Lecture 8: Character-Aware NLM, BiDAF, CoVe, ELMo, ULM-Fit

Andrei Arsene Simion

Columbia University

March 22, 2023

Outline

- High level idea for today: Contextual Embedding and applications of Attention
- Character-Aware NLM
- BiDAF
- CoVe
- ELMo
- ULM-Fit

Outline

- Character-Aware NLM
- BiDAF
- CoVe
- ELMo
- ULM-Fit

Character-Aware NLM

- One Application of LSTMs was language modeling
- Essentially, the LSTM allows us to model long sequences, we can naturally model $P(y_t|y_{t-1}, \ldots, y_0)$
- Specifically, we can pass in word embeddings (GloVe, Word2Vec) for each y_t and use this to predict y_{t+1}; we get more context as we go from left to right
- But we might have a problem: lots of parameters and word embeddings do not know much about each other
  - Words like eventful, eventfully, uneventful, and uneventfully should have structurally related embeddings in the vector space
  - GloVe (2014) and Word2Vec (2013) don't allow for this structure necessarily, FastText (2016) does (kind of)
  - Another idea: Character CNN-based embeddings!

Character-Aware NLM

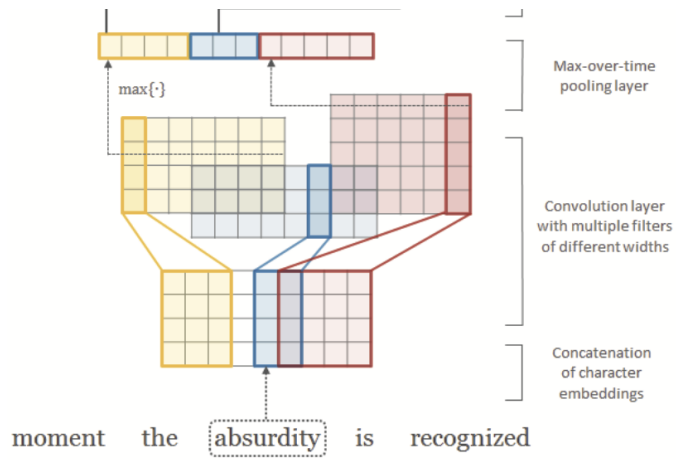
- In 2015, a few authors put together the LSTM, a CNN, and the Highway Connection into a nice language model that has relatively few parameters (this is one example)
- Such models might be very useful on edge devices, cell phones and other tools which have limited memory
- High level idea: replace each (independent) word embedding with an embedding at the character level (so, tokens are characters not words), then apply multiple filters to the word's character embedding matrix
  - Concatenate the results of the filters to get a word's (word-level) embedding

Character-Aware NLM

- Let's focus in on the word embedding mechanism
- Example: each character has a word embedding of dimension $d = 4$
- If a word x has l_x characters in it, apply various filter over the $(4 \times l_x)$ matrix C^x and get the embedding per filter
- Each filter has a matrix H of dimension $(4 \times w)$ and a scalar bias b and we get a result of convolution f^x with

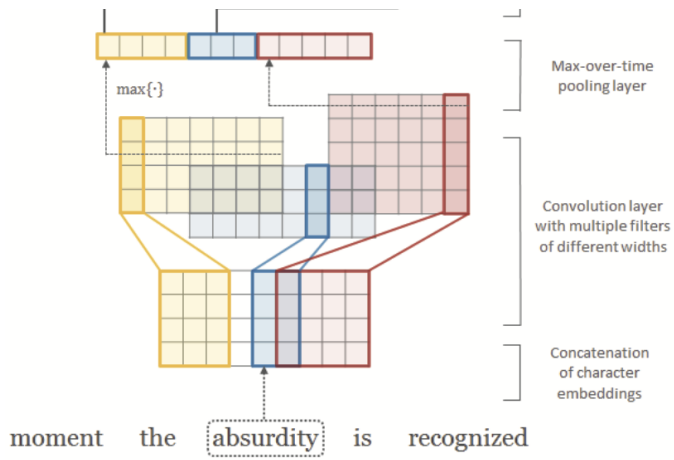
$$f^x[t] = tanh(C^x[* : t : t + w] \odot H + b)$$

  per filter
- Example: here we have 4 (4×3) filters, 3 (4×2) filters and 5 (4×4) filters



Character-Aware NLM

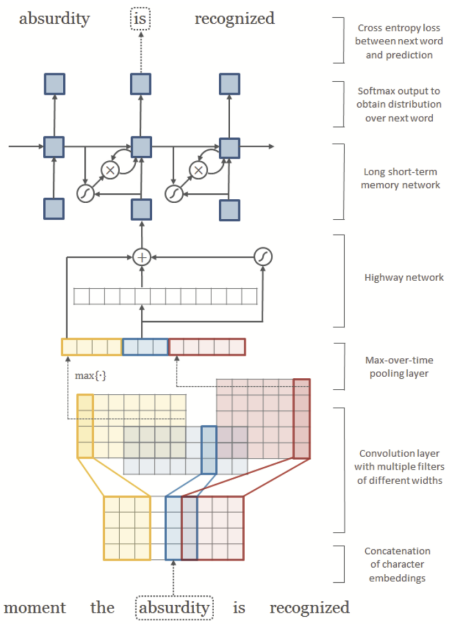
- For each filter, you get a vector of dimension $(1, l_x - w + 1)$ (no padding)
- You then take max-pool over time so you finally represent the j^{th} filter's information as $y_j^x = \max_t(f_j^x[t])$
- Apply h independent filters (various widths) to get a representation $(y_1^x, \ldots, y_h^x)$ for the word x, here $x =$ "*absurdity*"



Character -Aware NLM

- Another idea authors use to get better representations is a Highway architecture
- Like LSTMs, Highway networks use a gate $t = \sigma(Wy + b)$ to keep and forget information
- They pass each CNN-derived word embedding y through a highway connection so that they finally get the representation

$$z = t \odot ReLU(W_H y + b_H) + (1 - t) \odot y$$



Character-Aware NLM: Other ideas

- The paper uses different filter numbers depending on the width
- They use Hierarchical softmax: split the token vocabulary V into $c = \lceil \sqrt{|V|} \rceil$ clusters $\{V_1, \ldots, V_c\}$ of roughly equal size and randomly assign words to each cluster
- Then, $P(w_{t+1} = j | w_t, \ldots, w_1) \sim P(j \in V_r) P(w_{t+1} = j | V_r)$
- Specifically,

$$P(w_{t+1} = j | w_t, \ldots, w_1) = \frac{exp(h_t s^r + t^r)}{\sum_{r'=1}^{c} exp(h_t s^{r'} + t^{r'})} \frac{exp(h_t p_r^j + q_r^j)}{\sum_{j' \in V_r} exp(h_t p_r^{j'} + q_r^{j'})}$$

		Small	Large
CNN	d	15	15
	w	$[1, 2, 3, 4, 5, 6]$	$[1, 2, 3, 4, 5, 6, 7]$
	h	$[25 \cdot w]$	$[\min\{200, 50 \cdot w\}]$
	f	tanh	tanh
Highway	l	1	2
	g	ReLU	ReLU
LSTM	l	2	2
	m	300	650

Table 2: Architecture of the small and large models. d = dimensionality of character embeddings; w = filter widths; h = number of filter matrices, as a function of filter width (so the large model has filters of width $[1, 2, 3, 4, 5, 6, 7]$ of size $[50, 100, 150, 200, 200, 200, 200]$ for a total of 1100 filters); f, g = nonlinearity functions; l = number of layers; m = number of hidden units.

Char-NLM: Better Performance + Lower Params

- Fewer parameters, better perplexity

	PPL	Size
LSTM-Word-Small	97.6	5 m
LSTM-Char-Small	92.3	5 m
LSTM-Word-Large	85.4	20 m
LSTM-Char-Large	78.9	19 m
KN-5 (Mikolov et al. 2012)	141.2	2 m
RNN† (Mikolov et al. 2012)	124.7	6 m
RNN-LDA† (Mikolov et al. 2012)	113.7	7 m
genCNN† (Wang et al. 2015)	116.4	8 m
FOFE-FNNLM† (Zhang et al. 2015)	108.0	6 m
Deep RNN (Pascanu et al. 2013)	107.5	6 m
Sum-Prod Net† (Cheng et al. 2014)	100.0	5 m
LSTM-1† (Zaremba et al. 2014)	82.7	20 m
LSTM-2† (Zaremba et al. 2014)	78.4	52 m

Table 3: Performance of our model versus other neural language models on the English Penn Treebank test set. PPL refers to perplexity (lower is better) and size refers to the approximate number of parameters in the model. KN-5 is a Kneser-Ney 5-gram language model which serves as a non-neural baseline. †For these models the authors did not explicitly state the number of parameters, and hence sizes shown here are estimates based on our understanding of their papers or private correspondence with the respective authors.

- Each element (char embeddings, or Highway), has a benefit

	In Vocabulary					Out-of-Vocabulary		
	while	*his*	*you*	*richard*	*trading*	*computer-aided*	*misinformed*	*looooook*
LSTM-Word	*although*	*your*	*conservatives*	*jonathan*	*advertised*	–	–	–
	letting	*her*	*we*	*robert*	*advertising*	–	–	–
	though	*my*	*guys*	*neil*	*turnover*	–	–	–
	minute	*their*	*i*	*nancy*	*turnover*	–	–	–
LSTM-Char (before highway)	*chile*	*this*	*your*	*hard*	*heading*	*computer-guided*	*informed*	*look*
	whole	*hhs*	*young*	*rich*	*training*	*computerized*	*performed*	*cook*
	meanwhile	*is*	*four*	*richer*	*reading*	*disk-drive*	*transformed*	*looks*
	white	*has*	*youth*	*richter*	*leading*	*computer*	*inform*	*shook*
LSTM-Char (after highway)	*meanwhile*	*hhs*	*we*	*eduard*	*trade*	*computer-guided*	*informed*	*look*
	whole	*this*	*your*	*gerard*	*training*	*computer-driven*	*performed*	*looks*
	though	*their*	*doug*	*edward*	*traded*	*computerized*	*outperformed*	*looked*
	nevertheless	*your*	*i*	*carl*	*trader*	*computer*	*transformed*	*looking*

Table 6: Nearest neighbor words (based on cosine similarity) of word representations from the large word-level and character-level (before and after highway layers) models trained on the PTB. Last three words are OOV words, and therefore they do not have representations in the word-level model.

Char NLM: PCA

- Embeddings for prefixes, suffixes, etc get placed close together; embeddings for things with hyphens also get close placement
- Authors thought the morphemes (e.g. "in", "come", "-ing", forming incoming) would get picked up (which element of the char vector would get the maximum for each filter), but this was not the case

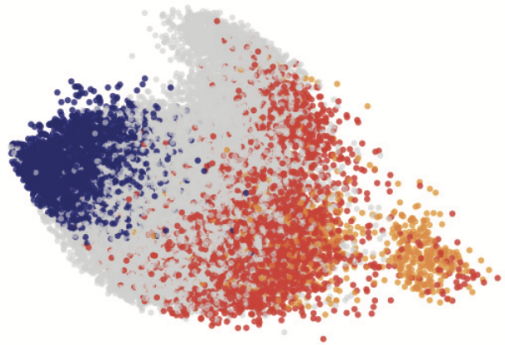


Figure 2: Plot of character n-gram representations via PCA for English. Colors correspond to: prefixes (red), suffixes (blue), hyphenated (orange), and all others (grey). Prefixes refer to character n-grams which start with the start-of-word character. Suffixes likewise refer to character n-grams which end with the end-of-word character.

Outline

- Character-Aware NLM
- BiDAF
- CoVe
- ELMo
- ULM-Fit

Question Answering

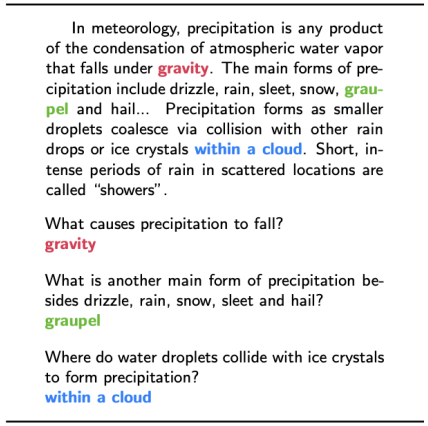
- What is Question Answer in NLP?
  - At a high level, we have a document D, which is long
  - We have a question Q which refers to something in the document
  - We what to get answer A to Q given the context D
- Lots of NLP problems can be formulated as Question Answering examples
  - Information Extraction
    - D : Barrack Obama was born in Hawaii. After graduating from Columbia, he worked as a community organizer in Chicago.
    - Q : Where did Obama graduate from?
    - Q : Where did Obama work after graduating?
  - Semantic Role Labeling - Who did what to whom?
    - D : The Miami Heat finished the 2000 championships as NBA champions, by beating the Knicks in the final.
    - Q - beat : When did someone beat someone? (in the final)
    - Q - beat : Who did someone beat? (the Knicks)
    - Q - finish : Who finished something? (the Miami Heat)
    - Q - finish : What did someone finish? (the 2000 championship)

Question Answering: SQuAD

- What is the SQuAD dataset?
  - 100K annotated (passage, question, answer) triplets
  - Passages are selected from English Wikipedia and are usually 100 to 150 words
  - Questions are crowd sourced
  - Each answer is a short piece of text from the question - the goal is to identify the indices spanning the answer
  - Notice that this is big limitation - but it is still one of the most popular NLP benchmarks and model performance has exceeded human performance

SQuAD Examples and 2 Metrics

- SQuAD 1.0 (2016)



In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called "showers".

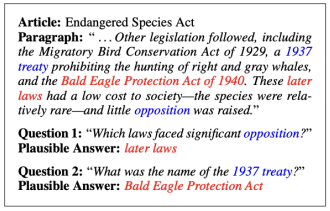
What causes precipitation to fall?
gravity

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

Figure 1: Question-answer pairs for a sample passage in the SQuAD dataset. Each of the answers is a segment of text from the passage.

- SQuAD 2.0 (2018)



Article: Endangered Species Act
Paragraph: "*. . . Other legislation followed, including the Migratory Bird Conservation Act of 1929, a 1937 treaty prohibiting the hunting of right and gray whales, and the Bald Eagle Protection Act of 1940. These later laws had a low cost to society—the species were relatively rare—and little opposition was raised.*"

Question 1: "*Which laws faced significant opposition?*"
Plausible Answer: *later laws*

Question 2: "*What was the name of the 1937 treaty?*"
Plausible Answer: *Bald Eagle Protection Act*

Figure 1: Two unanswerable questions written by crowdworkers, along with plausible (but incorrect) answers. Relevant keywords are shown in blue.

- Exact Match (EM): 0/1 metric if your model correctly identified the start / end indices
- F Score: A metric like above, but softer

BiDAF: Bidirectional Attention Flow

- The model is a neural model which heavily uses attention, let's see how it worked
- Consider a passage D with words $\{x_1, \ldots, x_T\}$ and question Q with words $\{q_1, \ldots, q_J\}$
- We want to link D and Q somehow and also incorporate the supervised answer A

BiDAF Model

- Character Embeddings
  - Taking ideas from before, for each word, we can split it into characters and run a CNN over that word
  - Many options to get such embeddings - Can even do RNN!

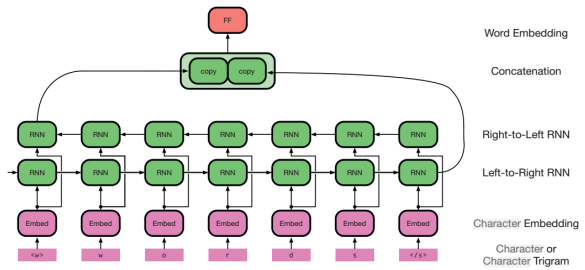


Figure 12.7 Word representation composed from a letter sequence using a bidirectional recurrent neural network. Instead of single letters, we may also use overlapping letter *n*-grams (here trigrams).

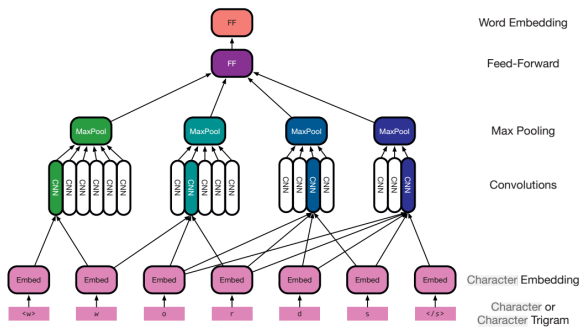
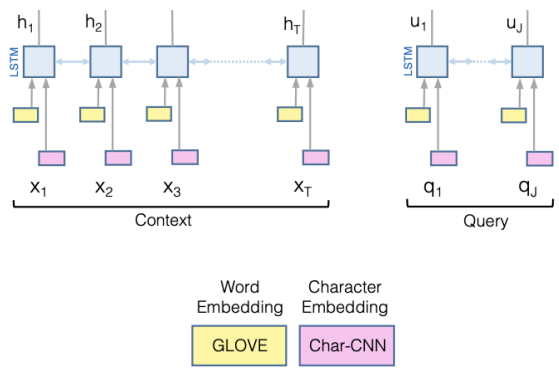


Figure 12.8 Word representation composed from a letter sequence using a convolutional neural network. Convolutions over 2-, 3-, 4-, and 5-letter *n*-grams are shown.

- Word Embeddings
  - Authors use GloVe as the embedding for each word
- Each word x has an embedding that is of two flavors and equal to $[GloVe(x), CNN(x)]$

BiDAF Model

- For each word, pass e into a highway network with two layers
- Get for each document a matrix X of embeddings of dimension $(d \times T)$
- Get for each query a matrix Q of embeddings of dimension $(d \times J)$
- Get Contextual Embeddings of X, Q, then refine further
  - Put a bidirectional LSTM and run the data (both X and Q) through to get new embeddings for X and Q
  - From X, get H of dimension $(2d \times T)$
  - From Q, get U of dimension $(2d \times T)$
  - Note that we concatenate the forward and backward states to get the $2d$ dimension, $h_t = (\overrightarrow{h}_t, \overleftarrow{h}_t)$



BiDAF Model : Context-to-Query Attention

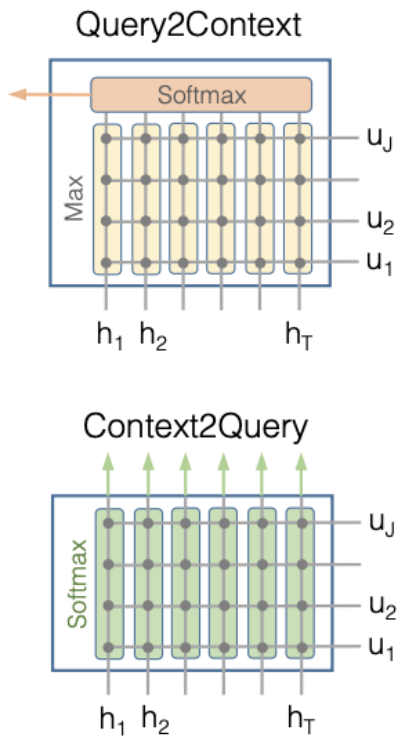
- Formulate a score matrix $S_{tj} = w_S^T[h_t, u_j, h_t \odot u_j]$ that is of dimension $(T \times J)$
- w_S is a parameter to learn and it is of dimension $\mathbb{R}^{6d}$
- Let a_{tj} be standard attention weights with $\sum_j a_{tj} = 1$ for any t - apply softmax to S across each row and get $A = softmax(S)$ so with rowSum(A) $= 1$
- Context-to-Query Attention sets $\tilde{u}_t = \sum_j a_{tj} u_j$ so that we have T vectors
- Each $\tilde{u}$ vector is a combination of *query* vectors so we are effectively associating with each context vector a combination of *query* vectors
- Idea: Express each context word as best possible using query words
- $\tilde{U}$ represents the $(2d \times T)$ matrix if we do this for all all t

BiDAF Model : Query-to-Context Attention

- Another idea the authors introduce is Query-to-Context Attention
- Idea: Which context vectors are most relevant to the query
- Consider $\beta_t \propto \max\{S_{t1}, \ldots, S_{tJ}\}$ and β are normalized via a softmax
- For each context word, we are taking the most important *query* word
- This type of attention signifies which context words have the closest similarity to one of the query words and are hence critical for answering the query
- We set (1 vector) $\tilde{h} = \sum_t \beta_t h_t$ which in some sense represents as close a combination of words from the context relative to the query
- $\tilde{H}$ represents the $(2d \times T)$ matrix if we use $\tilde{h}$ T times for each column (same $\tilde{h}$)

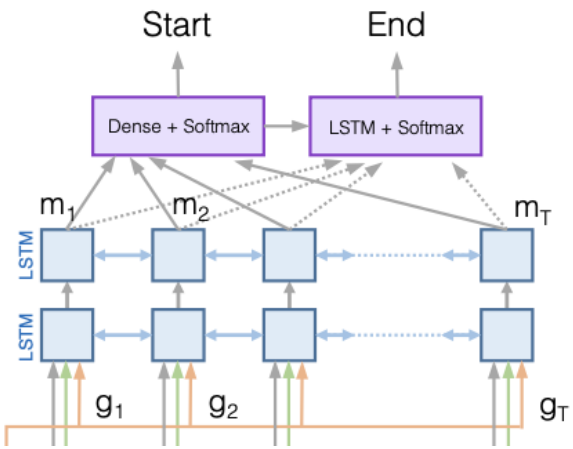
BiDAF Model : Attention

- Context2Query: For a fixed context word, get the importance weights of each query word; for each context word express it as a weighted sum of query words using the query embeddings
- Query2Context: For each context word, get the most important query word; using normalized weights, get a weighted sum of h_t to get a close representation for the query in terms of the context vectors



BiDAF Model : Modeling Layer

- Finally, combine the vectors into $G_t = (h_t, \tilde{u}_t, h_t \odot \tilde{u}_t, h_t \odot \tilde{h}_t)$
- The input to the modeling layer is G, which encodes the query-aware representations of context words
- The output of the modeling layer captures the interaction among the context words conditioned on the query
- This modeling layer M^{start} is gotten via another LSTM application and produces this $(2d \times T)$ representation

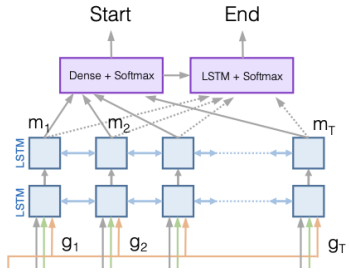


BiDAF Model : Modeling and Output Layer

- For the output, model the start index via $p^{start} = softmax(w^{\mathsf{T}}_{p^{start}}(G, M^{start}))$
- $w_{p^{start}}$ is in $\mathbb{R}^{10d}$ (!)
- Pass M^{start} into another LSTM to get a M^{end} representation of dimension $(2d \times T)$
- Model the probability of the end token as $p^{end} = softmax(w^{\mathsf{T}}_{p^{end}}(G, M^{end}))$
- Optimize the standard Cross-Entropy of over the real start and end indices via

$$L(\theta) = -\frac{\sum_i^N log\{p^{start}_{y_i^{start}}\} + log\{p^{end}_{y_i^{end}}\}}{N}$$

- Model parameters: all LSTM parameters, all CNN parameters, w_S, $w_{p^{start}}$ and $w_{p^{end}}$



BiDAF Model : Test

- At test time we need to compute the span (k, l) for with $k \leq l$ with maximum probability $p_k^{start} p_l^{end}$
- You can do this with Dynamic Programming in $O(T)$ time!
- Ensemble: 12 independent identical BiDAF models; this was SOTA in 2017
- Ablation study shows each element of BiDAF is important

	Single Model		Ensemble	
	EM	F1	EM	F1
Logistic Regression Baseline[a]	40.4	51.0	-	-
Dynamic Chunk Reader[b]	62.5	71.0	-	-
Fine-Grained Gating[c]	62.5	73.3	-	-
Match-LSTM[d]	64.7	73.7	67.9	77.0
Multi-Perspective Matching[e]	65.5	75.1	68.2	77.2
Dynamic Coattention Networks[f]	66.2	75.9	71.6	80.4
R-Net[g]	**68.4**	**77.5**	72.1	79.7
BIDAF (Ours)	68.0	77.3	**73.3**	**81.1**

(a) Results on the SQuAD test set

	EM	F1
No char embedding	65.0	75.4
No word embedding	55.5	66.8
No C2Q attention	57.2	67.7
No Q2C attention	63.6	73.7
Dynamic attention	63.5	73.6
BIDAF (single)	67.7	77.3
BIDAF (ensemble)	72.6	80.7

(b) Ablations on the SQuAD dev set

Table 1: (1a) The performance of our model BIDAF and competing approaches by Rajpurkar et al. (2016)[a], Yu et al. (2016)[b], Yang et al. (2016)[c], Wang & Jiang (2016)[d], IBM Watson[e] (unpublished), Xiong et al. (2016b)[f], and Microsoft Research Asia[g] (unpublished) on the SQuAD test set. A concurrent work by Lee et al. (2016) does not report the test scores. All results shown here reflect the SQuAD leaderboard (`stanford-qa.com`) as of 6 Dec 2016, 12pm PST. (1b) The performance of our model and its ablations on the SQuAD dev set. Ablation results are presented only for single runs.

BiDAF Model : Test

- They also look at cloze-like tasks - Can you comprehend text? Fill in the missing word!
- See the paper for other nice analysis

	CNN		DailyMail	
	val	test	val	test
Attentive Reader (Hermann et al., 2015)	61.6	63.0	70.5	69.0
MemNN (Hill et al., 2016)	63.4	6.8	-	-
AS Reader (Kadlec et al., 2016)	68.6	69.5	75.0	73.9
DER Network (Kobayashi et al., 2016)	71.3	72.9	-	-
Iterative Attention (Sordoni et al., 2016)	72.6	73.3	-	-
EpiReader (Trischler et al., 2016)	73.4	74.0	-	-
Stanford AR (Chen et al., 2016)	73.8	73.6	77.6	76.6
GAReader (Dhingra et al., 2016)	73.0	73.8	76.7	75.7
AoA Reader (Cui et al., 2016)	73.1	74.4	-	-
ReasoNet (Shen et al., 2016)	72.9	74.7	77.6	76.6
BIDAF (Ours)	**76.3**	**76.9**	**80.3**	**79.6**
MemNN* (Hill et al., 2016)	66.2	69.4	-	-
ASReader* (Kadlec et al., 2016)	73.9	75.4	78.7	77.7
Iterative Attention* (Sordoni et al., 2016)	74.5	75.7	-	-
GA Reader* (Dhingra et al., 2016)	76.4	77.4	79.1	78.1
Stanford AR* (Chen et al., 2016)	77.2	77.6	80.2	79.2

Table 3: Results on CNN/DailyMail datasets. We also include the results of previous ensemble methods (marked with *) for completeness.

Outline

- Character-Aware NLM
- BiDAF
- CoVe
- ELMo
- ULM-Fit

CoVe

- Word2Vec, FastText, GloVe all have 1 problem: they assume there is one true word embedding for a word
- A word like "bank" can have multiple meaning, and they depend on the context
  - I was on the river bank
  - I robbed the bank
- How can we learn Contextualized Word Vectors?
- Big idea in what follows is Transfer Learning: build a model that captures context in both directions per word: a recursive language model, a Encoder from NMT, etc
- Save all or part of this model, and use it as a preprocessor to any other model; the big idea is as below
  - Step 1: Build a model to help us "walk" (base knowledge)
  - Step 2: Use Step 1 to help us "run", or "sing", or "add" (transfer Step 1 and add a little more)
- Your choice: Either freeze the old model (Step 1) or fine-tune the parameters of this model to the new task/data
- Question: How do you use these models?

Transfer Learning using an Encoder

- One idea: use the Encoder from a translation model

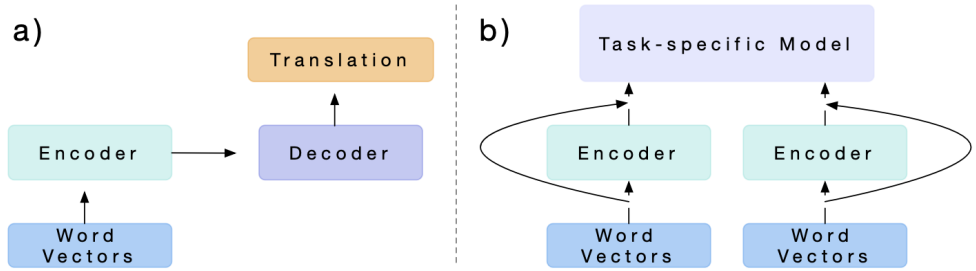
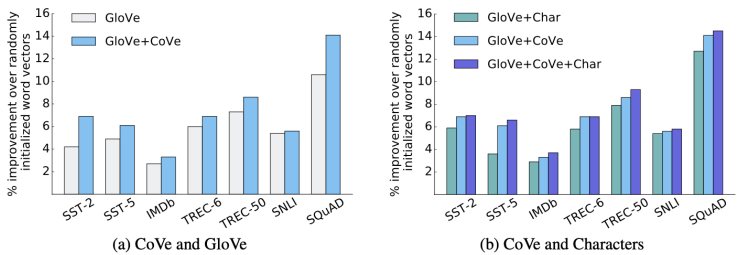


Figure 1: We a) train a two-layer, bidirectional LSTM as the encoder of an attentional sequence-to-sequence model for machine translation and b) use it to provide context for other NLP models.

- Authors use a bidirectional attention-based LSTM Encoder-Decoder with two layers for each
- Initialize the Encoder with GloVe embeddings, and train your model for the SMT task, then save the Encoder
- Specifically, the Encoder is a bidirectional LSTM
  - Model $p(x_1) \dots p(x_t|x_{t-1}, \dots, x_1)$ and $p(x_1|x_2, \dots, x_t) \dots p(x_t)$
  - The final representation of token t is the concatenation of the forward LSTM's and backward LSTM's hidden states
  - Explicitly: $h_t = (\overrightarrow{h}_t, \overleftarrow{h}_t)$
  - h_t has information about the left and the right context of the word x_t!

CoVe

- Idea: Don't just use $GloVe(x_t)$ as en embedding, use $(GloVe(x_t), CoVe(x_t))$ where $CoVe$ is the NMT Encoder
- Train 3 Encoder-Decoder NMT models considering English-German translation on larger datasets: $CoVe - S$, $CoVe - M$, $CoVe - L$
- GloVe + CoVe + Char is superior to any ablation
- Examples involve Text Classification + SQuAD



(a) CoVe and GloVe (b) CoVe and Characters

Figure 3: The Benefits of CoVe

Dataset	Random	GloVe	GloVe+ Char	GloVe+ CoVe-S	GloVe+ CoVe-M	GloVe+ CoVe-L	GloVe+ Char+CoVe-L
SST-2	84.2	88.4	90.1	89.0	90.9	91.1	**91.2**
SST-5	48.6	53.5	52.2	54.0	54.7	54.5	**55.2**
IMDb	88.4	91.1	91.3	90.6	91.6	91.7	**92.1**
TREC-6	88.9	94.9	94.7	94.7	95.1	95.8	**95.8**
TREC-50	81.9	89.2	89.8	89.6	89.6	90.5	**91.2**
SNLI	82.3	87.7	87.7	87.3	87.5	87.9	**88.1**
SQuAD	65.4	76.0	78.1	76.5	77.1	79.5	**79.9**

Table 2: CoVe improves validation performance. CoVe has an advantage over character n-gram embeddings, but using both improves performance further. Models benefit most by using an MT-LSTM trained with MT-Large (CoVe-L). Accuracy is reported for classification tasks, and F1 is reported for SQuAD.

CoVe Results

- They observe that better / larger data leads to better performance

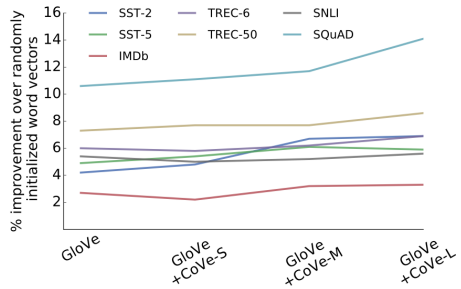


Figure 4: The Effects of MT Training Data

- They get SOTA on some Classification tasks

	Model	Test		Model	Test
SST-2	P-LSTM [Wieting et al., 2016]	89.2	TREC-6	SVM [da Silva et al., 2011]	95.0
	CT-LSTM [Looks et al., 2017]	89.4		SVM [Van-Tu and Anh-Cuong, 2016]	95.2
	TE-LSTM [Huang et al., 2017]	89.6		DSCNN-P [Zhang et al., 2016]	95.6
	NSE [Munkhdalai and Yu, 2016a]	89.7		*BCN+Char+CoVe [Ours]*	*95.8*
	BCN+Char+CoVe [Ours]	*90.3*		TBCNN [Mou et al., 2015]	96.0
	bmLSTM [Radford et al., 2017]	**91.8**		**LSTM-CNN [Zhou et al., 2016]**	**96.1**
SST-5	MVN [Guo et al., 2017]	51.5	TREC-50	SVM [Loni et al., 2011]	89.0
	DMN [Kumar et al., 2016]	52.1		SNoW [Li and Roth, 2006]	89.3
	LSTM-CNN [Zhou et al., 2016]	52.4		*BCN+Char+CoVe [Ours]*	*90.2*
	TE-LSTM [Huang et al., 2017]	52.6		RulesUHC [da Silva et al., 2011]	90.8
	NTI [Munkhdalai and Yu, 2016b]	53.1		SVM [Van-Tu and Anh-Cuong, 2016]	91.6
	BCN+Char+CoVe [Ours]	***53.7***		**Rules [Madabushi and Lee, 2016]**	**97.2**
IMDb	*BCN+Char+CoVe [Ours]*	*91.8*	SNLI	DecAtt+Intra [Parikh et al., 2016]	86.8
	SA-LSTM [Dai and Le, 2015]	92.8		NTI [Munkhdalai and Yu, 2016b]	87.3
	bmLSTM [Radford et al., 2017]	92.9		re-read LSTM [Sha et al., 2016]	87.5
	TRNN [Dieng et al., 2016]	93.8		btree-LSTM [Paria et al., 2016]	87.6
	oh-LSTM [Johnson and Zhang, 2016]	94.1		600D ESIM [Chen et al., 2016]	88.0
	Virtual [Miyato et al., 2017]	**94.1**		***BCN+Char+CoVe [Ours]***	***88.1***

Table 4: Single model test accuracies for classification tasks.

CoVe Results

- They show that CoVe is a better way to encode a sentence

	GloVe+Char+	
Dataset	Skip-Thought	CoVe-L
SST-2	88.7	**91.2**
SST-5	52.1	**55.2**
TREC-6	94.2	**95.8**
TREC-50	89.6	**91.2**
SNLI	86.0	**88.1**

Table 5: Classification validation accuracies with skip-thought and CoVe.

- They get SOTA on SQuAD task

Model	EM	F1
LR [Rajpurkar et al., 2016]	40.0	51.0
DCR [Yu et al., 2017]	62.5	72.1
hM-LSTM+AP [Wang and Jiang, 2017]	64.1	73.9
DCN+Char [Xiong et al., 2017]	65.4	75.6
BiDAF [Seo et al., 2017]	68.0	77.3
R-NET [Wang et al., 2017]	71.1	79.5
DCN+Char+CoVe [Ours]	***71.3***	***79.9***

Table 3: Exact match and F1 validation scores for single-model question answering.

CoVe Usage

- How do you use CoVe vectors?
- For Text Classification, do we just average the CoVe vectors?
- No, they use Attention!

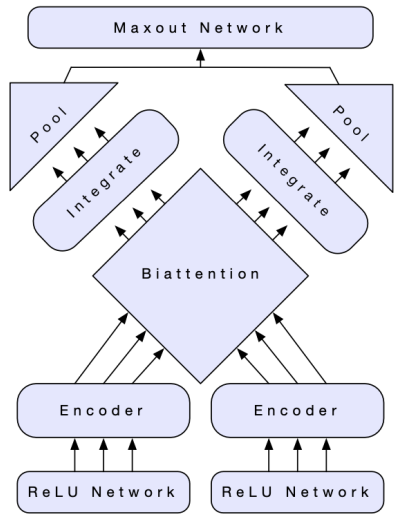


Figure 2: Our BCN uses a feedforward network with ReLU activation and biLSTM encoder to create task-specific representations of each input sequence. Biattention conditions each representation on the other, a biLSTM integrates the conditional information, and a maxout network uses pooled features to compute a distribution over possible classes.

CoVe Attention

- For Entailment (does sentence w^x follow from sentence w^y) you have two sentences
- Assume w^x, w^y have lengths T_a and T_b
- For text classification, you have 1 sentence so $w^x = w^y$ in what follows
- The process goes in several steps, the big idea is you get contextual representations
  - Set $\tilde{w}^x = (GloVe(w^x); CoVe(w^x))$ and similarly for w^y
  - Each word vector has a dimension d, and there are T_a in x and T_b in y
  - Set $x = biLSTM(FF(\tilde{w}^x))$ and similarly for y
  - Stack X and Y so that dimensions are $(T_a \times d)$ and $(T_b \times d)$
  - Create a score matrix $S = XY^\intercal$ $(T_a \times T_b)$
  - Create column-wise normalizations $A_x = softmax(S)$ and $A_y = softmax(S^\intercal)$
    - For each column of A_x, we have weights telling is which x is most important to a fixed y
    - Similarly for A_y

CoVe Attention

- A_x is $(T_a \times T_b)$
- Create $C_x = A_x^{\intercal} X$ which is $(T_b \times d)$, a representation of y from x
- Similarly, create C_y
- Create representations again by running another biLSTM so we get
  - $X_{|y} = biLSTM([X; X - C_y; X \odot C_y])$ which is $(T_a \times d)$
  - $Y_{|x} = biLSTM([Y; Y - C_x; Y \odot C_x])$ which is $(T_b \times d)$
- Create weights $\beta_x = softmax(X_{|y} v_1 + b_1)$ which is $(T_a \times 1)$
- Create weights $\beta_y = softmax(Y_{|x} v_2 + b_2)$ which is $(T_b \times 1)$

CoVe Attention

- Pool across time to get d dimensional vectors $max(X_{|y})$, $mean(X_{|y})$ and $min(X_{|y})$
- Multiply $(T_a \times 1)$ vector β_x by $(d \times T_a)$ matrix $X_{|y}^{\intercal}$ to get d dimensional vector $x_{self} = X_{|y}^{\intercal}\beta_x$ $(d \times 1)$
- Set $x_{pool} = [max(X_{|y}); mean(X_{|y}); min(X_{|y}); x_{self}]$
- Similarly get y_{pool}
- Feed the representation $[x_{pool}; y_{pool}]$ through a three-layer, batch normalized, maxout network and get the class probabilities (just a MLP)
- Big idea: you want a mixed representation, taking into account w^x and w^y; Attention allows you to do this
- SQuAD like tasks are similar; see the paper for minor differences

Outline

- Character-Aware NLM
- BiDAF
- CoVe
- ELMo
- ULM-Fit

ELMo

- Another model that generated word embeddings was titled like from Sesame Street and called *ELMo*: Embeddings From Language Models
- Similar idea to *CoVe*, but better - used Language Modeling directly as opposed to Encoder-Decoder
  - Language Modeling was the main task for *ELMo*
  - Highway Networks and bidirectional LSTM are again used
  - Model is initialized not with GloVe but with Character level embeddings that are learned (in *CoVe* the embeddings at the GloVe input level were fixed) and used to get a starting word embedding

ELMo

- The key ingredients are are follows
  - For each word token, get a character level embedding for the word
  - Pass each character level embedding through two Highway Layers
  - Use the result of the Highway Layers as the input to a bidirectional LSTM as in CoVe
  - Do more than CoVe did - CoVe just took the top representations of the Encoder and then "mixed" them

ELMo: LSTM details

- The LSTM objective is to maximize

$$\sum_{k=1}^{N} [\log p(x_k | x_1, \ldots, x_{k-1}, \overrightarrow{\Theta}) + \log p(x_k | x_{k+1}, \ldots, x_N, \overleftarrow{\Theta})]$$

- The input data and output Softmax weights are shared by both models
- After optimization, if each LSTM has L layers, we have representations at each time step t
  - $\overrightarrow{h}_t^0, \ldots, \overrightarrow{h}_t^L$ for the forward LSTM
  - $\overleftarrow{h}_t^0, \ldots, \overleftarrow{h}_t^L$ for the backward LSTM
- We can get a layer level representation by setting $h_t^l = (\overrightarrow{h}_t^l, \overleftarrow{h}_t^l)$
- Note that we do not just take the final layer! We combine layers!

ELMo: How to get the embeddings

- Given a token x_t and a specific task, ELMo uses a combination of all h_t^l to represent x_t
  - We push the data through the trained LSTM, and get $h_t^0, \ldots, h_t^L$
  - For a specific task, we then use $ElMo(x_t) = \gamma^{task} \sum_{l=0}^{L} s_l^{task} h_t^l$
  - γ^{task} is a parameter that is task specific
  - $(s_0^{task}, \ldots, s_L^{task})$ are parameters whose values is conditioned to be positive and sum to 1
- For a general RNN model, you can now pass $(GloVe(x_t), ELMo(x_t))$ to the RNN instead of just $GloVe(x_t)$
- For SQuAD or SNLI (Entailment) tasks they use ELMo at the top of the output RNN, so you use $(k_t, ELMo(x_t))$ to predict, not just k_t

ELMo

- Upshot: ELMo should be used as an input and can be used as an output
- ELMo should not be just 1 of the h_t^l, it should be a weighted sum of all of them $h_t = \gamma^{task} \sum_{l=0}^{L} s_l^{task} h_t^l$

Task	Baseline	Last Only	All layers	
			λ=1	λ=0.001
SQuAD	80.8	84.7	85.0	**85.2**
SNLI	88.1	89.1	89.3	**89.5**
SRL	81.6	84.1	84.6	**84.8**

Table 2: Development set performance for SQuAD, SNLI and SRL comparing using all layers of the biLM (with different choices of regularization strength λ) to just the top layer.

Task	Input Only	Input & Output	Output Only
SQuAD	85.1	**85.6**	84.8
SNLI	88.9	**89.5**	88.7
SRL	**84.7**	84.3	80.9

Table 3: Development set performance for SQuAD, SNLI and SRL when including ELMo at different locations in the supervised model.

- ELMo enhances almost all baselines and gets SOTA on many

TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 $\pm$ 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 $\pm$ 0.19	90.15	92.22 $\pm$ 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 $\pm$ 0.5	3.3 / 6.8%

Table 1: Test set comparison of ELMo enhanced neural models with state-of-the-art single model baselines across six benchmark NLP tasks. The performance metric varies across tasks – accuracy for SNLI and SST-5; F_1 for SQuAD, SRL and NER; average F_1 for Coref. Due to the small test sizes for NER and SST-5, we report the mean and standard deviation across five runs with different random seeds. The "increase" column lists both the absolute and relative improvements over our baseline.

ELMo

- ELMo allows you to get better close words - you capture "meaning" better

	Source	Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM	Chico Ruiz made a spectacular play on Alusik 's grounder {...}	Kieffer , the only junior in the group , was commended for his ability to hit in the clutch , as well as his all-round excellent play .
	Olivia De Havilland signed to do a Broadway play for Garson {...}	{...} they were actors who had been handed fat roles in a successful play , and had talent enough to fill the roles competently , with nice understatement .

Table 4: Nearest neighbors to "play" using GloVe and the context embeddings from a biLM.

Model	**F_1**
WordNet 1st Sense Baseline	65.9
Raganato et al. (2017a)	69.9
Iacobacci et al. (2016)	**70.1**
CoVe, First Layer	59.4
CoVe, Second Layer	*64.7*
biLM, First layer	67.4
biLM, Second layer	*69.0*

Table 5: All-words fine grained WSD F_1. For CoVe and the biLM, we report scores for both the first and second layer biLSTMs.

Model	**Acc.**
Collobert et al. (2011)	97.3
Ma and Hovy (2016)	97.6
Ling et al. (2015)	**97.8**
CoVe, First Layer	*93.3*
CoVe, Second Layer	92.8
biLM, First Layer	*97.3*
biLM, Second Layer	96.8

Table 6: Test set POS tagging accuracies for PTB. For CoVe and the biLM, we report scores for both the first and second layer biLSTMs.

ELMo

- If you don't have much data, ELMo allows for a great boost upfront
- Easy to visualize which level of the ELMo embedding is important by considering s^{task}

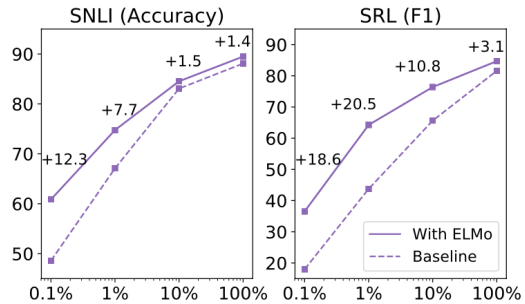


Figure 1: Comparison of baseline vs. ELMo performance for SNLI and SRL as the training set size is varied from 0.1% to 100%.

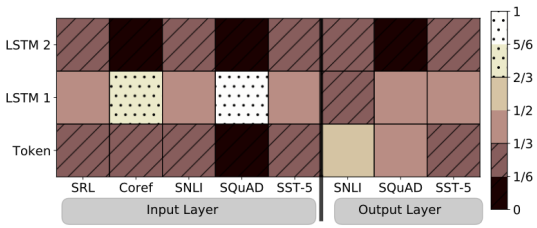


Figure 2: Visualization of softmax normalized biLM layer weights across tasks and ELMo locations. Normalized weights less then 1/3 are hatched with horizontal lines and those greater then 2/3 are speckled.

Outline

- Character Aware NLM
- BiDAF
- COVE
- ELMO
- ULM-Fit

ULM-Fit (Transfer Learning for Text Classification)

- Roughy at the same time ELMo came out, some other authors started realizing language models can be used as encoders of data
- The specific goal of this work was Text Classification
- Instead of using word embedding to initialize each token, they trained a language model first and then added a classification "head" on top
- The authors of ULM-Fit specifically think of their method as Transfer Learning: Given a task T and a source task $S \neq T$, how can you use S to get better performance on T?
- As with ELMo, S = Language Modeling
- This is a subset of Transfer Learning since their (language) model is necessarily changed (fine-tuned) to new data and then a new "head" (for classification) is added to the model - this is optimized step by step
- In Transfer Learning, the Encoder model (ELMo, CoVe), need not be optimized further - you pick

ULM-Fit

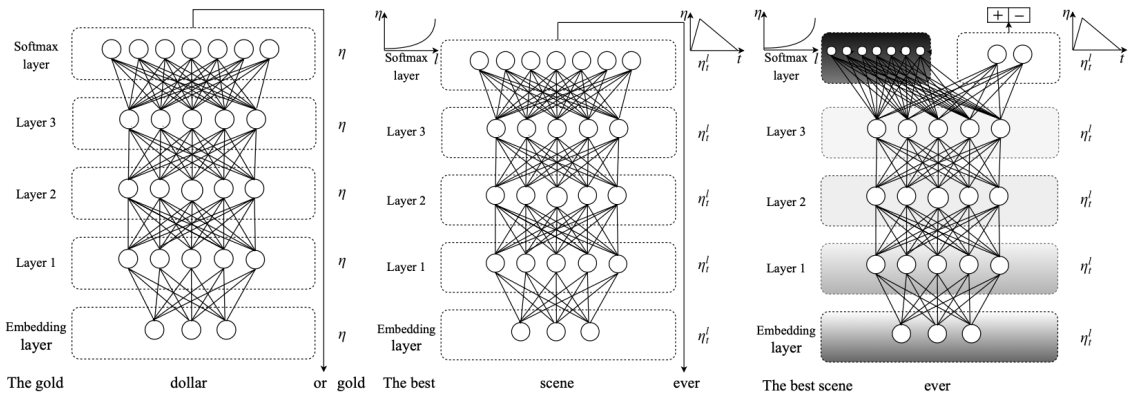
- The goal of ULM-Fit is Text Classification
- The steps of ULM-Fit are as follows
  - First use a general bidirectional LSTM to get a contextual encoding for each word - this is usually a few layers in the vertical direction
  - For example, train on Wikitext-103
  - Fine tune the language model on data from the target domain
    - Use Discriminative Fine Tuning: each layer of the language model gets a learning rate with lower layers getting a lower learning rate
    - The idea is lower layers have general features, while upper layers have specific features that can be changed
    - Generally $\nu^{l-1} = \nu^{l}/2.6$ is what they used
    - Slanted Triangular Learning Rates: the authors have an interesting learning rate scheduler for ν

ULM-Fit

- The final step is classifier fine tuning
  - The classification head is a MLP that takes in a representation for the text
  - Concatenate all the hidden layers of the forward (and backward) language model has $\overrightarrow{H} = (\overrightarrow{h}_1, \ldots, \overrightarrow{h}_T)$
  - Feed into the classifier $(\overrightarrow{h}_T, maxpool(\overrightarrow{H}), minpool(\overrightarrow{H}))$
  - Use gradual unfreezing when training the classifier:
    - Freeze all but the last language (L) model layer and fine tune (along with the MLP)
    - Freeze all but the last two layers of the language model ($L-1, L$) and fine tune (along with the MLP)
    - Etc.
  - Use BPTT for Text Classification: break up long documents into batch-chunks, and use the last hidden layer from one batch as the initial layer of another batch
  - Use a bidirectional LM and add a classification head to each: average the results to get the final prediction

ULM-Fit

- Nice visual of the main 3 steps for ULM-Fit
- Step 1: General LM
- Step 2: Learning rate depends on layer, domain specific data
- Step 3: Gradual unfreezing and fine-tuning



(a) LM pre-training (b) LM fine-tuning (c) Classifier fine-tuning

Figure 1: ULMFiT consists of three stages: a) The LM is trained on a general-domain corpus to capture general features of the language in different layers. b) The full LM is fine-tuned on target task data using discriminative fine-tuning ('*Discr*') and slanted triangular learning rates (STLR) to learn task-specific features. c) The classifier is fine-tuned on the target task using gradual unfreezing, '*Discr*', and STLR to preserve low-level representations and adapt high-level ones (shaded: unfreezing stages; black: frozen).

ULM-Fit

- This method got SOTA in 2018 on a few Text Classification tasks

	Model	Test		Model	Test
IMDb	CoVe (McCann et al., 2017)	8.2	TREC-6	CoVe (McCann et al., 2017)	4.2
	oh-LSTM (Johnson and Zhang, 2016)	5.9		TBCNN (Mou et al., 2015)	4.0
	Virtual (Miyato et al., 2016)	5.9		LSTM-CNN (Zhou et al., 2016)	3.9
	ULMFiT (ours)	**4.6**		ULMFiT (ours)	**3.6**

Table 2: Test error rates (%) on two text classification datasets used by McCann et al. (2017).

	AG	DBpedia	Yelp-bi	Yelp-full
Char-level CNN (Zhang et al., 2015)	9.51	1.55	4.88	37.95
CNN (Johnson and Zhang, 2016)	6.57	0.84	2.90	32.39
DPCNN (Johnson and Zhang, 2017)	6.87	0.88	2.64	30.58
ULMFiT (ours)	**5.01**	**0.80**	**2.16**	**29.98**

Table 3: Test error rates (%) on text classification datasets used by Johnson and Zhang (2017).

- Better data efficiency, fine tuning the LM using task data helps, don't just focus on the classifier's examples use both (Step 2 + Step 3)

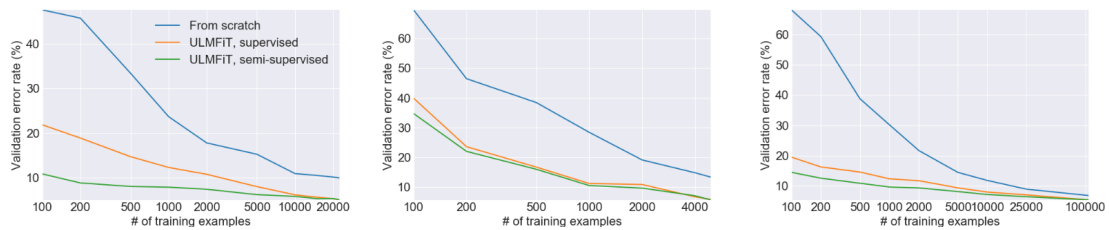


Figure 3: Validation error rates for supervised and semi-supervised ULMFiT vs. training from scratch with different numbers of training examples on IMDb, TREC-6, and AG (from left to right).

ULM-Fit

- Step 1 pretraining is crucial

Pretraining	IMDb	TREC-6	AG
Without pretraining	5.63	10.67	5.52
With pretraining	**5.00**	**5.69**	**5.38**

Table 4: Validation error rates for ULMFiT with and without pretraining.

- Each of the different "tricks" are useful
  - Dropout is critical to the LM

LM	IMDb	TREC-6	AG
Vanilla LM	5.98	7.41	5.76
AWD-LSTM LM	**5.00**	**5.69**	**5.38**

Table 5: Validation error rates for ULMFiT with a vanilla LM and the AWD-LSTM LM.

LM fine-tuning	IMDb	TREC-6	AG
No LM fine-tuning	6.99	6.38	6.09
Full	5.86	6.54	5.61
Full + discr	5.55	6.36	5.47
Full + discr + stlr	**5.00**	**5.69**	**5.38**

Table 6: Validation error rates for ULMFiT with different variations of LM fine-tuning.

Classifier fine-tuning	IMDb	TREC-6	AG
From scratch	9.93	13.36	6.81
Full	6.87	6.86	5.81
Full + discr	5.57	6.21	5.62
Last	6.49	16.09	8.38
Chain-thaw	5.39	6.71	5.90
Freez	6.37	6.86	5.81
Freez + discr	5.39	5.86	6.04
Freez + stlr	5.04	6.02	5.35
Freez + cos	5.70	6.38	**5.29**
Freez + discr + stlr	**5.00**	**5.69**	5.38

Table 7: Validation error rates for ULMFiT with different methods to fine-tune the classifier.

ULM-Fit

- Gradual unfreezing is critical: you should not fine tune all the layers at once

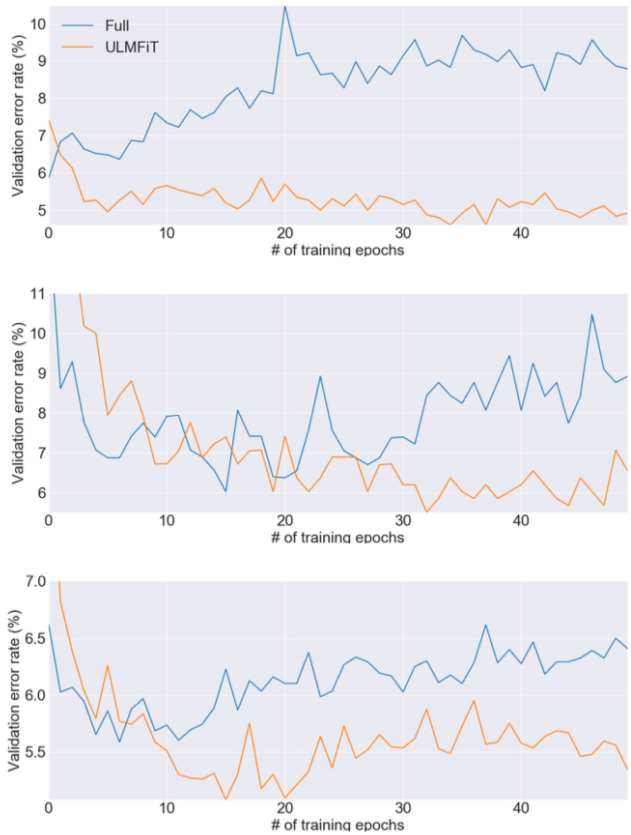


Figure 4: Validation error rate curves for fine-tuning the classifier with ULMFiT and '*Full*' on IMDb, TREC-6, and AG (top to bottom).

References

- SMT Book
- NMT Book
- ELMO
- BiDaf
- ULMFit
- COVE
- Character Aware NLM
- ELMO Slides
- Jay Ammar's Visuals on ELMo