Neural Network based Language Model

Presenters: Tianwei Xing, Kaiwen Huang, Yiwen Meng, Jiageng Liu

Example of Language Model (e.g. RNN)

Shakespeare samples generator:

- Concatenate all works of Shakespeare - 10,000 character sample into one file
- Train a 3-layer RNN with 512 nodes on each hidden layer
- Character based prediction: sampling speaker's names and contents

PANDARUS:
Alas, I think he shall be come approached and the day DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

A Neural Probabilistic Language Model

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, Christian Jauvin

NIPS 2001, JMLR 2003

Tianwei Xing

Background

Curse of
Dimensionality
n=10, |V|=100k,
param = 10^50

- Naive Probability Model: $P(W) = P(w_1, w_2, ..., w_{t-1}, w_T)$
- **Conditional probability** of upcoming word: $P(w_T | w_1, w_2, ..., w_{t-1})$
- Chain Rule: $P(w_1, w_2, ..., w_{t-1}, w_T) = P(w_1)P(w_2 | w_1)P(w_3 | w_1, w_2)...P(w_T | w_1, w_2, ..., w_T)$

$$P(w_1, w_2, ..., w_{t-1}, w_T) = \prod_{t=1}^{T} P(w_t | w_1, w_2, ..., w_{t-1})$$

- (n-1)th order **Markov assumption**: $P(w_1, w_2, ..., w_{t-1}, w_T) \approx \prod_{t=1}^{T} P(w_t | w_{t-n+1}, w_{t-n+2}, ..., w_{t-1})$
- **N-gram**

$$P(w_1, w_2, ..., w_{t-1}, w_T) = \prod_{t=1}^{T} P(w_t | w_1, w_2, ..., w_{t-1}) \approx \prod_{t=1}^{T} P(w_t | \mathbf{w}_{t-n+1}^{t-1})$$

Background

<u>Limitations of N-gram:</u>

Calculated from n-gram frequency counts: $P(w_i | w_{i-(n-1)}, \dots, w_{i-1}) = \frac{count(w_{i-(n-1)}, \dots, w_{i-1}, w_i)}{count(w_{i-(n-1)}, \dots, w_{i-1})}$

(Conditional likelihood of seeing a sub-sequence of length *n* in available training data)

Limitation: (discrete model ---- each word is a token)

- Incomplete coverage of the training dataset

  Vocabulary of size V words: V^n possible n-grams (exponential in *n*)

- Semantic similarity between word tokens is not exploited

the cat sat on the **rug** $P(w_t | \mathbf{w}_{t-5}^{t-1}) = ?$

my cat sat on the **mat** $P(w_t | \mathbf{w}_{t-5}^{t-1}) = ?$

Workarounds:

- Smoothing, interpolation, back-off etc.

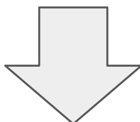
Continuous space language model

Ideas:

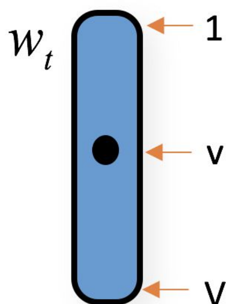
- Words mapped to vectors in a **low-dimensional space**
  - A word w is associated with a distributed feature vector (a real-valued vector in $\mathbb{R}^m$)
- **Vector-space representation** enables semantic/syntactic **similarity** between words/sentences
- NN express the joint probability func of word sequences in terms of word embeddings.
- **Learn simultaneously** the word feature vector and the parameters of model
  - A distributed representation for each word: distributed word feature vector
  - The probability func for word sequences, expressed in terms of these representations
- **Generalization** can be obtained

Vector-space representation & formulation

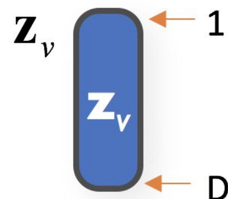
Originally:
"**One-hot**" vector
Representation of a word token at position *t* in the text corpus, with **vocabulary of size *V***



Mapping C



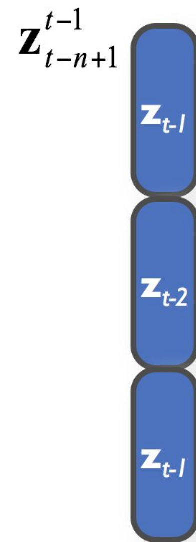
Real-value low dimensional representation
Represent any word *v* in the vocabulary using a vector of **dimension *m***



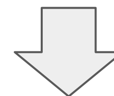
Input:
word history
(**n-1 words**)

Output:
target word (**one-hot** or **vector representation**)

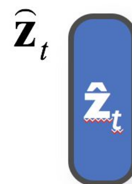
Objective: model



Input:
Vector-space representation of the tth **word history**:
e.g., concatenation of *n*-1 vectors of size *D*



Function g



Output:
Vector-space representation of the prediction of **target word w_t**
(we predict a vector of size *D*)

NPL Model formulation

System diagram

MLP model(1 hidden 1 direct connect)
Parameter set: $\boldsymbol{\Theta} = (\mathbf{C}, \boldsymbol{\theta})$
Normalized prob:

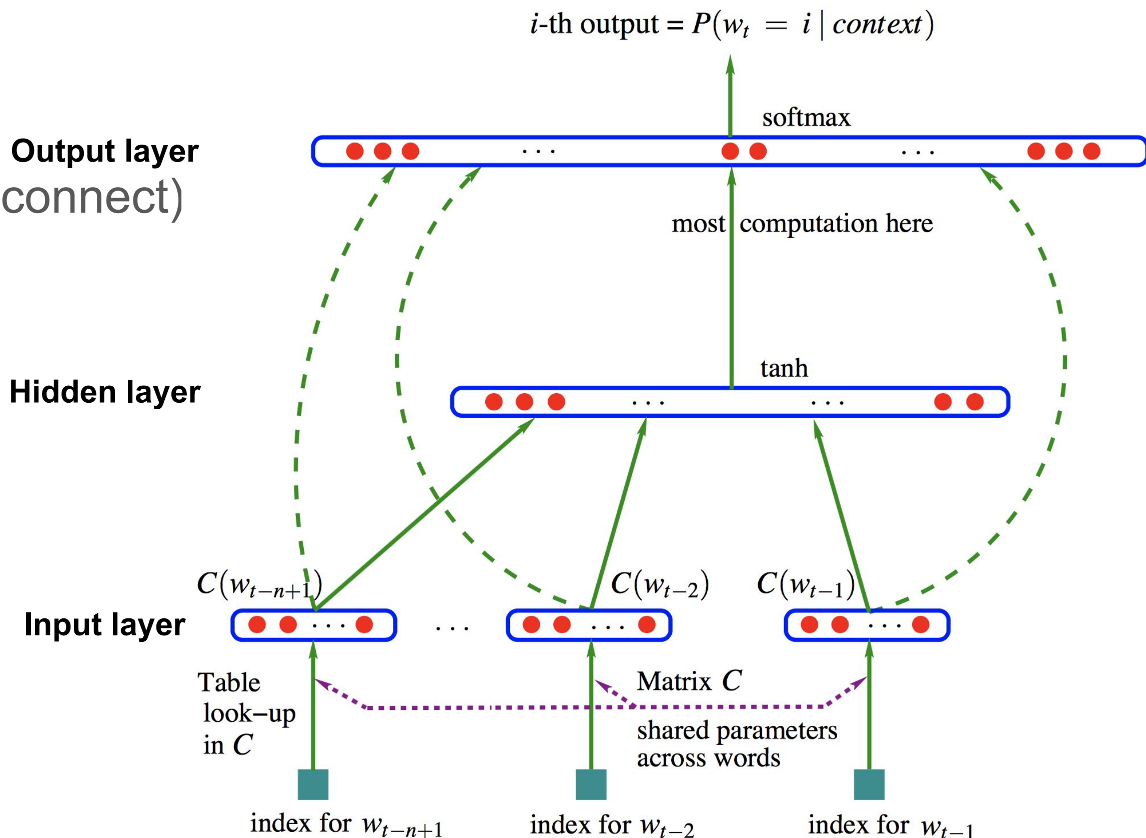
$$P\left(w_t = w \mid \mathbf{w}_1^{t-1}\right) = \frac{e^{s_\theta(w)}}{\sum_{v=1}^{V} e^{s_\theta(v)}}$$

Loglikelihood

$$L_t = \log P\left(w_t = w \mid \mathbf{w}_1^{t-1}\right) = s_\theta(w) - \log \sum_{v=1}^{V} e^{s_\theta(v)}$$

Loss function: +reg

$$\lambda||\theta||^2$$



NPL Model Computation

Number of free parameters

≅|V|(nm+h)

Scales linearly with V and n.

Large Model : speedup

- Distributed computing.
- Short list
- Table look-up
- Initialization

$$y = b + Wx + U\tanh(d + Hx)$$

Output layer

$U \in R^{|V|\times h}$

$b \in R^{|V|}$

$W \in R^{|V|\times(n-1)m}$

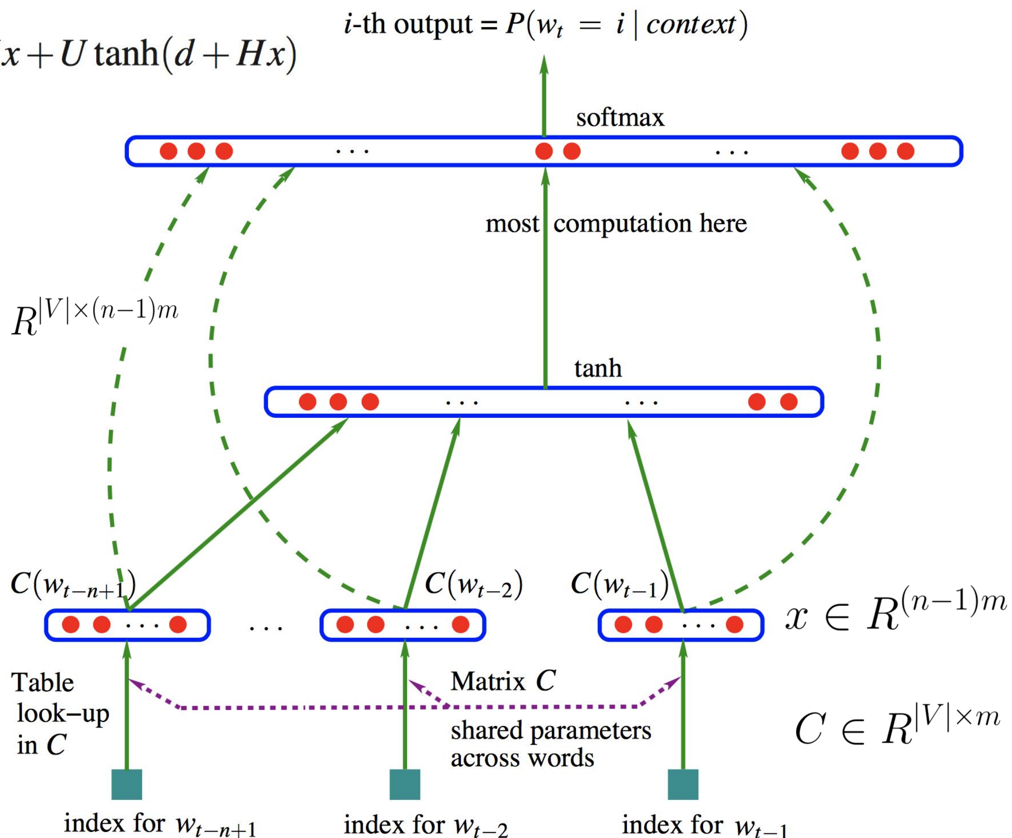
Hidden layer

$H \in R^{h\times(n-1)m}$

$d \in R^{h}$

Input layer

$$x = (C(w_{t-1}), C(w_{t-2}), \cdots, C(w_{t-n+1})).$$



Simulation result

- The neural network performs much better than the smoothed trigram.
- Metric: perplexity

- More context is useful
- Hidden units help
- Learning word features jointly is important

	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	**252**
Del. Int.	3						31	352	336
Kneser-Ney back-off	3							334	323
Kneser-Ney back-off	4							332	321
Kneser-Ney back-off	5							332	321
class-based back-off	3	150						348	334
class-based back-off	3	200						354	340
class-based back-off	3	500						326	**312**
class-based back-off	3	1000						335	319
class-based back-off	3	2000						343	326
class-based back-off	4	500						327	312
class-based back-off	5	500						327	312

Contribution and limitation

- Successfully applies NN to language modeling problem
- Learn embeddings and model params jointly.

- Computationally expensive to train
- Bottleneck: need to evaluate probability of each word over the entire vocabulary
- Very long training time (days, weeks)

3 weeks of training (40 CPUs) on
14,000,000 words training set |V|=17964

- Ignores long-range dependencies
- Fixed time windows
- RNN?

Long-Short Term Memory Model

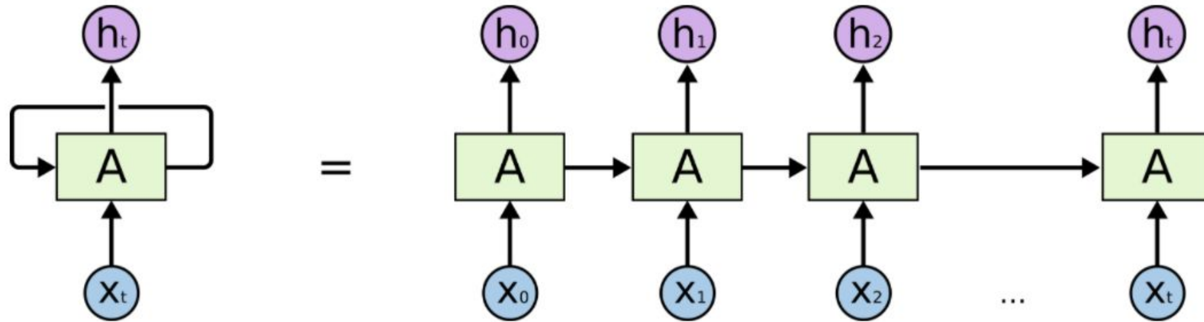
Sepp Hochreiter, Jürgen Schmidhuber

Kaiwen Huang

RNN (Recurrent Neural Network)

What's special about RNN: (from traditional NN)

- Allow sequences of vectors for input and output, no requirement on size.
- Address the issue of hidden state dependency -- Use reasoning from previous events

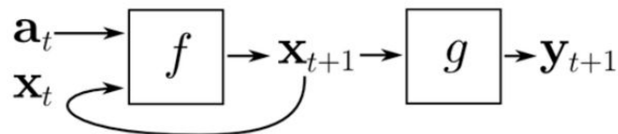


Training RNN - BPTT

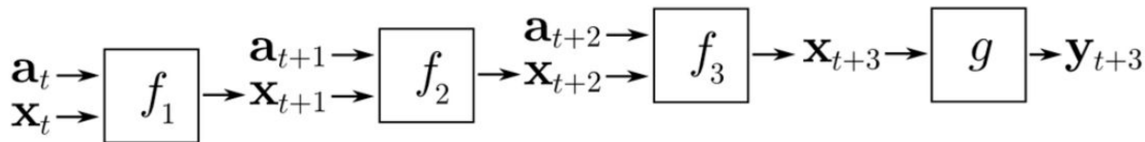
- BPTT -- Backpropagation Through Time
- Training:
  - Training data:

$$\langle \mathbf{a}_0, \mathbf{y}_0 \rangle, \langle \mathbf{a}_1, \mathbf{y}_1 \rangle, \langle \mathbf{a}_2, \mathbf{y}_2 \rangle, \ldots, \langle \mathbf{a}_{k-1}, \mathbf{y}_{k-1} \rangle$$

  - Unfolding a recurrent neural network in time



unfold through time



BPTT

- Training cost:
  - average of costs from each of the time steps
  - Cost from each time step can be computed separately
- **Pros:** Faster for training RNN than general optimization techniques
- **Cons:** More frequent local optima problems than feed-forward neural network

Success of RNN and Limitation

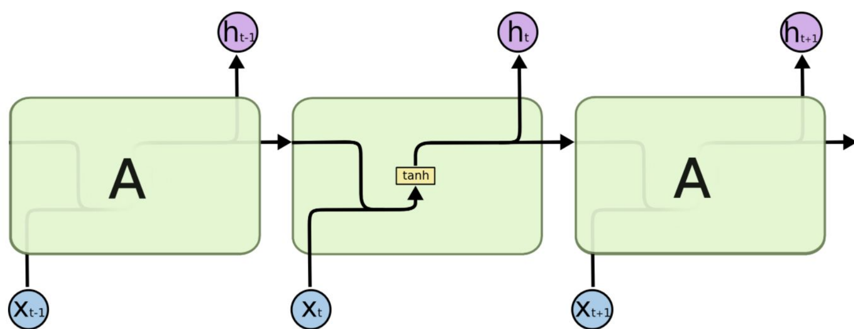
- RNN has been successful in a great many applications
    - speech recognition, translation, image captioning

Limitation of RNN in **long-term dependency**

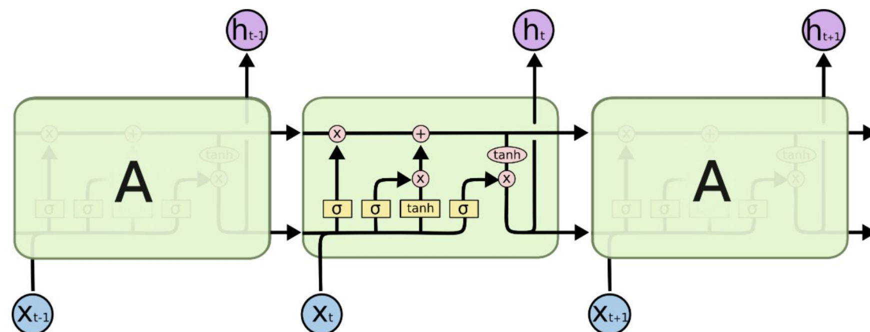
- Sometimes we only need recent previous information, sometimes further back in time
- RNN loses connection to information with larger gaps
- E.g.
    - the clouds are in the *sky*
    - I grew up in France… I speak fluent *French*.

LSTM - an improved RNN

- LSTM -- Long Short Term Memory Network
- LSTM is capable of learning long-term dependencies
    - Remembering information for long periods of time
    - Introduced by **Hochreiter & Schmidhuber (1997)**, were refined and popularized later



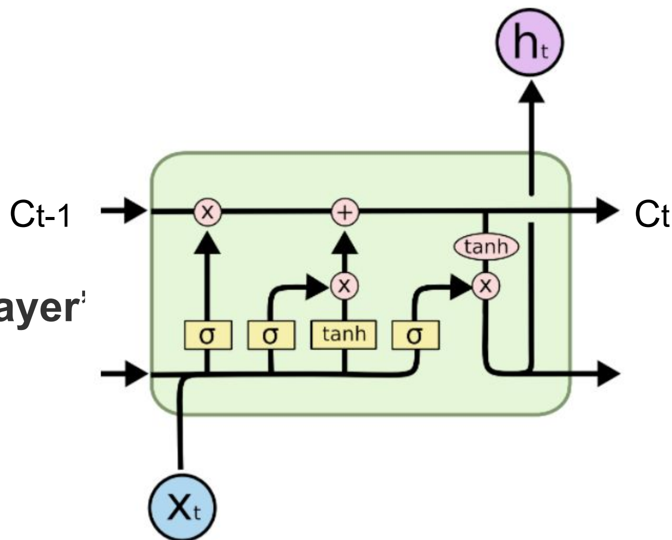
Standard RNN

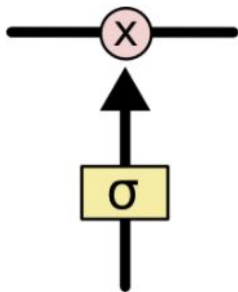


LSTM

LSTM Structure

- Core Idea:
  - Cell State: Ct-1, Ct
  - Gates:
    - Remove or add information to the cell state
    - Composed of:
      - Sigmoid neural net layer -- **“Forget gate layer”**
      - A pointwise multiplication operation

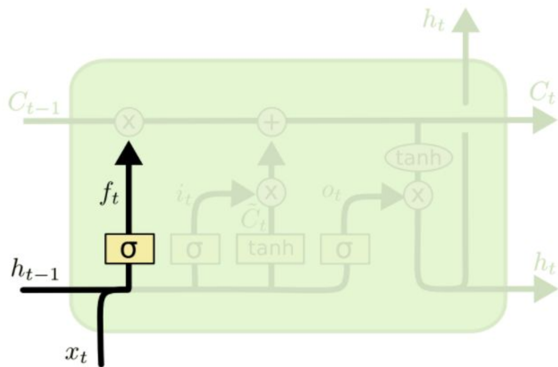




Gate Structure

A Work through of a LSTM module

- Step 1: Determine what information to keep/forget
    - Output a number between 0 and 1 to indicate how much info to keep forget



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

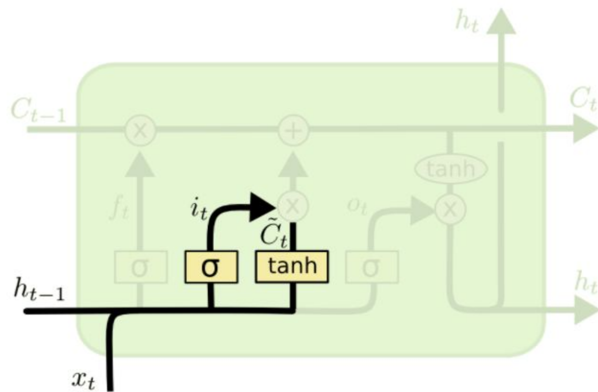
LSTM steps

- Step 2: Decide what information to store in current cell state
  - A sigmoid layer -- **“input gate layer”,** to determine which values we will update

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

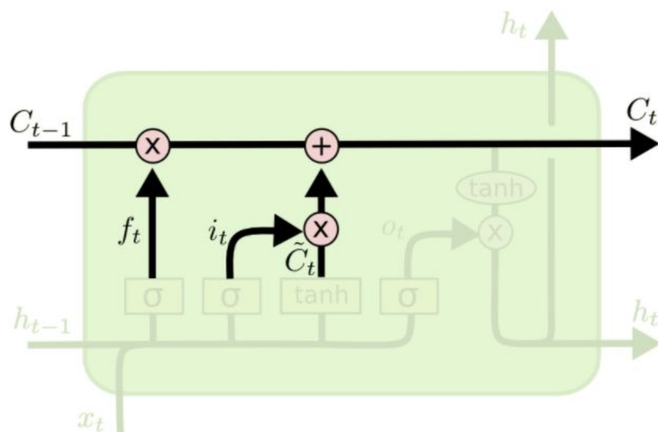
  - A tanh layer creates a vector of new candidates values that could be added to the state

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$



LSTM steps

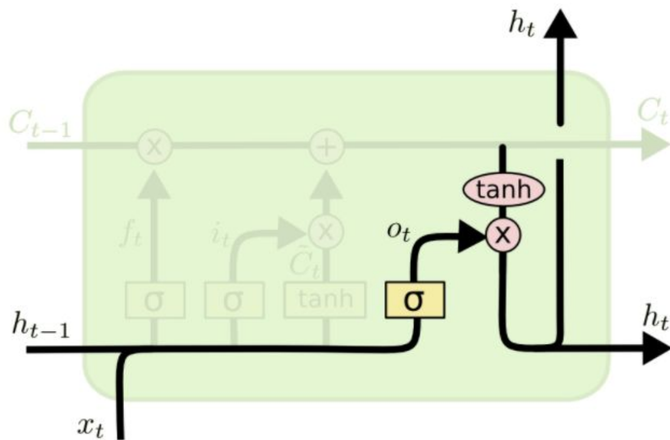
- Step 3: Update cell state
  - Forget things that we decided to forget
  - Add new candidate values scaled by how much we decided to update each state



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM steps

- Step 4: Decide what to output
  - A sigmoid layer to decide what parts of the cell state we want to output
  - Put the cell state through ***tanh*** layer → push values to -1 to 1; and multiply output of the sigmoid layer

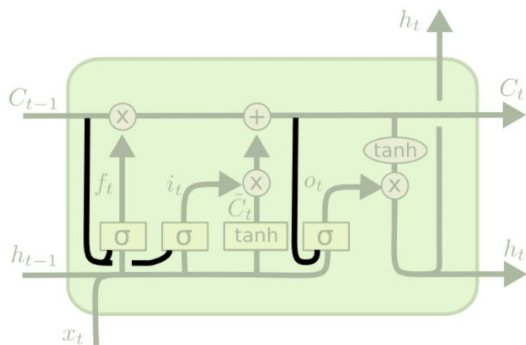


$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

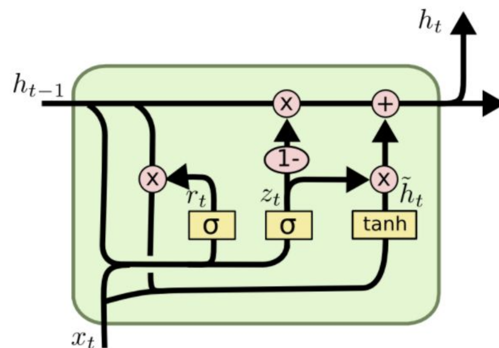
$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM variants and performance

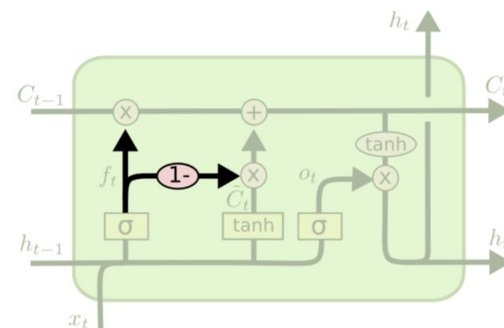
- There are also variants of LSTM:



Gers & Schmidhuber (2000)



Cho, et al. (2014)



Coupled forget and input layers:
Only forget when we are going to put new things
Only input new values when we we forget something older

Character-Word (CW) LSTM Language Model (LM)

Lyan Verwimp, Joris Pelemans, Hugo Van hamme, Patrick Wambacq

2017 Annual Conference of Computational Linguistics

Yiwen Meng

Drawbacks of Current LSTM LM

- ➢ Requires lots of training to optimize parameters for infrequent words
- ➢ Models do make use of internal structure of words
- ❖ Example: “Felicity” → Happiness
- ❖ Out of vocabulary (OOV)
- ❖ Suffix: “ity” → input vector → noun
- ❖ Subword information is significant in performance of LM → Character

Current work of RNN LMs

- Replace word embedding entirely by character in neural machine translation (NML) (Ling et al.,2015 and Costa-juss`a and Fonollosa, 2016)
- Subword-level encoder and a character-level decoder for NMT (Chung et al.,2016)
- In dependency parsing, achieve improvements by generating character-level embeddings with a bidirectional LSTM (Ballesteros et al.,2015)
- Kim et al. (2016) achieve state-of-the-art results in language modeling for several languages by combining a character-level CNN with highway (Srivastava et al., 2015) and LSTM layers
- Chen et al. (2015) and Kang et al. (2011) work on models combining words and Chinese characters to learn embeddings

Character-Word (CW) LSTM LM

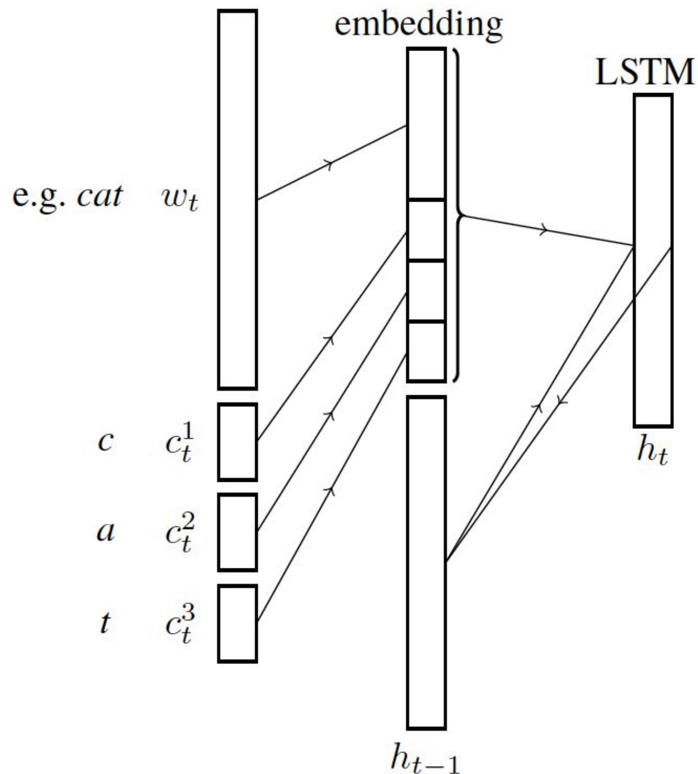
$$\mathbf{e}_t = \mathbf{W}_w \times \mathbf{w}_t$$

$$\mathbf{e}_t^\top = [(\mathbf{W}_w \times \mathbf{w}_t)^\top (\mathbf{W}_c^1 \times \mathbf{c}_t^1)^\top (\mathbf{W}_c^2 \times \mathbf{c}_t^2)^\top \ldots (\mathbf{W}_c^n \times \mathbf{c}_t^n)^\top]$$

- w_t: one column vector of encoded word at time t
- w_w: word embedding matrix
- e_t: word embedding as input for LSTM

- c_t^1: one column vector encoding of first character added, n characters in total
- w^1_c: word embedding matrix for that character
- e_t: word-character embedding as input for LSTM

Character-Word (CW) LSTM LM



- Concatenate character and word embeddings to feed into LSTM, preserve the order of characters implicitly
- Fix the number of characters to n. If C > n, only keep the first/last n characters. If C < n, padded with a special symbol
- Keep the order of characters in both forward (prefix) and backorder (suffix) based on the need
- Character embedding has much smaller size, thus, leading to small embedding matrix

Example: Multidimensional

Character-Word (CW) LSTM LM

$$\mathbf{e}_t^\top = [(\mathbf{W}_w \times \mathbf{w}_t)^\top (\mathbf{W}_c^1 \times \mathbf{c}_t^1)^\top (\mathbf{W}_c^2 \times \mathbf{c}_t^2)^\top \dots (\mathbf{W}_c^n \times \mathbf{c}_t^n)^\top]$$

$\rightarrow$

$$\mathbf{e}_t^\top = [(\mathbf{W}_w \times \mathbf{w}_t)^\top (\mathbf{W}_c \times \mathbf{c}_t^1)^\top (\mathbf{W}_c \times \mathbf{c}_t^2)^\top \dots (\mathbf{W}_c \times \mathbf{c}_t^n)^\top]$$

- Weight share between matrix for characters, total number in vocabulary is the same → Shrink the size of parameters
- Both weight sharing and unsharing are tested

Size of Parameter

$V \times E$ → $V \times (E - n \times E_c) + n \times (C \times E_c)$ → $V \times (E - n \times E_c) + C \times E_c$

Word embedding

Character- Word embedding

Character- Word embedding with weight sharing

V: vocabulary size >> C: character size → Shrink embedding size

Test CW LSTM Model

- Tensor flow
- small model: 2 hidden layers, 200 units
- large model: 2 hidden layers, 560 units

	Training	Validation	Test	Character Size
English(PTB)	900K	70K	80K	48
Dutch (CGN)	1.4M	180K	190K	88

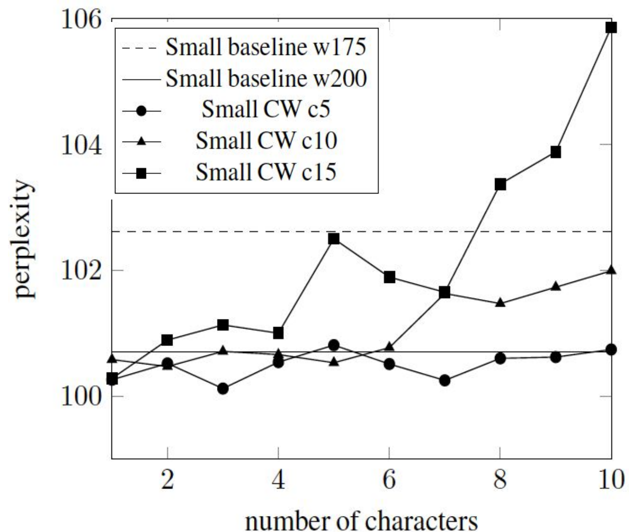
Baseline I: same hidden units

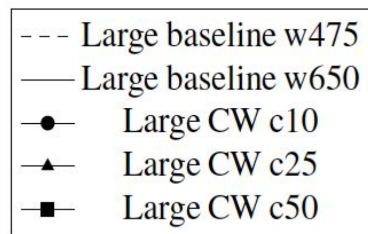
Hidden units	Word model	C-W model
Small	200	200
Large	650	650

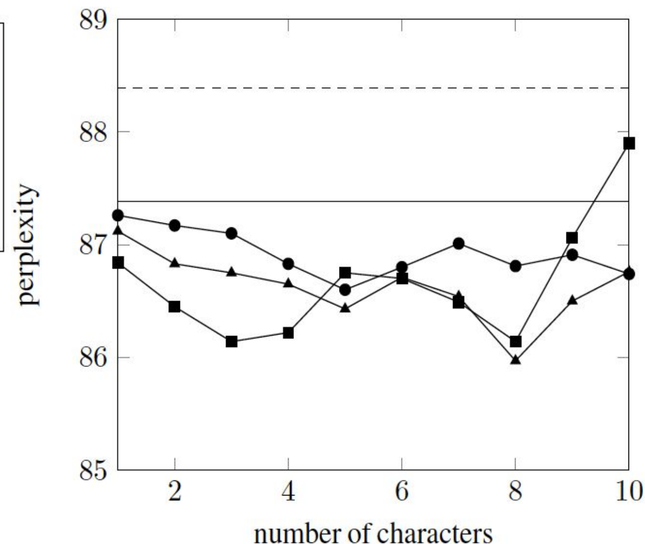
Baseline II: Approximately same parameters

Hidden units	Word model	C-W model
Small	200	175
Large	650	475

Results: Small model, Large model (validation)



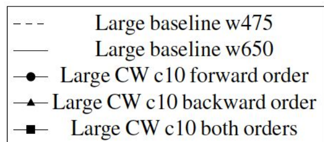


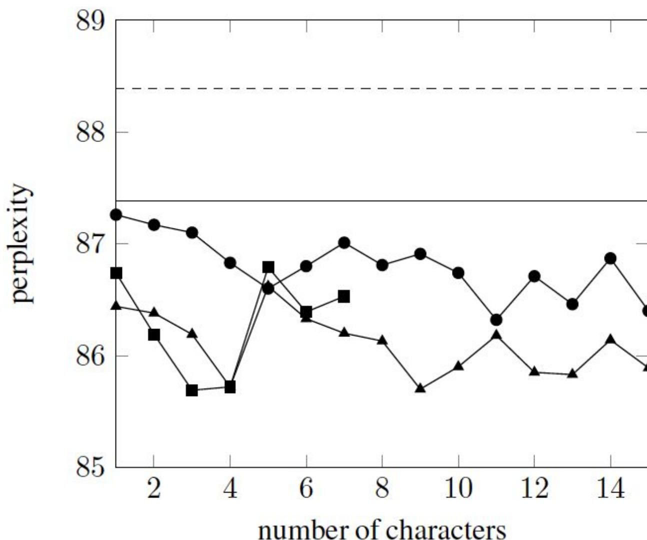


- Performance of CW models is significantly higher than word models for same hidden units
- In small models, with same number of parameters, performance of CW models varies based on number of characters, and size of embedding
- For large models, with same number of parameters, almost all CW models performs better than word models

Results: Order of Large CW Models

Validation





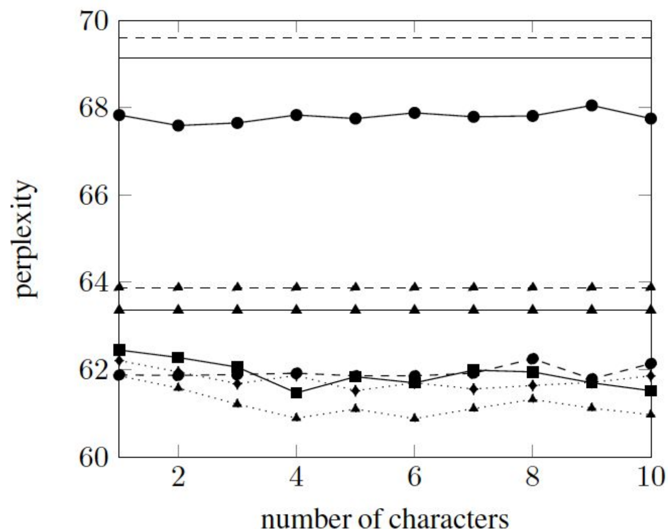
Test

Small model	**Perplexity**
Baseline w175/w200	98.82/96.86
(Kim et al., 2016)	100.3
(Miyamoto and Cho, 2016)	109.05
c5 with $n=3$	**96.21**
c5 with $n=7$	96.35
Large model	**Perplexity**
Baseline w475/w650	84.38/83.6
(Kim et al., 2016)	84.6
c25 with $n=8$	82.69
c10 with $n=9(b)$	82.68
c10 with $n=3+3(b)$	**82.04**

- Backorder CW models performs the best, while increase number of character in both forward and back order would decrease the performance for large n
- In small models, 3 character with embedding size of 5 performs the best
- In large models, 6 characters with 3 in forward order and 3 in backward order performs the best

Results: Dutch

Validation



Test

Small model	**Perplexity**
Baseline w175/w200	76.78/76
c10 with $n=2$	75.23
c10 with $n=3$	**75.04**
Large model	**Perplexity**
Baseline w475/w650	70.88/70.69
c25 with $n=4$	68.79
c25 with $n=6(b)$	**67.64**

- In both small and large models, the performance of CW models are significantly higher for both same number of hidden units and size of parameter
- In the test set, 3 character with embedding size of 10 is best in small models while 6 character with backorder of embedding size of 25 is the best in large models

Results: Share Weight

		Relative change in valid perplexity w.r.t. Baseline	Char-Word
PTB	small c10	0.53 (0.88)	0.19 (0.67)
	large c10	- 0.54 (0.37)	- 0.02 (0.22)
CGN	small c10	- 1.70 (0.34)	0.24 (0.30)
	large c10	- 2.10 (0.32)	0.15 (0.50)

- Results are averaged over number of character from 1 to 10
- Number in the bracket is standard deviation
- CW models with weight sharing are better than baseline word models but are not different for CW models
- Meaning that the position of each character has the significance

Conclusion

- ❖ Subword information is also an important factor for LM, so concatenate character and word embedding
- ❖ CW models can both reduce the size of parameter matrix and increase the performance
- ❖ Preserve the order of characters in each word plays an important role in LSTM LM
- ❖ Results show characters can convey different meanings based on the position, which indicates the decision of weight sharing for each language

Regularizing and Optimizing LSTM Language Models

Stephen Merity, Nitish Shirish Keskar, Richard Socher

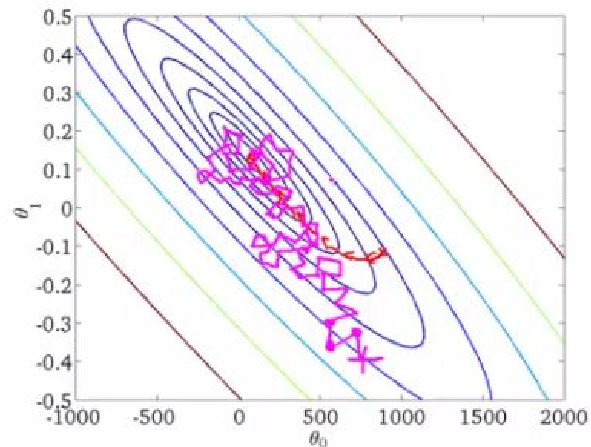
Salesforce Research

Jiageng Liu

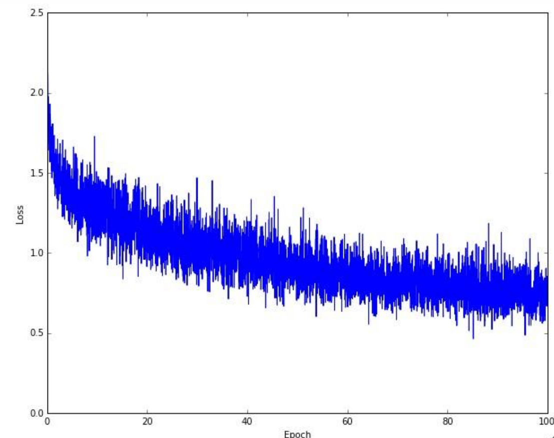
Train LSTM with SGD

- **S**tochastic **G**radient **D**escent (SGD)
  - In each training iteration…
    - take one random data and update one gradient step
    - using the random approximation of the true gradient

$$x_{k+1} = x_k - \eta_k \widehat{\nabla f}(x_k)$$



- Good side
  - Fast (no traversing the whole dataset)
  - Avoid local minima/saddle points (due to the randomness)
  - Better generalization (avoid overfitting the training dataset)
- Bad side
  - Result keeps wiggling near the optimal



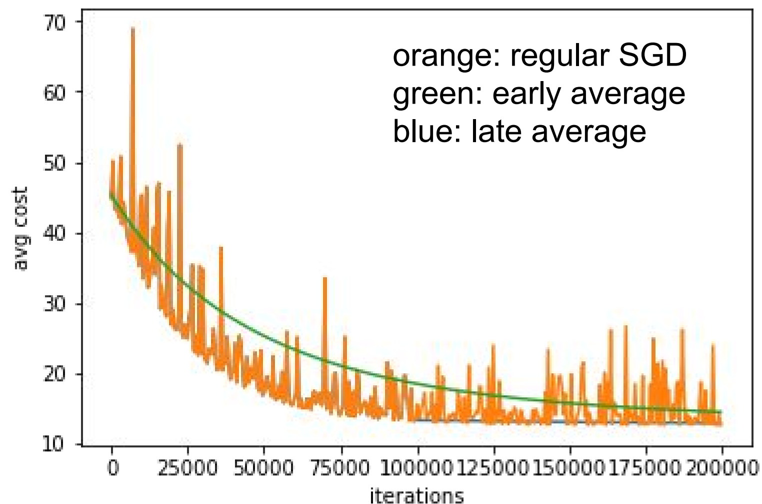
Averaged **SGD**

- Idea: average up the wiggles

$$x_{k+1} = x_k - \eta_k \widehat{\nabla} f(x_k)$$

$$\overline{x}_{k+1} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

  - reduces the variance of the iterates
  - better estimate of the global optimal
  - **proved** to achieve the best possible convergence without additional info (Polyak 1992)
- Problem: when to start averaging?
  - too late - not enough acceleration
  - too early - introduce "bad" iterates at the start
  - idea: when the loss function starts to plateau



Non-monotonically **T**riggered **ASGD**

- Idea: record whether the loss (perplexity) has stopped dropping
  - however, stochasticity may cause the loss to fluctuate anyway
  - algorithm: check if the loss decreases every several iterates
  - specific strategy may vary

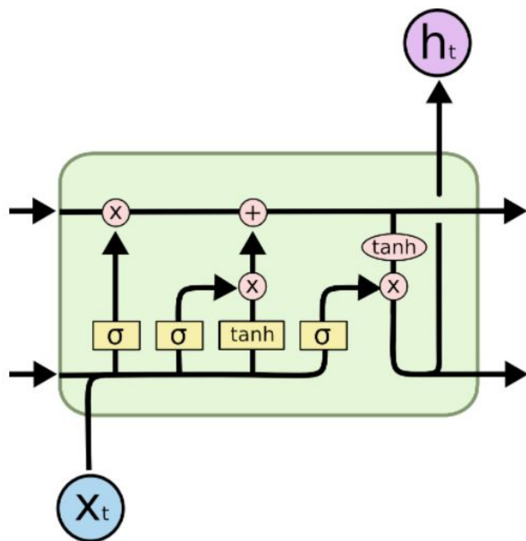
Algorithm 1 Non-monotonically Triggered ASGD (NT-ASGD)

Inputs: Initial point w_0, learning rate γ, logging interval L, non-monotone interval n.

1: Initialize $k \leftarrow 0, t \leftarrow 0, T \leftarrow 0,$ `logs` $\leftarrow$ `[]`
2: **while** stopping criterion not met **do**
3:   Compute stochastic gradient $\hat{\nabla} f(w_k)$ and take SGD step (1).
4:   **if** $\mathrm{mod}(k, L) = 0$ and $T = 0$ **then**
5:     Compute validation perplexity v.
6:     **if** $t > n$ and $v > \min_{l \in \{t-n, \cdots, t\}}$ `logs[l]` **then**
7:       Set $T \leftarrow k$
8:     **end if**
9:     Append v to `logs`
10:     $t \leftarrow t + 1$
11:   **end if**
12: **end while**

return $\frac{\sum_{i=T}^{k} w_i}{(k-T+1)}$

Overfitting



complex structure

$$i_t = \sigma(W^i x_t + U^i h_{t-1})$$

$$f_t = \sigma(W^f x_t + U^f h_{t-1})$$

$$o_t = \sigma(W^o x_t + U^o h_{t-1})$$

$$\tilde{c}_t = \tanh(W^c x_t + U^c h_{t-1})$$

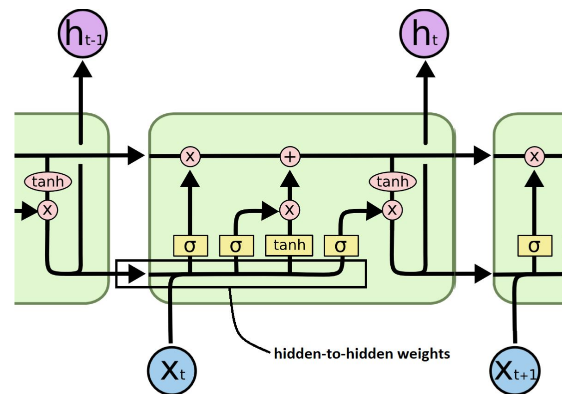
$$c_t = i_t * \tilde{c}_t + f_t * h_{t-1}$$

$$h_t = o_t * \tanh(c_t)$$

many parameters to train
(8 fc matrices in one layer)

Regularize with **DropConnect**

- Idea: **randomly set** some hidden-to-hidden weights to **zeros during training**
- prevent the network from relying on certain neuron weights too much
- In BPTT, the same individual dropped weights remain dropped for the entirety of the forward and backward pass
- focus on dropping recurrent weights which are more likely to “accumulate” overfitting over time



$$i_t = \sigma(W^i x_t + U^i h_{t-1})$$
$$f_t = \sigma(W^f x_t + U^f h_{t-1})$$
$$o_t = \sigma(W^o x_t + U^o h_{t-1})$$
$$\tilde{c}_t = \tanh(W^c x_t + U^c h_{t-1})$$
$$c_t = i_t * \tilde{c}_t + f_t * h_{t-1}$$
$$h_t = o_t * \tanh(c_t)$$

hidden to hidden

Other techniques

- Variable length BPTT
  - batch-SGD training: not backpropagate the information from the starting word to the last batch
  - solution: randomly choose batch sizes
  - tradeoff: too much variability → less efficient training on GPU

  {Four score and seven years ago our fathers brought**}** **{**forth on this continent, a new nation, conceived**}** **{**in Liberty, and dedicated to the proposition that all**}** **{**men are created equal...

- Embedding dropout
  - dropout on the embedding matrix at a word level for regularization
  - remaining embeddings are scaled up to compensate
  - more robust to change of specific words

Other techniques

- Weight tying
  - reuse weights from input word embedding as the output classification (softmax)
  - much fewer parameters to train
  - theoretical motivation

- (Temporal) Activation Regularization
  - Use L_2 decay on
    - the individual unit activations to prevent large spikes (AR)
    - minimizes differences between states to prevent large changes (TAR)
  - only applied to the output of the final RNN layer (not explained in the paper)

Other models improvement

- Neural Cache Model
  - store recent hidden activations and use them as representation for the context
  - exploit the long-range dependency of words in a document
  - “tiger” consists 2.8% of words in the Wikipedia page “tiger”, compared to 0.0037% overall

- Pointer Sentinel Model
  - Incorporate pointer (reference to previous words) and RNN (vocabulary embeddings)
  - Let the pointer (sentinel) decide whether it’s confidence enough to skip scanning the vocabulary
  - Avoid needing to learn to store the identity of the token to be produced
  - Helps solving the rare words/out-of-vocabulary problems

Results (PTB)

Model	Parameters	Validation	Test
Mikolov & Zweig (2012) - KN-5	2M[‡]	–	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M[‡]	–	125.7
Mikolov & Zweig (2012) - RNN	6M[‡]	–	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M[‡]	–	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M[‡]	–	92.0
Zaremba et al. (2014) - LSTM (medium)	20M	86.2	82.7
Zaremba et al. (2014) - LSTM (large)	66M	82.2	78.4
Gal & Ghahramani (2016) - Variational LSTM (medium)	20M	81.9 ± 0.2	79.7 ± 0.1
Gal & Ghahramani (2016) - Variational LSTM (medium, MC)	20M	–	78.6 ± 0.1
Gal & Ghahramani (2016) - Variational LSTM (large)	66M	77.9 ± 0.3	75.2 ± 0.2
Gal & Ghahramani (2016) - Variational LSTM (large, MC)	66M	–	73.4 ± 0.0
Kim et al. (2016) - CharCNN	19M	–	78.9
Merity et al. (2016) - Pointer Sentinel-LSTM	21M	72.4	70.9
Grave et al. (2016) - LSTM	–	–	82.3
Grave et al. (2016) - LSTM + continuous cache pointer	–	–	72.1
Inan et al. (2016) - Variational LSTM (tied) + augmented loss	24M	75.7	73.2
Inan et al. (2016) - Variational LSTM (tied) + augmented loss	51M	71.1	68.5
Zilly et al. (2016) - Variational RHN (tied)	23M	67.9	65.4
Zoph & Le (2016) - NAS Cell (tied)	25M	–	64.0
Zoph & Le (2016) - NAS Cell (tied)	54M	–	62.4
Melis et al. (2017) - 4-layer skip connection LSTM (tied)	24M	60.9	58.3
AWD-LSTM - 3-layer LSTM (tied)	24M	60.0	57.3
AWD-LSTM - 3-layer LSTM (tied) + continuous cache pointer	24M	53.9	52.8

Model Ablation

Remove each one of the techniques to see how worse the model performs.

Model	PTB Validation	PTB Test	WT2 Validation	WT2 Test
AWD-LSTM (tied)	60.0	57.3	68.6	65.8
– fine-tuning	60.7	58.8	69.1	66.0
– NT-ASGD	66.3	63.7	73.3	69.7
– variable sequence lengths	61.3	58.9	69.3	66.2
– embedding dropout	65.1	62.7	71.1	68.1
– weight decay	63.7	61.0	71.9	68.7
– AR/TAR	62.7	60.3	73.2	70.1
– full sized embedding	68.0	65.6	73.7	70.7
– weight-dropping	71.1	68.9	78.4	74.9