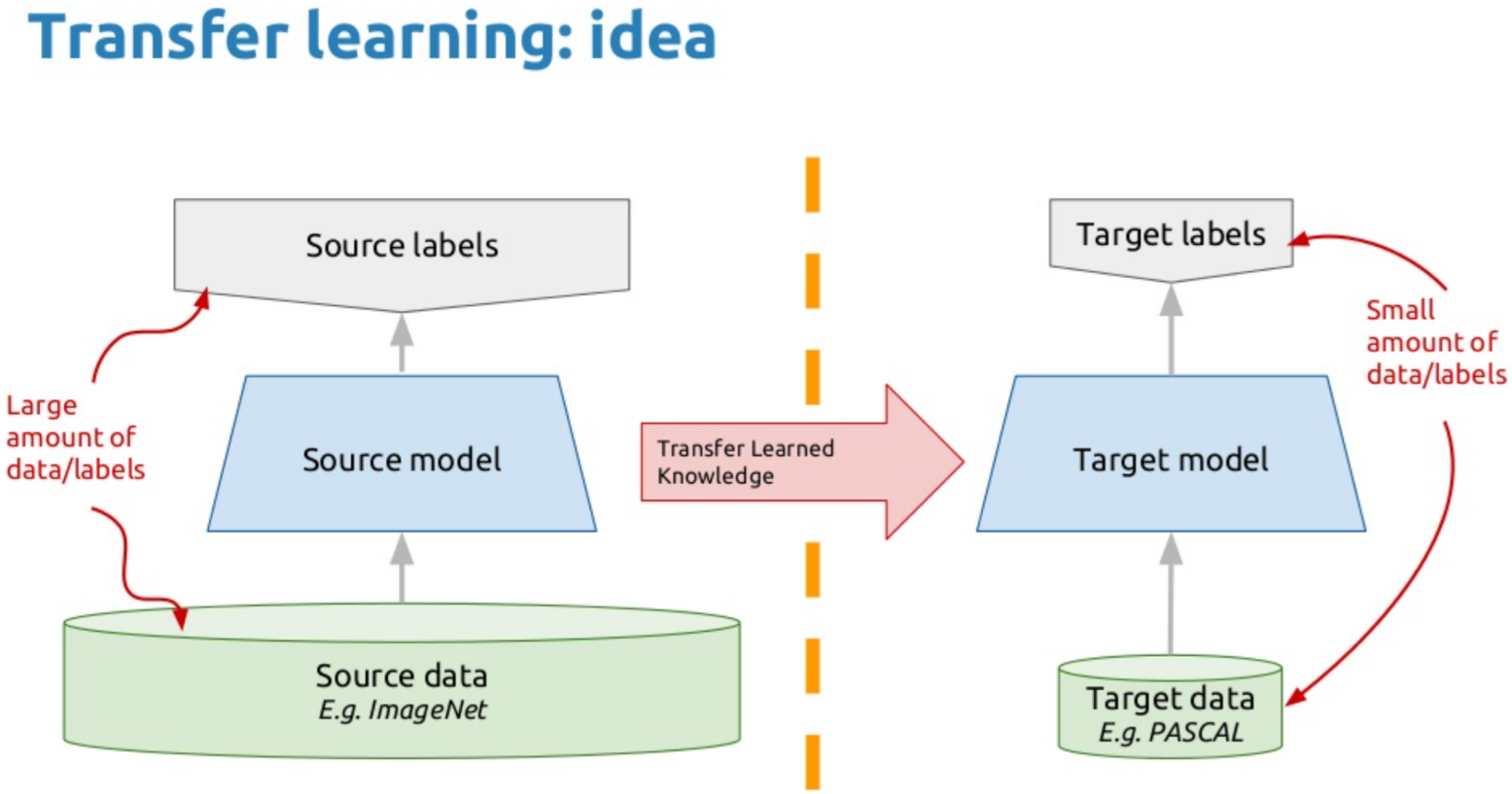




https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

Transfer Learning

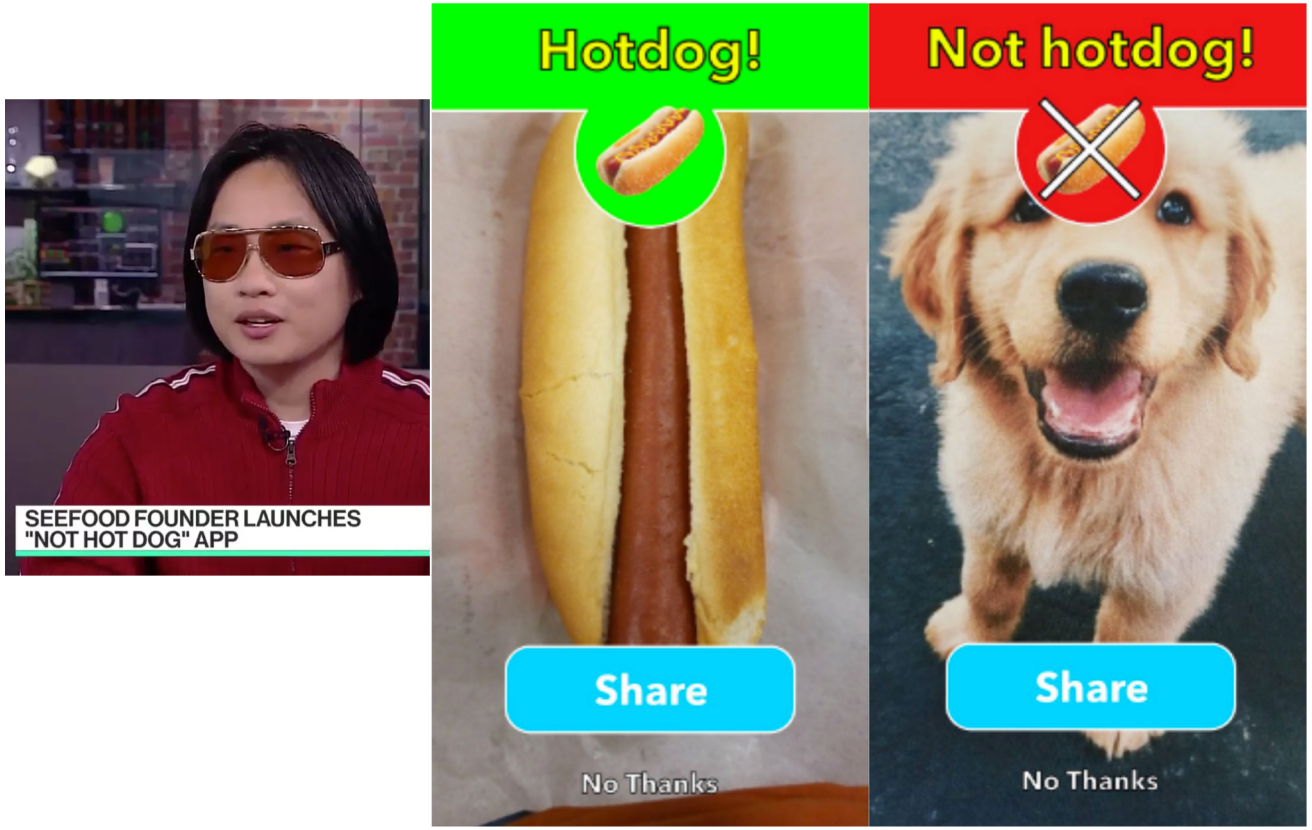
- *"Situation where what has been learned in one setting is exploited to improve generalization in another setting" - Goodfellow et al.*
- Models tasked to solve complex problems depend on copious data, but getting a ton of labelled data for supervised models can be difficult
- *"After supervised learning — Transfer Learning will be the next driver of ML commercial success" - Andrew Ng*



https://medium.com/the-official-integrate-ai-blog/transfer-learning-explained-7d275c1e34e2

Related Work

Transfer Learning for Computer Vision:
VGG-16, VGG-19, ResNet-50 etc.



https://www.oreilly.com/content/not-hotdog-vs-mission-critical-ai-applications-for-the-enterprise/
https://www.theverge.com/tldr/2017/5/14/15639784/hbo-silicon-valley-not-hotdog-app-download

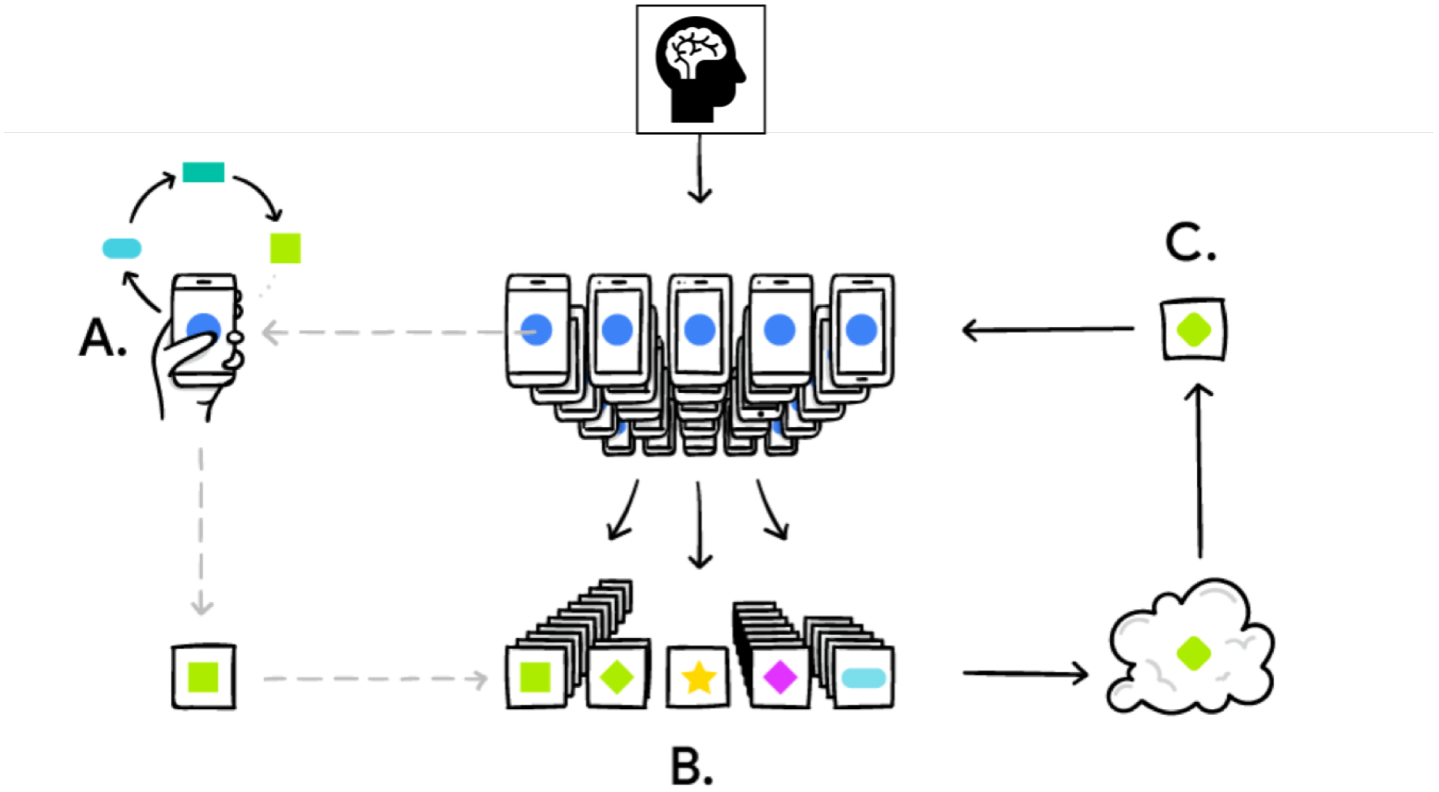
Transfer Learning for NLP:
BERT, XLNet, RoBERTa etc.

https://www.seroundtable.com/google-bert-update-28427.html

Problem Formulation: Pretrained Federated Models for Text

In this research we:

- Combine the ideas of FL and Transfer Learning to produce pretrained federated models
- Develop and enhance baseline text models using LSTMs for the task of Next Word Prediction in the federated setting
- Introduce the idea of pretrained models and pretrained word embeddings to reduce required training rounds and increase the accuracy of federated text models



https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

Methods

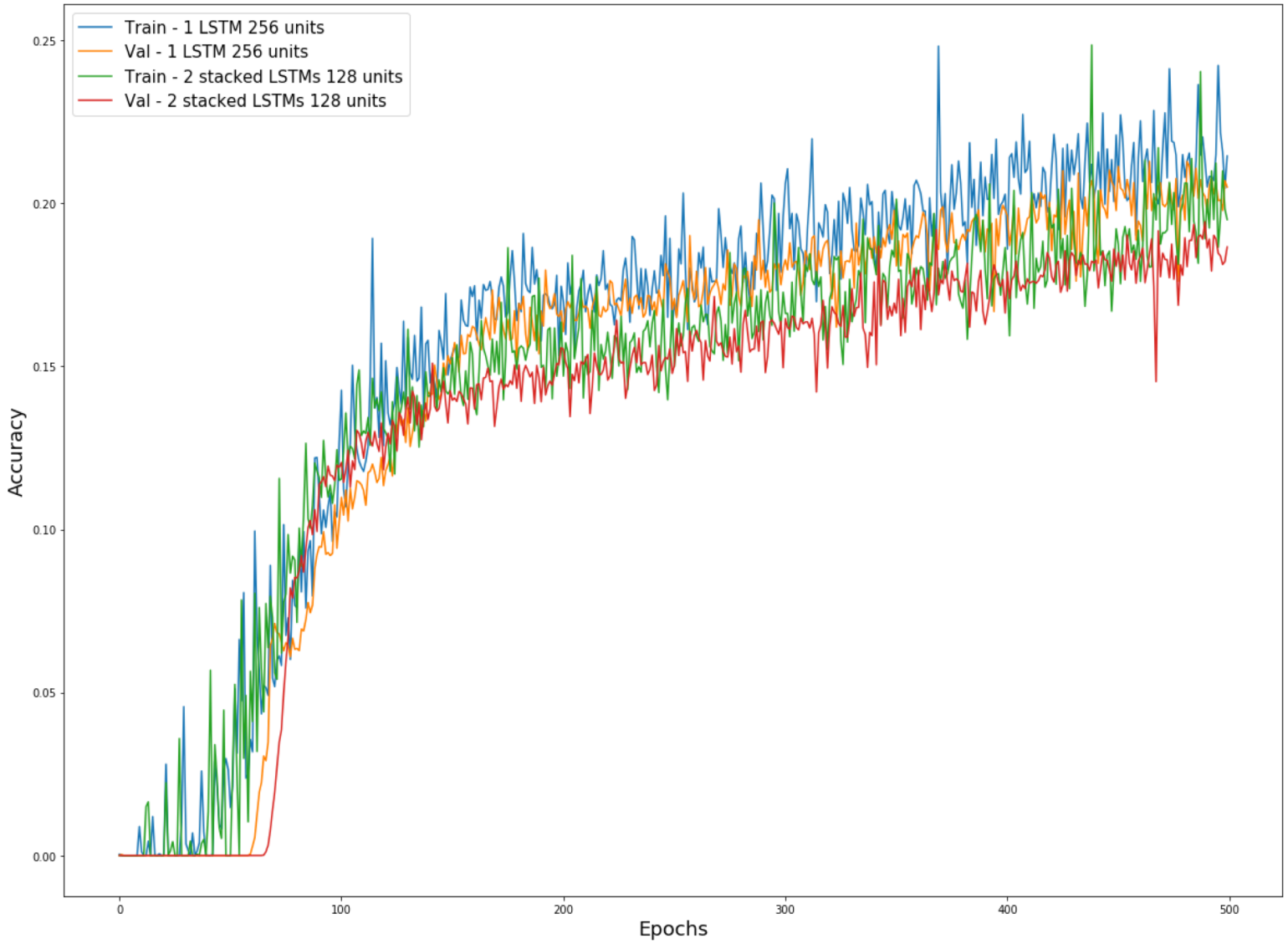
In this research we replicate a baseline network architecture for next word prediction using the Federated Averaging algorithm to train an LSTM on the Stack Overflow dataset. Our enhancements include:

- Centrally pretraining text models followed by federated fine tuning
- Incorporating pretrained word embeddings instead of randomly initialized embeddings and fine tuning these embeddings while training the full network in the federated setting
- Combining centralized pretraining and pretrained word embeddings with federated fine tuning

Experiments & Results: Deep vs Wide

- Ceteris Paribus, the model architecture was changed by replacing 1 LSTM layer having 256 units, with a 2-stacked LSTM layer with 128 units each
- The wider single LSTM network outperforms the deeper 2-stacked LSTM layer
- Difference in train time is negligible

Train and Validation Accuracy by Epoch: Small Network Deep vs Wide

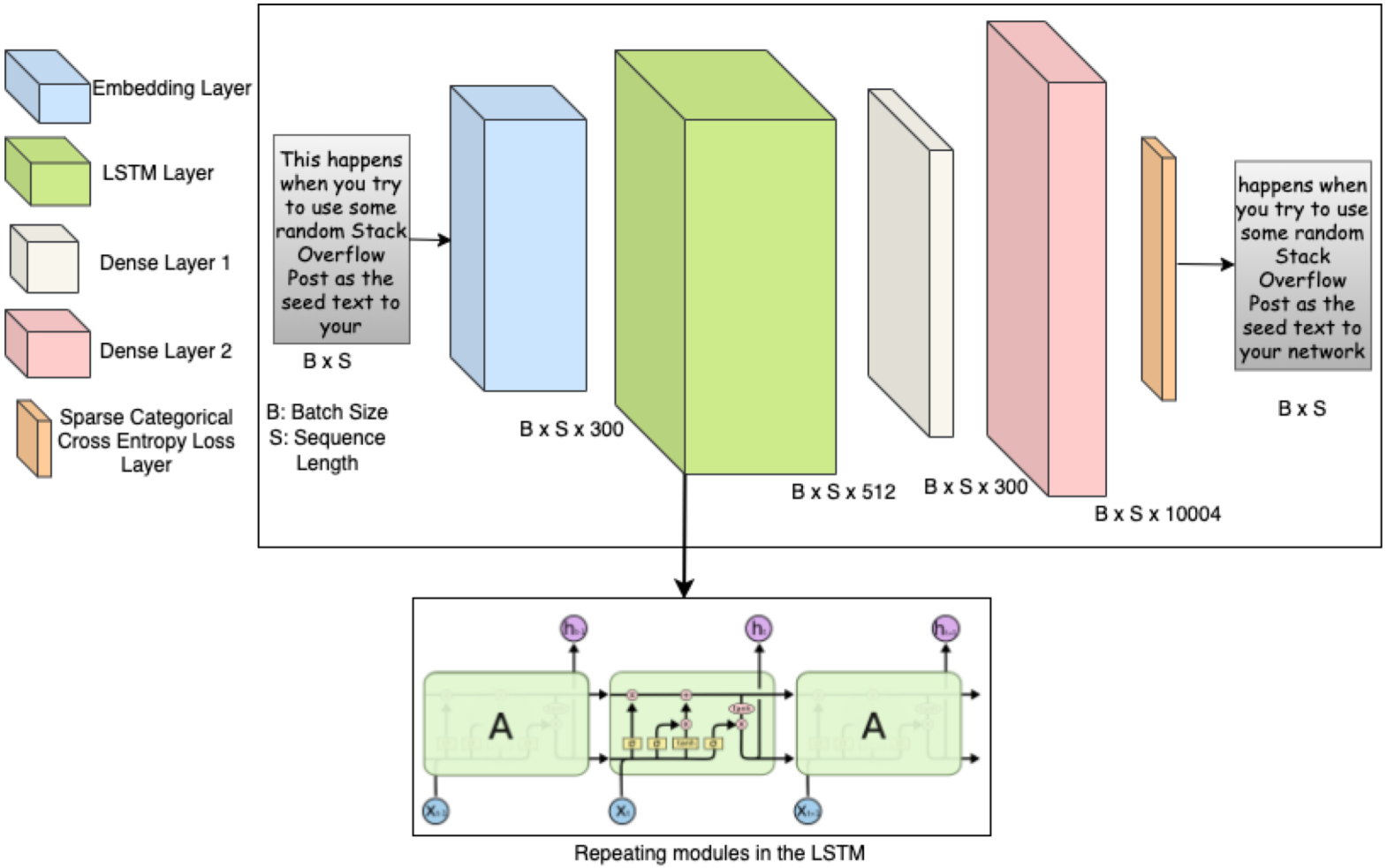


Model Architecture

- The output layer represents the top 10,000 most frequently occurring vocab words in the Stack Overflow dataset plus four special tokens: padding, beginning of a sentence, end of a sentence, and out of vocabulary
- Accuracy is reported with and without these tokens
- The model uses the Adam optimizer and Sparse Categorical Cross-entropy loss
- We compare train and validation accuracy at each training round by sampling 10 non-IID client datasets per round, with a max of 5k text samples per client with 20 tokens each, and 20k val samples

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Large Network Architecture

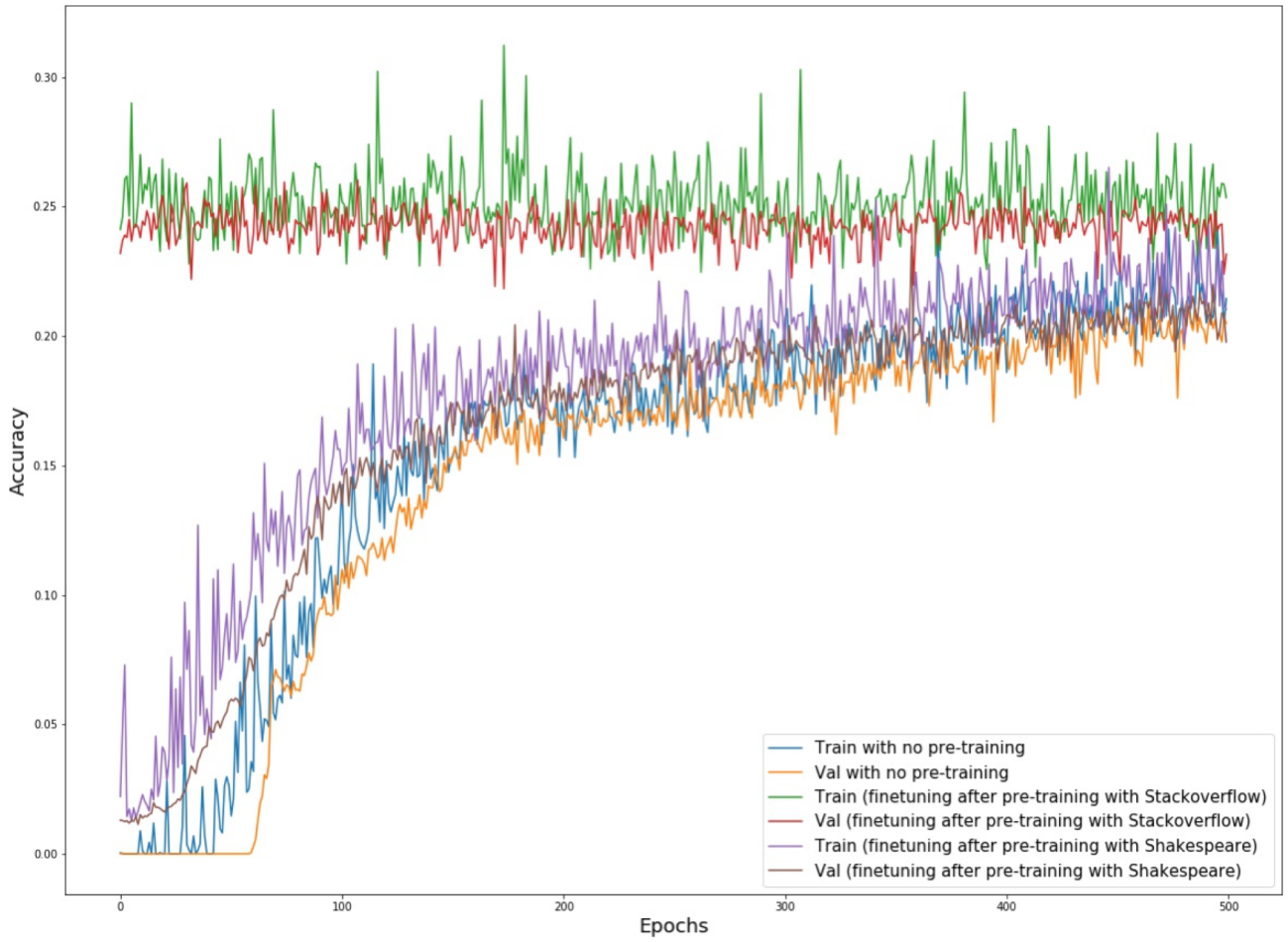


Repeating modules in the LSTM

Experiments & Results: Pretrained Models

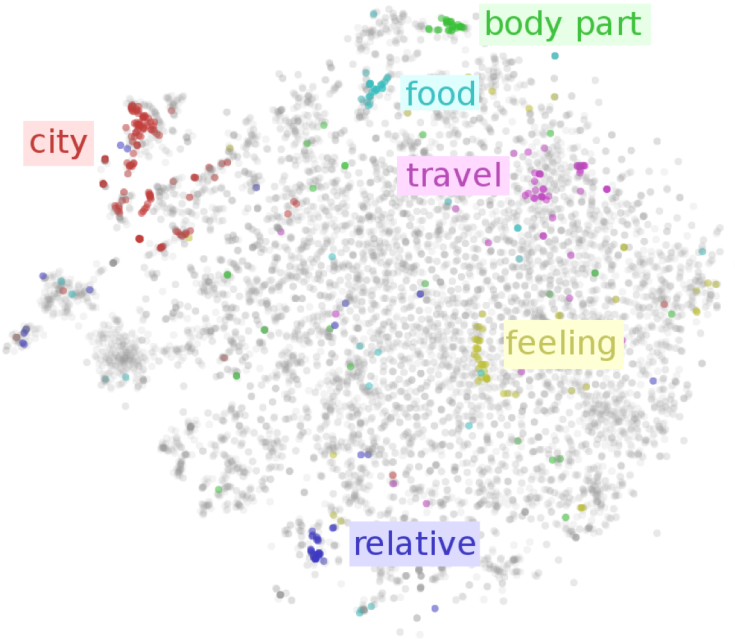
- Model centrally pretrained using (a) Shakespeare Data and (b) Stack Overflow data
- Pretrained model, fine-tuned in federated style using Stack Overflow data (distinct samples)
- Central pretraining outperforms no pretraining for both cases
- Central pretraining using SO outperforms SP
- Downstream fine-tuning adds value for SP based pretraining, but not for SO based fine-tuning

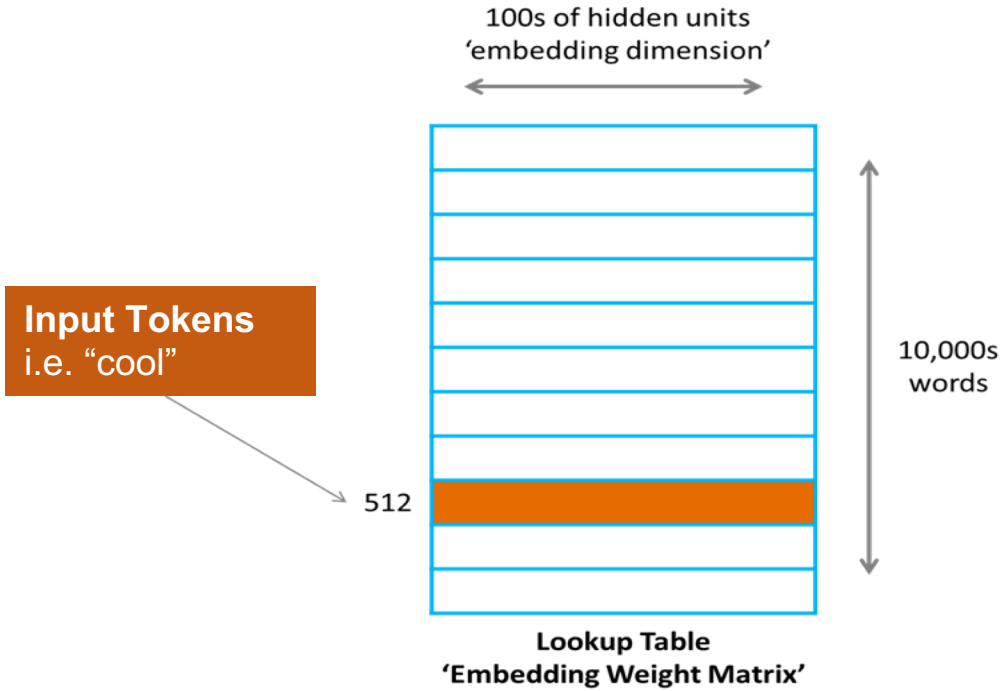
Train and Validation Accuracy by Epoch: Small Network Full Model Pretraining



Word Embeddings

- “You shall know a word by the company it keeps” - John Rupert Firth
- Word2vec and Glove are to NLP what VGGNet is to vision: intelligent weight initialization
- Large text DNNs are prohibitively large for federated training at present, but information from large networks can be captured in word embeddings





https://towardsdatascience.com/what-the-heck-is-word-embedding-b30f67f01c81

https://ruder.io/word-embeddings-1/

Word Embeddings: Dimensionality Reduction Algorithm with PCA and Post-Processing

What about really large embeddings?

Algorithm 1: Post Processing Algorithm PPA(X, D)

Data: Word Embedding Matrix X, Threshold Parameter D
Result: Post-Processed Word Embedding Matrix X

```
/* Subtract Mean Embedding                    */
X = X - X̄ ;
/* Compute PCA Components                     */
u_i = PCA(X), where i = 1,2, . . . ,d. ;
/* Remove Top-D Components                    */
for all v in X do
    v = v − Σ_{i=1}^{D} (u_i^T · v) u_i
end
```

Algorithm 2: Dimensionality Reduction Algorithm

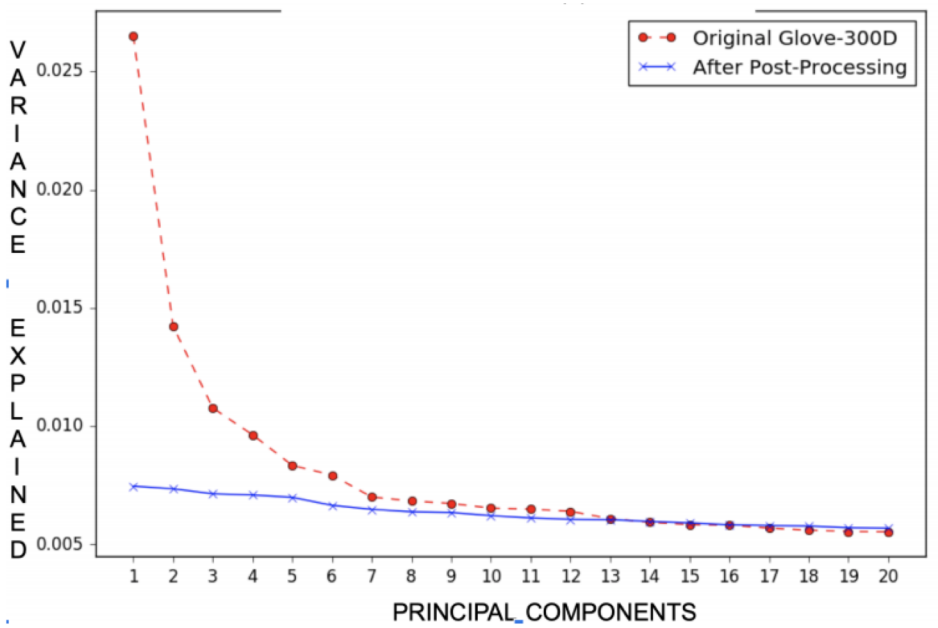
Data: Word Embedding Matrix X, New Dimension N, Threshold Parameter D
Result: Word Embedding Matrix of Reduced Dimension N: X

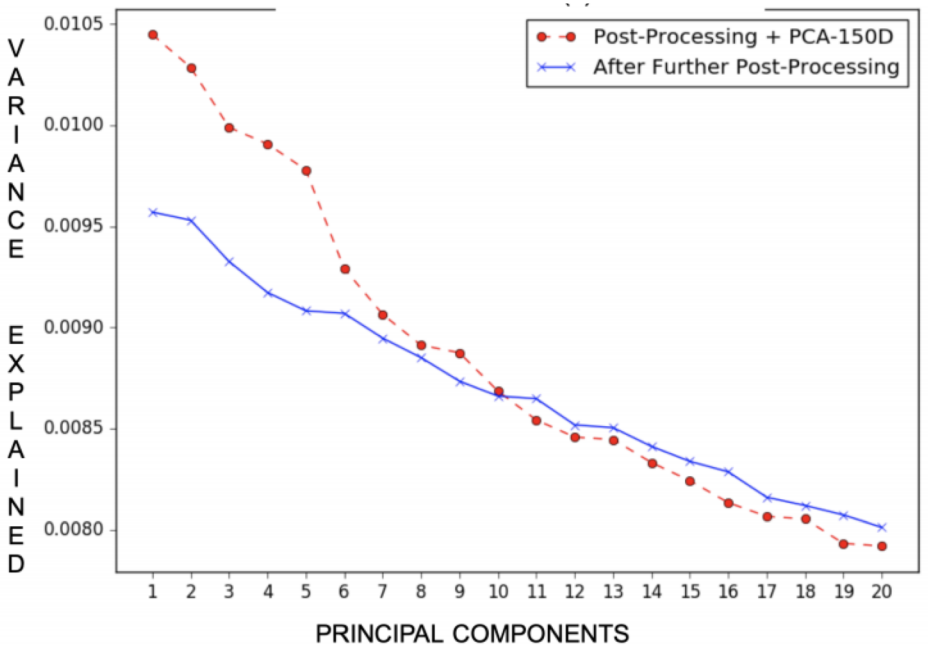
```
/* Apply Algorithm 1 (PPA)                    */
X = PPA(X, D) ;
/* Transform X using PCA                      */
X = PCA(X) ;
/* Apply Algorithm 1 (PPA)                    */
X = PPA(X, D) ;
```

Mu and Viswanath. "All-but-the-Top: Simple and Effective Postprocessing for Word Representations."
Raunak et al. "Effective Dimensionality Reduction for Word Embeddings."

Word Embeddings: Variance Explained before and after Post-Processing with Algorithms 1 & 2

- Variance explained by top 20 principal components for GloVe embeddings
- Algorithm 1 (top): subtract the mean vector from all word vectors as well as the directions of variation explained by the top D principle components
- Algorithm 2 (bottom): post-processing + PCA + post-processing
- Raunak et al. demonstrate the effectiveness of Algorithm 2 on a variety of word similarity benchmarks, achieving performance equal to or better than embeddings of twice the size across a majority of similarity tasks





Mu and Viswanath. "All-but-the-Top: Simple and Effective Postprocessing for Word Representations."
Raunak et al. "Effective Dimensionality Reduction for Word Embeddings."

Word Embeddings: Pretrained Embeddings

- GloVe from Pennington et al.:

  Learn word vectors based on the probability of word co-occurrence across a text corpus

- FastText from Bojanowski and Grave:

  Learn word vectors by summing character level n-gram vectors based on character level n-gram context (predict target n-gram given context or vice versa)

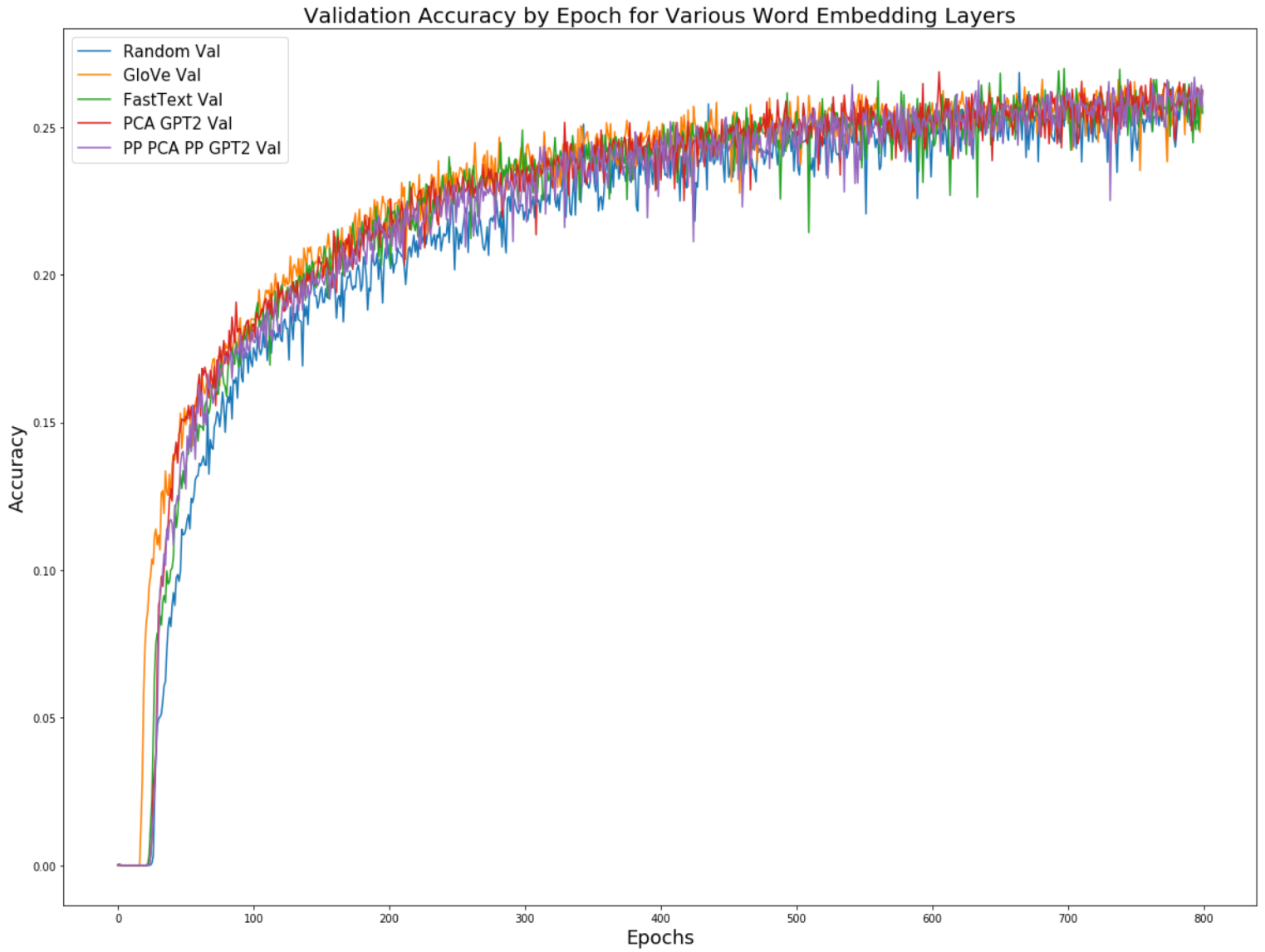
- GPT2 Radford and Wu:

  Learn word vectors from the task of next word prediction with a large Transformer architecture

Experiments & Results: Pretrained Word Embeddings

- Pretrained embeddings achieve the same level of accuracy in fewer training rounds than random embeddings
- Pretrained embeddings slightly outperform random embeddings in later training rounds, though random embeddings start to “catch up” on accuracy

Large Network Architecture



Experiments & Results: Pretrained Word Embeddings contd.

- Pretrained embeddings outperform random embeddings for the larger network architecture by over a half percent on our test set of 1m text samples
- Smaller networks demonstrate little to no increase compared to random embeddings

  - *Baseline
  - **Best
  - PP = PCA Dimensionality Reduction Post Processing Algorithm (PP + PCA + PP)

Model	Accuracy	Accuracy No OOV No EOS
Small Random*	**0.2246**	**0.1821**
Small GloVe	0.2269	0.1838
Small PCA FastText	0.2250	0.1823
Small PCA FastText + PP	0.2285	0.1852
Small PCA GPT2	0.2293	0.1859
Small PCA GPT2 + PP	0.2262	0.1834
Large Random*	**0.2485**	**0.2086**
Large GloVe	0.2557	0.2162
Large FastText	0.2548	0.2137
Large PCA GPT2	0.2522	0.2118
Large PCA GPT2 + PP**	**0.2569**	**0.2169**

Model	Parameters	Weights Size (MB)
Small	2.4M	9.6
Large	7.8M	31.3

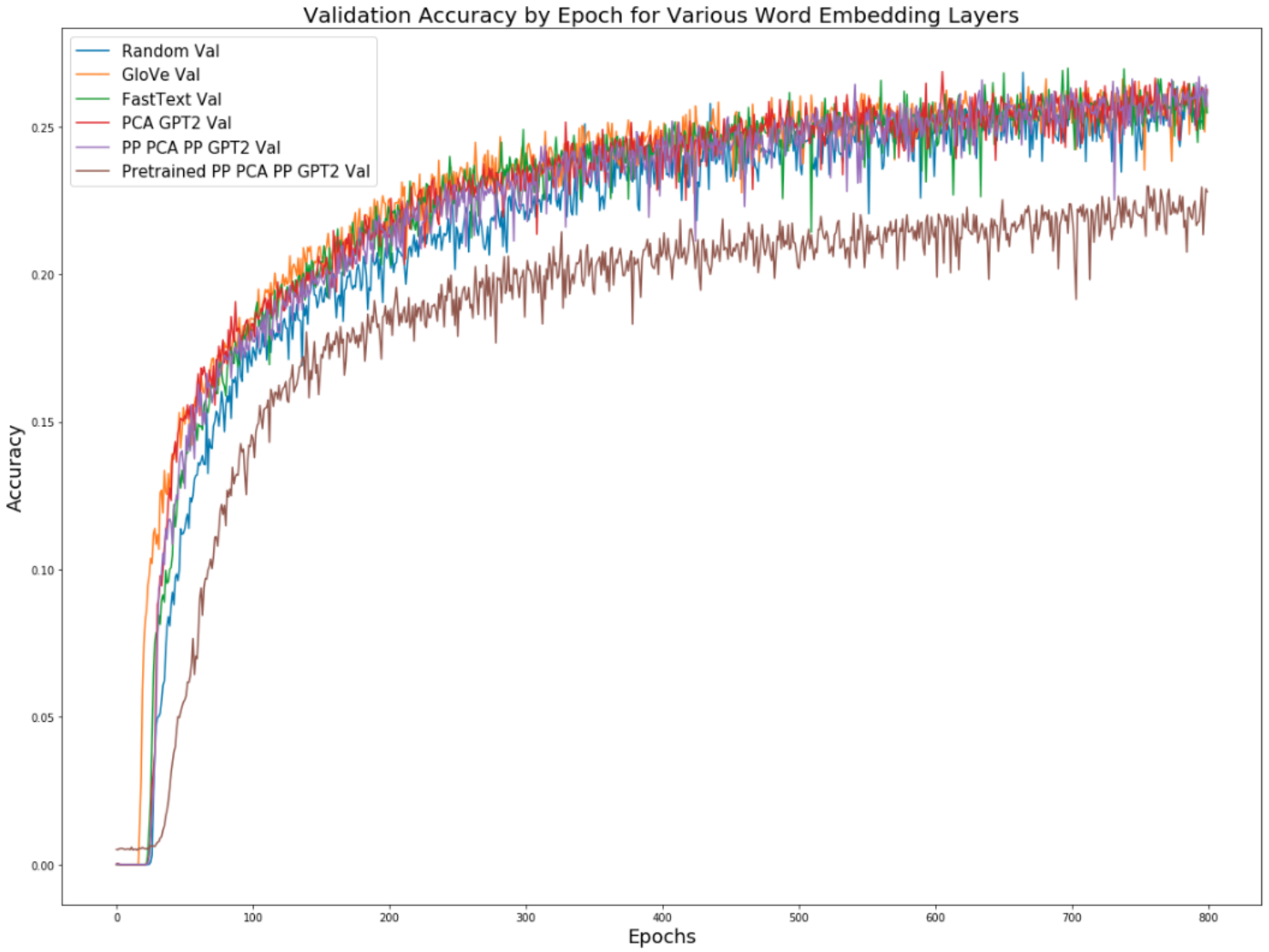
Experiments & Results: Pretrained Model + Pretrained Word Embeddings

- Our experiments demonstrate how to pretrain a model centrally and fine tune it in the federated setting, however…
- Pretraining the large network on Shakespeare and fine tuning on Stack Overflow with the best word embedding method performed worse than federated training with random and pretrained embeddings

We suspect the following would yield better results for full model pretraining:

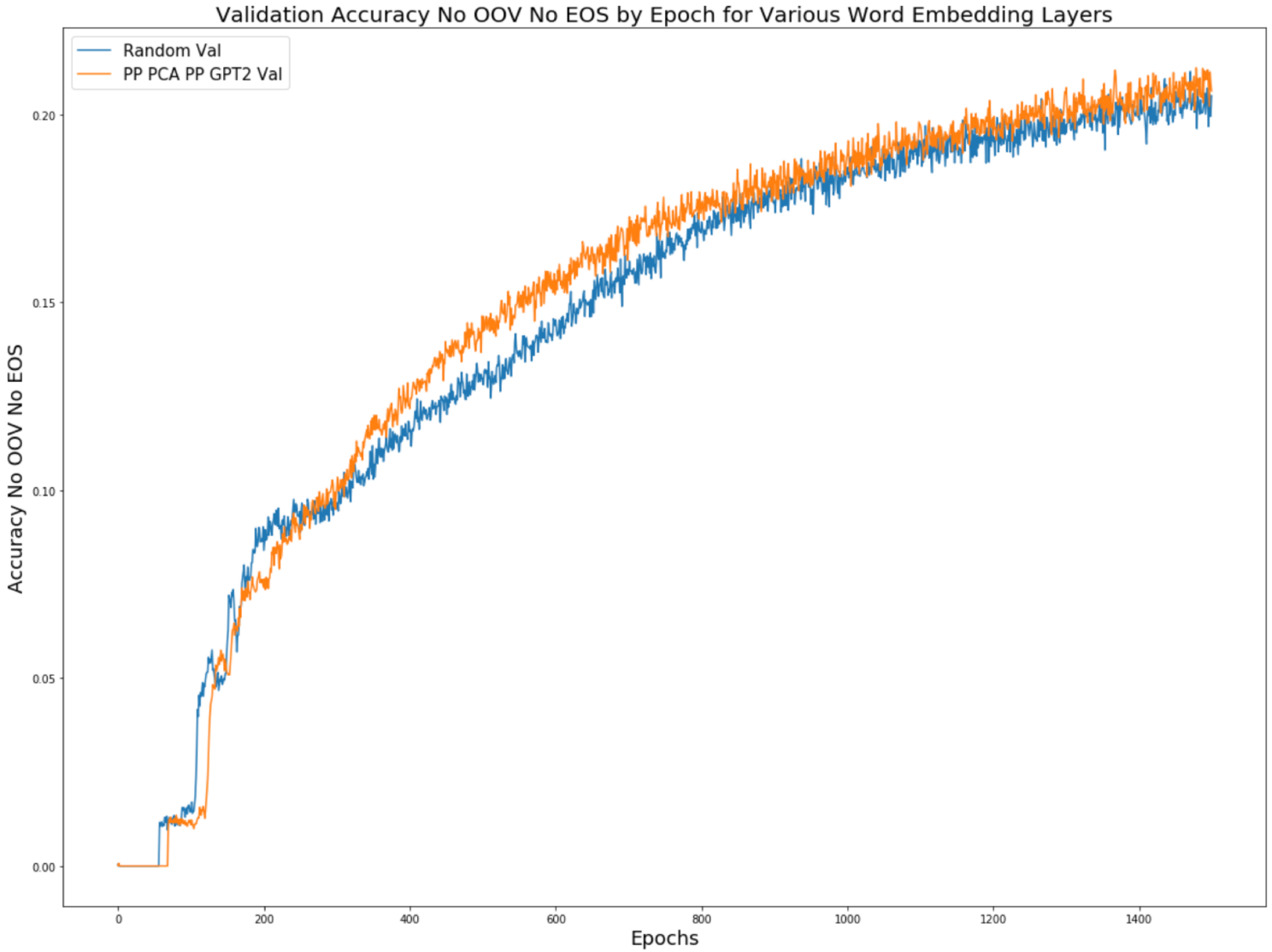
- A dataset more similar to Stack Overflow
- Learning rate optimization
- Federated instead of central pretraining

Large Network Architecture



Experiments & Results: Pretrained Word Embeddings contd.

- We compare our best embedding approach to randomly initialized word embeddings using the client sampling strategy and model architecture from "Adaptive Federated Optimization"
- We use the Adam defaults for optimization



Model	Accuracy No OOV No EOS Last 100 Validation Rounds
Random	0.2019
Large PCA GPT2 + PP	0.2065

Future Work

- Current research for NWP is limited to the Stack Overflow federated dataset, because the other federated text dataset, Shakespeare, currently only deals with character level tasks

- With more federated datasets available for NWP tasks, we can extend our pre-training and make it more robust

- Subsequently, we can also evaluate federated pretraining and federated fine-tuning, in addition to the current experiments using central pretraining and federated fine-tuning

- While small models seem to limit accuracy, further experimentation with small model architectures may yield improved results

Thank You!

Git Repo

Final Research Report