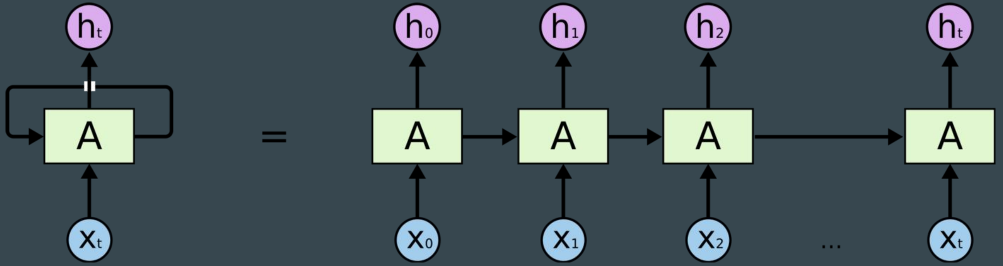
Recurrent Models & Language Modeling

Sepehr Sameni(@Separius)
Deep Learning Group (2nd session)

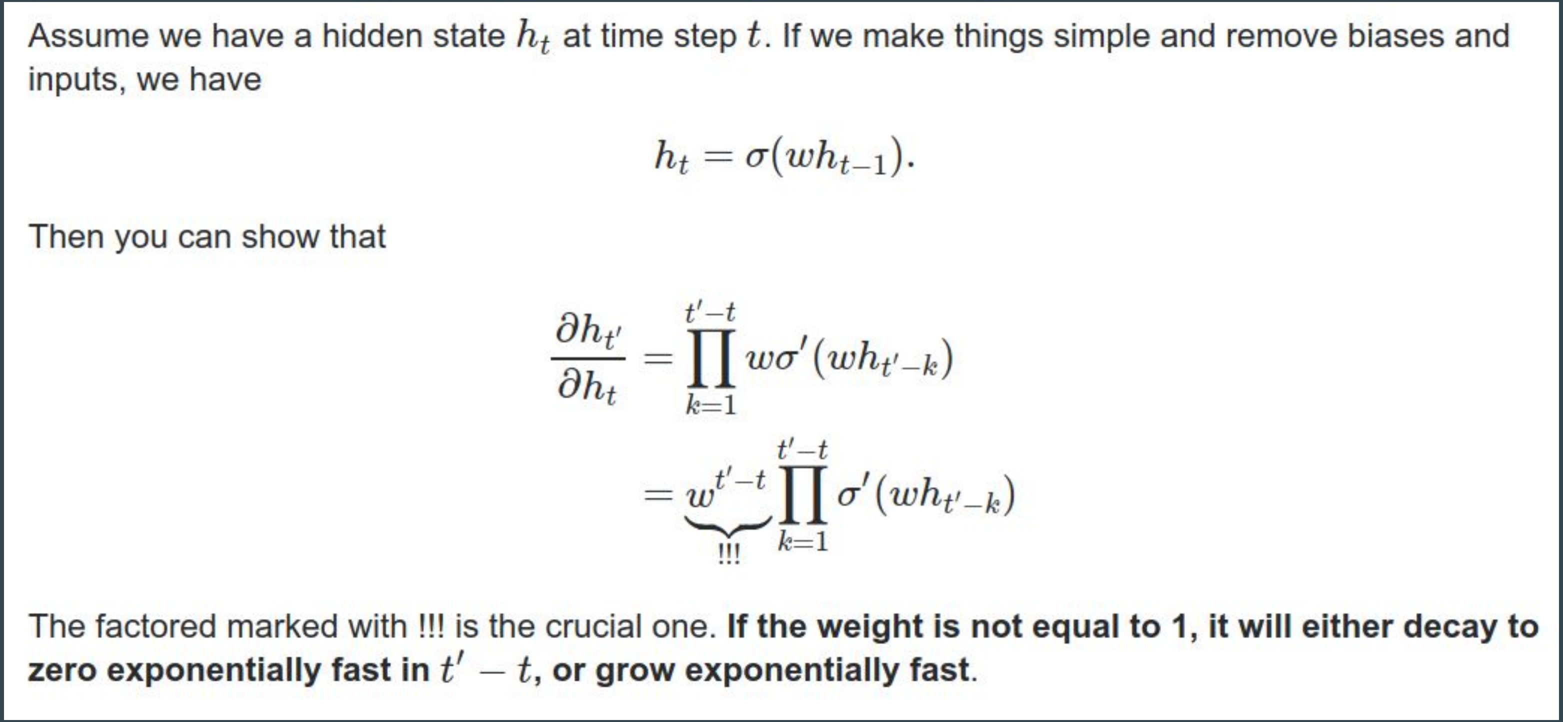
RNNs[1]

- Another instance of structural bias(temporal weight sharing)
- RNNs can approximate any computable function(algorithm) = Universal Turing Machine
- It's possible to use a RNN to explicitly simulate a pushdown automaton with two stacks
- BPTT and teacher forcing



$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}),$$

Vanishing Gradient Problem



Assume we have a hidden state h_t at time step t. If we make things simple and remove biases and inputs, we have

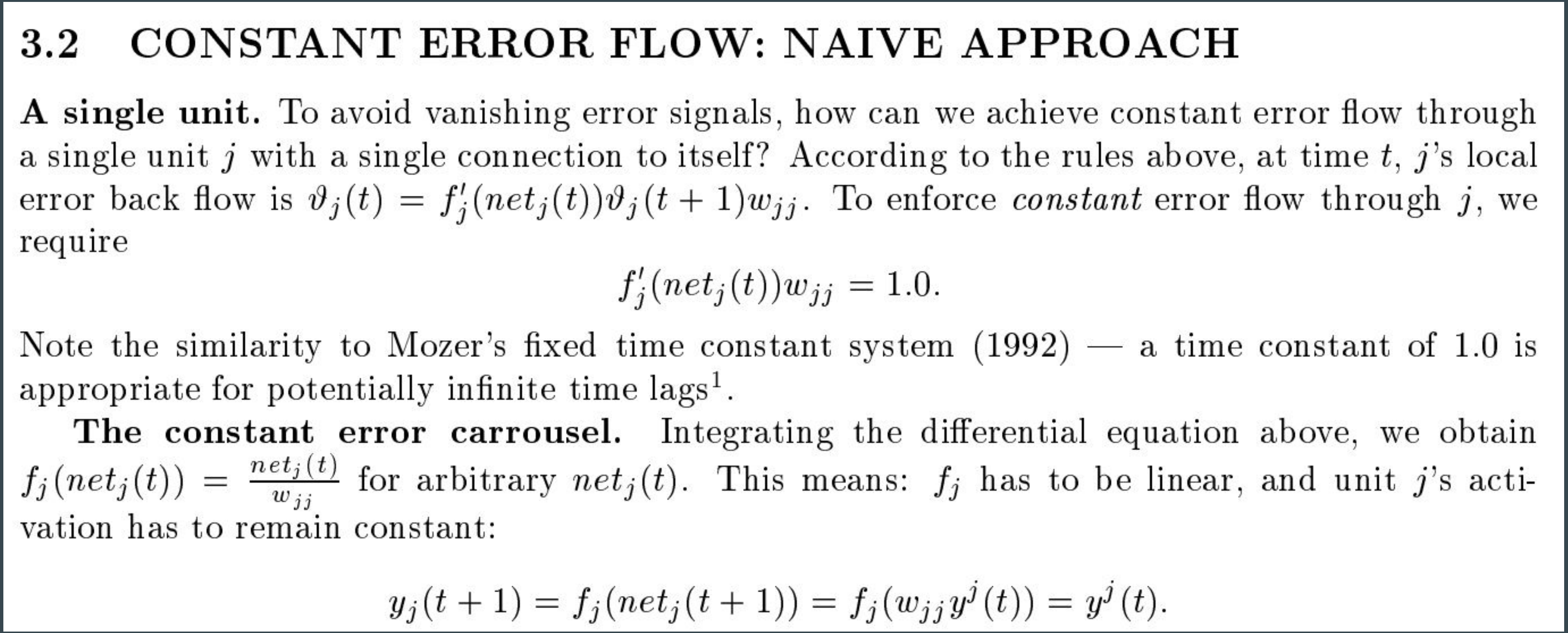
$$h_t = \sigma(w h_{t-1}).$$

Then you can show that

$$\begin{aligned}\frac{\partial h_{t'}}{\partial h_t} &= \prod_{k=1}^{t'-t} w \sigma'(w h_{t'-k}) \\ &= \underbrace{w^{t'-t}}_{!!!} \prod_{k=1}^{t'-t} \sigma'(w h_{t'-k})\end{aligned}$$

The factored marked with !!! is the crucial one. **If the weight is not equal to 1, it will either decay to zero exponentially fast in $t' - t$, or grow exponentially fast**.

LSTM(CEC) [2]



3.2 CONSTANT ERROR FLOW: NAIVE APPROACH

A single unit. To avoid vanishing error signals, how can we achieve constant error flow through a single unit j with a single connection to itself? According to the rules above, at time t, j's local error back flow is $\vartheta_j(t) = f'_j(net_j(t))\vartheta_j(t+1)w_{jj}$. To enforce *constant* error flow through j, we require

$$f'_j(net_j(t))w_{jj} = 1.0.$$

Note the similarity to Mozer's fixed time constant system (1992) — a time constant of 1.0 is appropriate for potentially infinite time lags[1].

The constant error carrousel. Integrating the differential equation above, we obtain $f_j(net_j(t)) = \frac{net_j(t)}{w_{jj}}$ for arbitrary $net_j(t)$. This means: f_j has to be linear, and unit j's activation has to remain constant:

$$y_j(t+1) = f_j(net_j(t+1)) = f_j(w_{jj}y^j(t)) = y^j(t).$$

Improving CEC [2]

- Input weight conflict: not overriding the cell with irrelevant info
- Output weight conflict: protecting output from the memory cell

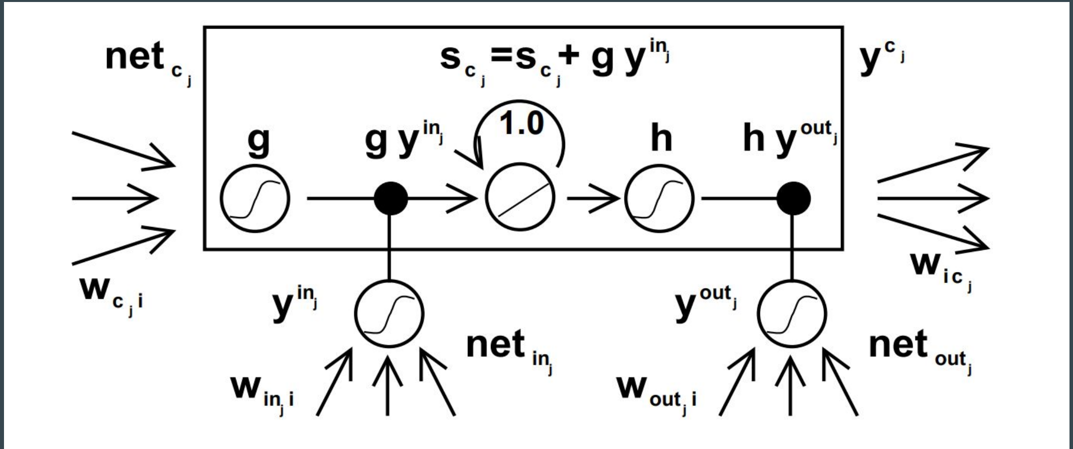


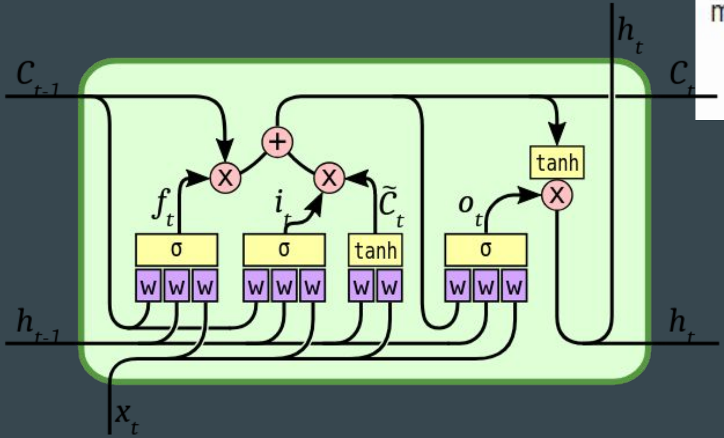
Figure 1: *Architecture of memory cell c_j (the box) and its gate units in_j, out_j. The self-recurrent connection (with weight 1.0) indicates feedback with a delay of 1 time step. It builds the basis of the "constant error carrousel" CEC. The gate units open and close access to CEC. See text and appendix A.1 for details.*

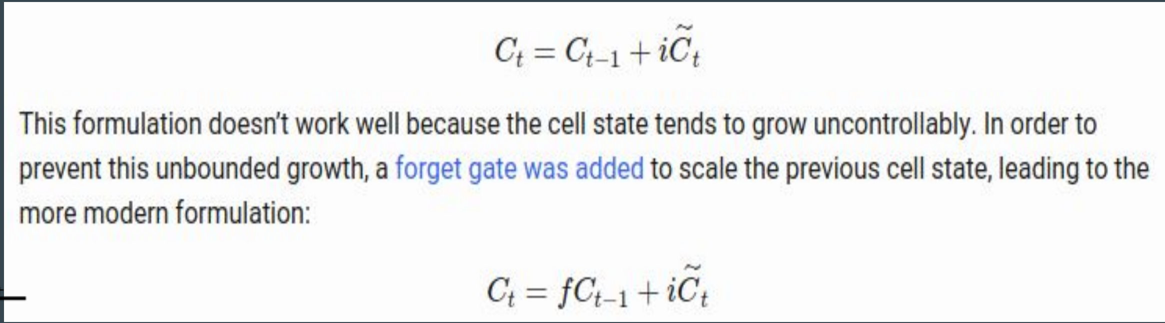
Experiments [2]

- Reber grammar parser (short term)
- (y, a[0:p], y) and (x, a[0:p], x) => (y, rand, y) and (x, rand, x) => rand length
- (N*[1/0], T*noise) => noisy target
- Adding Problem
- Multiplication Problem

Changes to LSTM [3]

- No truncation (dCt/dCt-1 != 1)
- Peephole connection
- Forget Gate



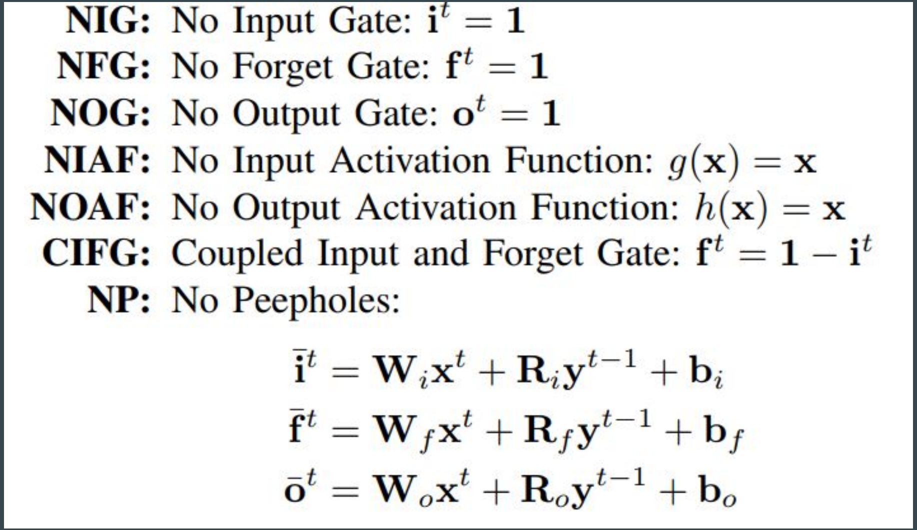


$$C_t = C_{t-1} + i\tilde{C}_t$$

This formulation doesn't work well because the cell state tends to grow uncontrollably. In order to prevent this unbounded growth, a forget gate was added to scale the previous cell state, leading to the more modern formulation:

$$C_t = fC_{t-1} + i\tilde{C}_t$$

Variants [3]



NIG: No Input Gate: $\mathbf{i}^t = \mathbf{1}$

NFG: No Forget Gate: $\mathbf{f}^t = \mathbf{1}$

NOG: No Output Gate: $\mathbf{o}^t = \mathbf{1}$

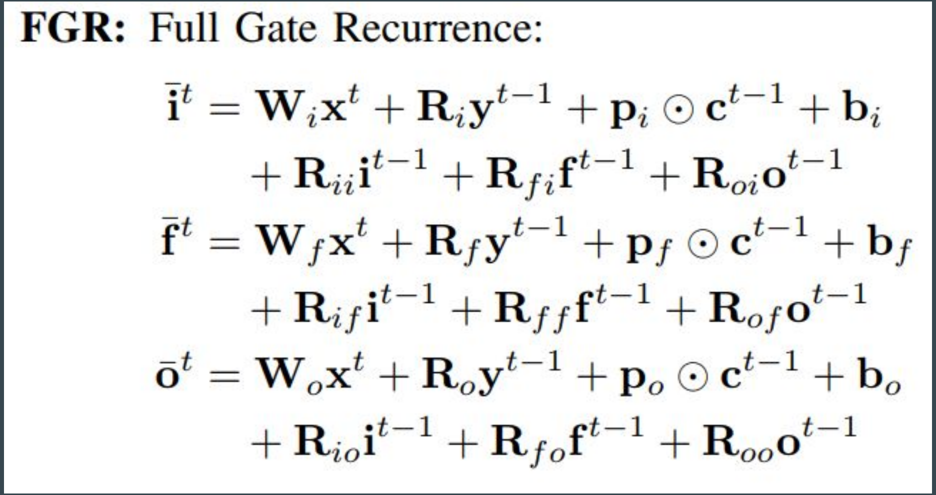
NIAF: No Input Activation Function: $g(\mathbf{x}) = \mathbf{x}$

NOAF: No Output Activation Function: $h(\mathbf{x}) = \mathbf{x}$

CIFG: Coupled Input and Forget Gate: $\mathbf{f}^t = \mathbf{1} - \mathbf{i}^t$

NP: No Peepholes:

$$\begin{aligned}
\bar{\mathbf{i}}^t &= \mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{b}_i \\
\bar{\mathbf{f}}^t &= \mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{b}_f \\
\bar{\mathbf{o}}^t &= \mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{b}_o
\end{aligned}$$



FGR: Full Gate Recurrence:

$$\begin{aligned}
\bar{\mathbf{i}}^t = {} & \mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i \\
& + \mathbf{R}_{ii}\mathbf{i}^{t-1} + \mathbf{R}_{fi}\mathbf{f}^{t-1} + \mathbf{R}_{oi}\mathbf{o}^{t-1} \\
\bar{\mathbf{f}}^t = {} & \mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f \\
& + \mathbf{R}_{if}\mathbf{i}^{t-1} + \mathbf{R}_{ff}\mathbf{f}^{t-1} + \mathbf{R}_{of}\mathbf{o}^{t-1} \\
\bar{\mathbf{o}}^t = {} & \mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^{t-1} + \mathbf{b}_o \\
& + \mathbf{R}_{io}\mathbf{i}^{t-1} + \mathbf{R}_{fo}\mathbf{f}^{t-1} + \mathbf{R}_{oo}\mathbf{o}^{t-1}
\end{aligned}$$

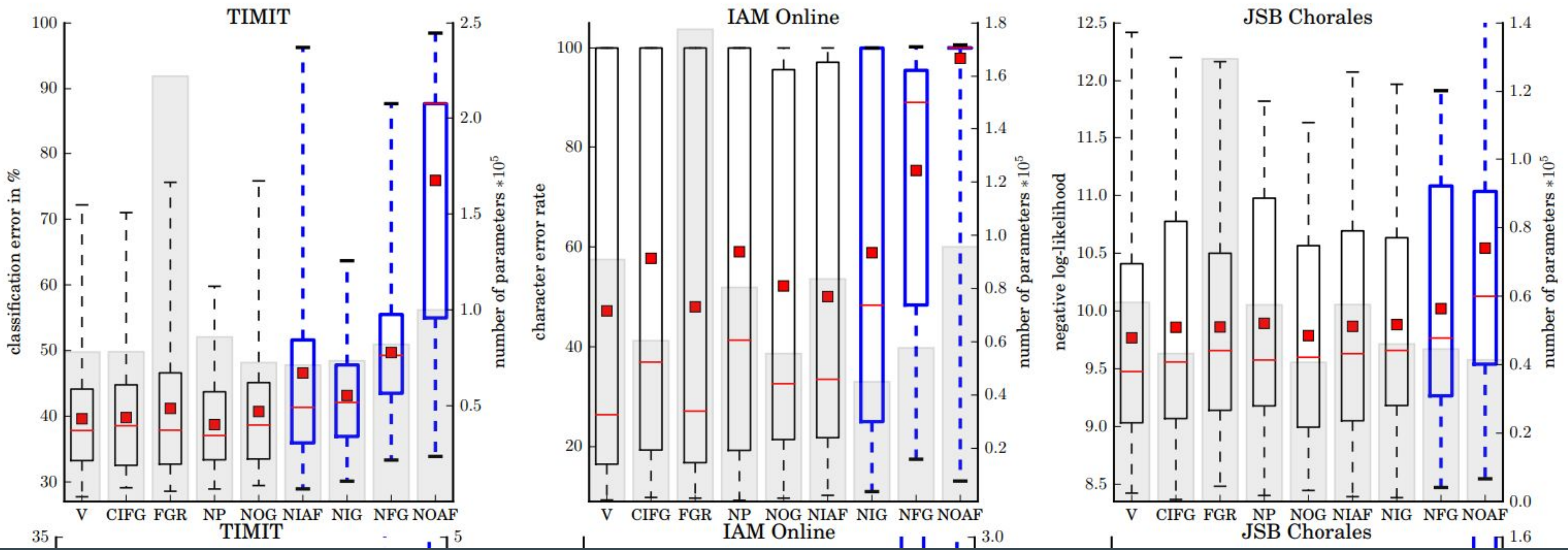
Compare [3]

- TIMIT speech corpus + per frame classification
- IAM handwriting database + CTC for char detection
- JSB Chorales + next note prediction

We performed 27 random searches (one for each combination of the nine variants and three datasets)

- Lstm hidden size log uniform in [20, 200]
- LR log uniform in [10e-6, 10e-2]
- Momentum
- Input noise

Results [3]



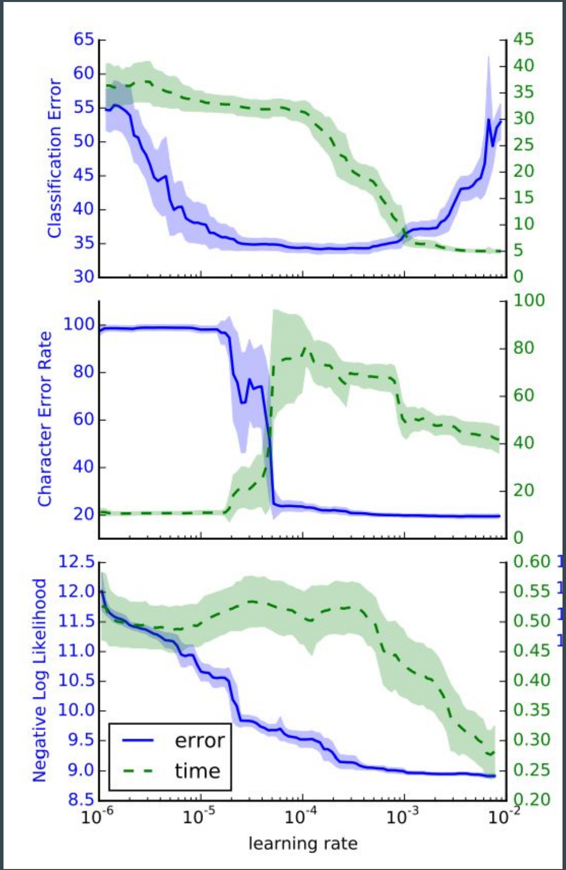
TIMIT
classification error in %
number of parameters *10^5
V CIFG FGR NP NOG NIAF NIG NFG NOAF
IAM Online
character error rate
JSB Chorales
negative log-likelihood

Results Explained(variants) [3]

- removing the output activation function (NOAF) or the forget gate (NFG) significantly hurt performance on all three datasets => coupling helps
- CIFG and NP are both as good as the vanilla version (lower computation)
- FGR adds a lot of parameters and it is discouraged
- NIG, NOG, and NIAF perform poorly on real-valued supervised tasks

Hyperparams Explained [3]

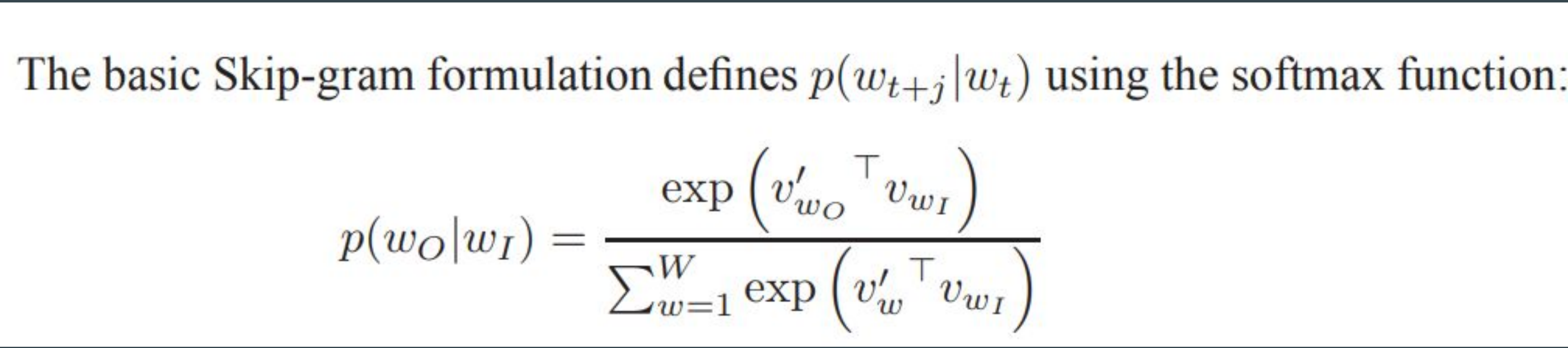
- LR is the most important one
- Gaussian noise is actually bad
- Momentum is not that important
- Larger Hidden size is better but with diminishing returns (and higher T)
- We can set LR with a tiny net and then increase the hidden size



W2V [4, 5]

Prev work: language modeling, usage of word vectors in NLP, SENNA, BoW, linguistics

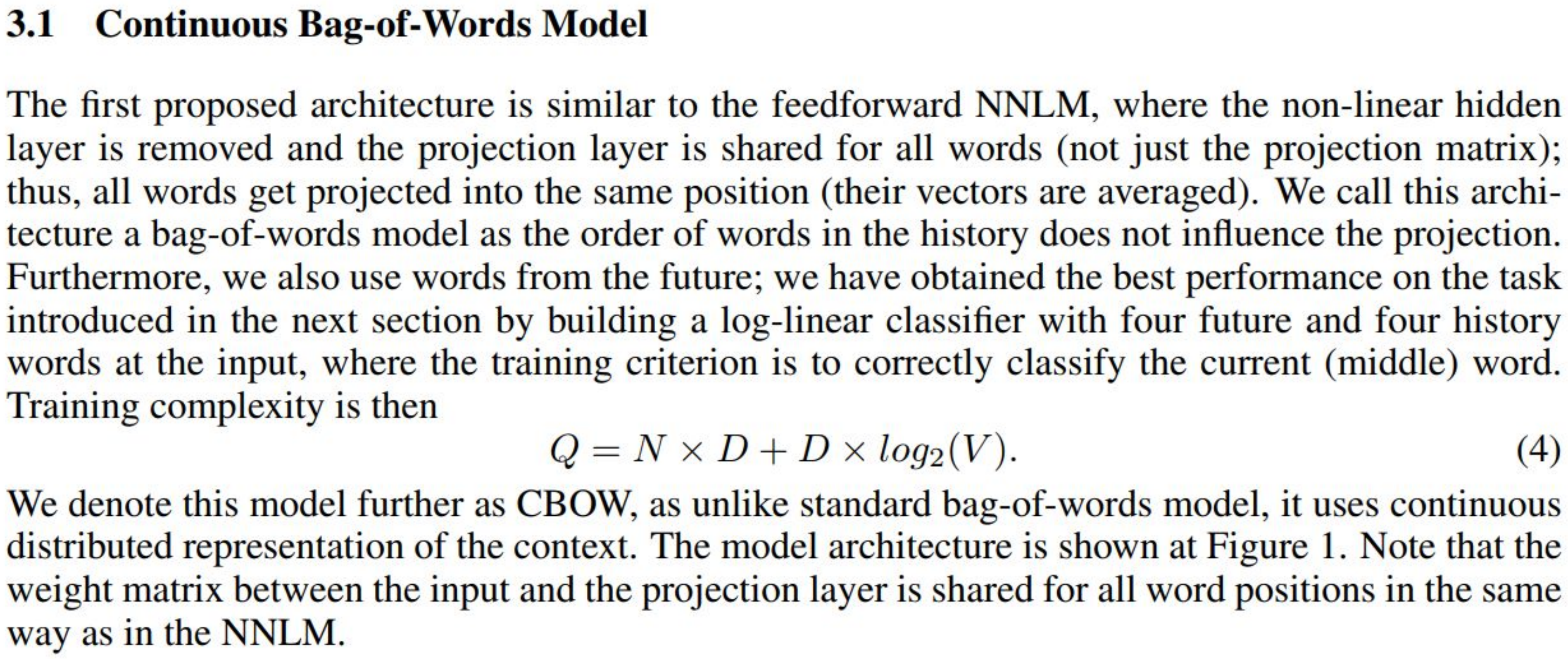
This work: no hidden layer to reduce the compute power + use of huffman coding => later: negative sampling



The basic Skip-gram formulation defines $p(w_{t+j}|w_t)$ using the softmax function:

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^\top v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'_w}^\top v_{w_I}\right)}$$

CBOW [4, 5]



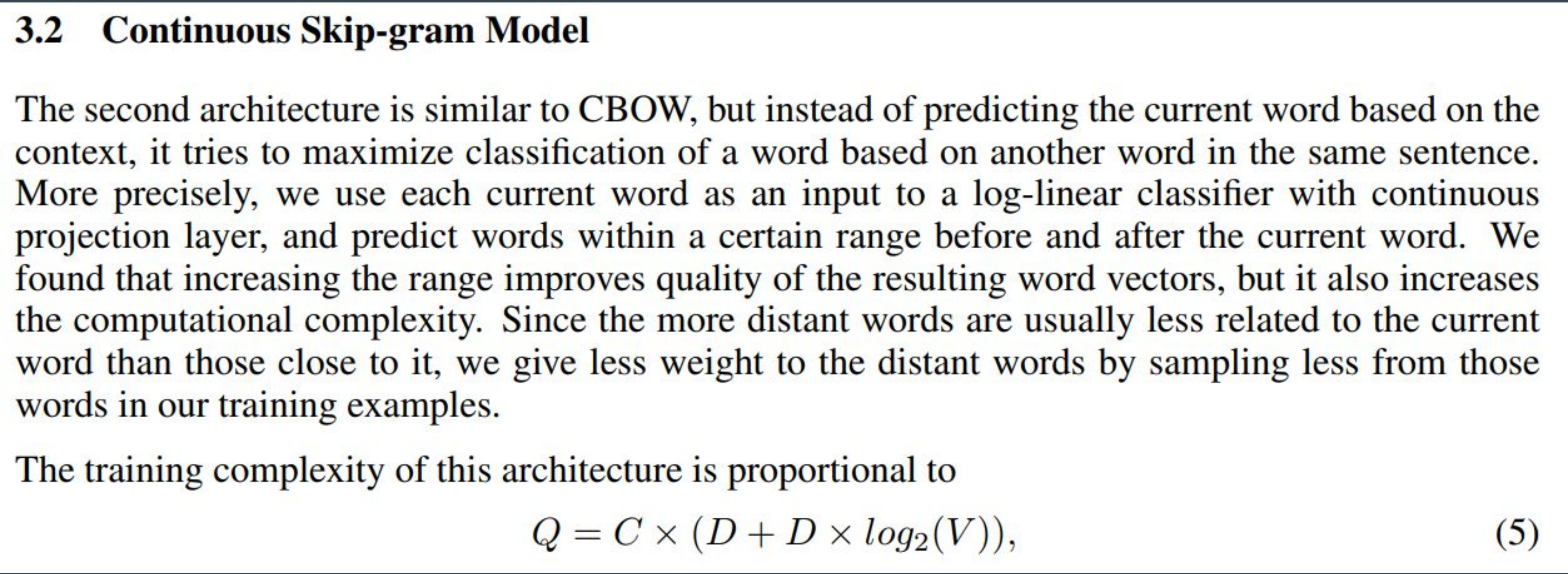
3.1 Continuous Bag-of-Words Model

The first proposed architecture is similar to the feedforward NNLM, where the non-linear hidden layer is removed and the projection layer is shared for all words (not just the projection matrix); thus, all words get projected into the same position (their vectors are averaged). We call this architecture a bag-of-words model as the order of words in the history does not influence the projection. Furthermore, we also use words from the future; we have obtained the best performance on the task introduced in the next section by building a log-linear classifier with four future and four history words at the input, where the training criterion is to correctly classify the current (middle) word. Training complexity is then

$$Q = N \times D + D \times log_2(V). \tag{4}$$

We denote this model further as CBOW, as unlike standard bag-of-words model, it uses continuous distributed representation of the context. The model architecture is shown at Figure 1. Note that the weight matrix between the input and the projection layer is shared for all word positions in the same way as in the NNLM.

Skip-gram [4, 5]



3.2 Continuous Skip-gram Model

The second architecture is similar to CBOW, but instead of predicting the current word based on the context, it tries to maximize classification of a word based on another word in the same sentence. More precisely, we use each current word as an input to a log-linear classifier with continuous projection layer, and predict words within a certain range before and after the current word. We found that increasing the range improves quality of the resulting word vectors, but it also increases the computational complexity. Since the more distant words are usually less related to the current word than those close to it, we give less weight to the distant words by sampling less from those words in our training examples.

The training complexity of this architecture is proportional to

$$Q = C \times (D + D \times log_2(V)), \tag{5}$$

Results [4, 5]

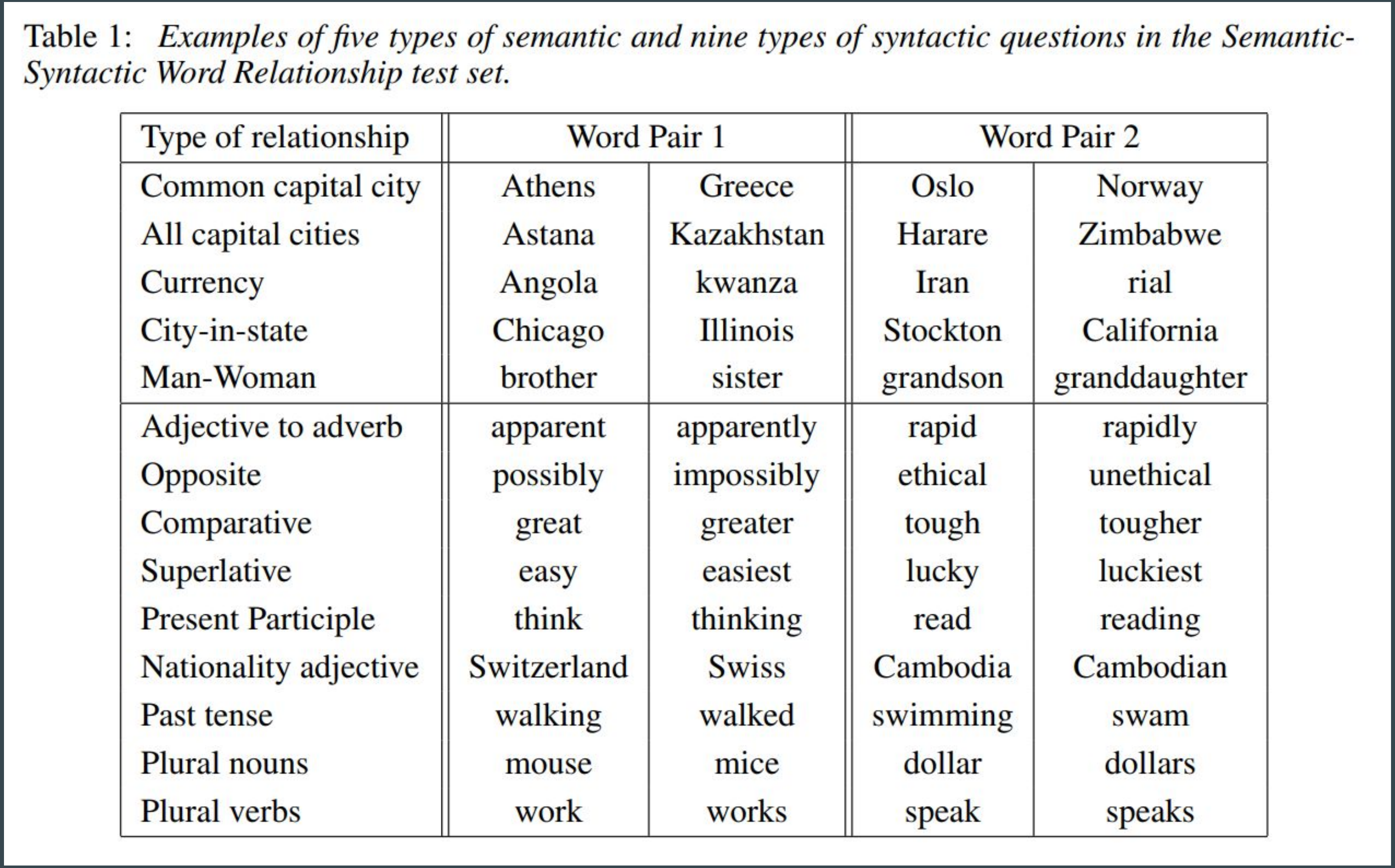


Table 1: *Examples of five types of semantic and nine types of syntactic questions in the Semantic-Syntactic Word Relationship test set.*

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Results(2) [4, 5]

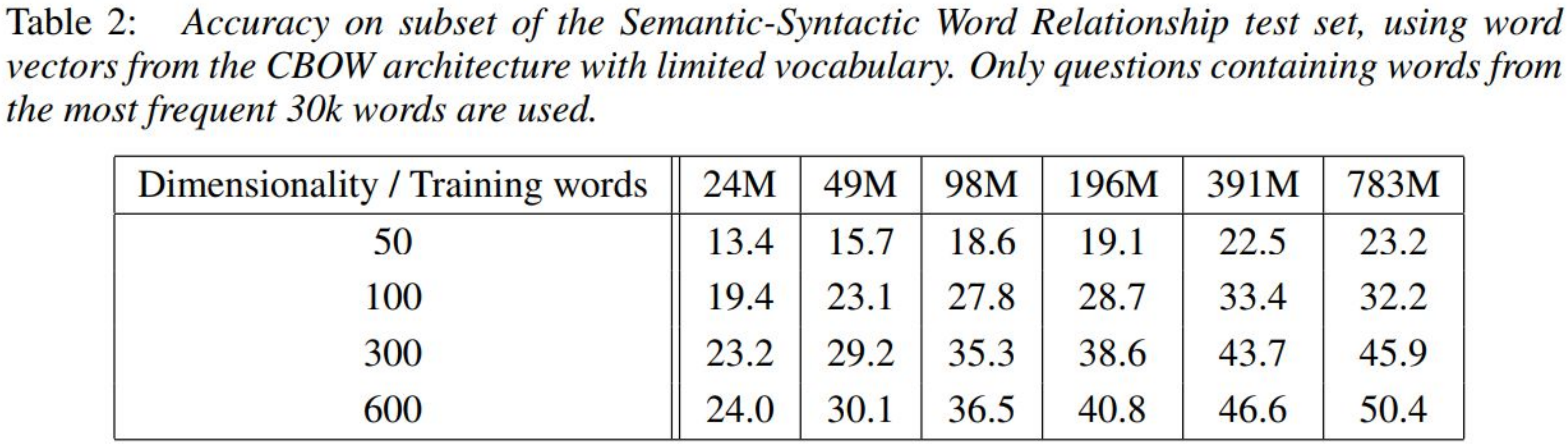


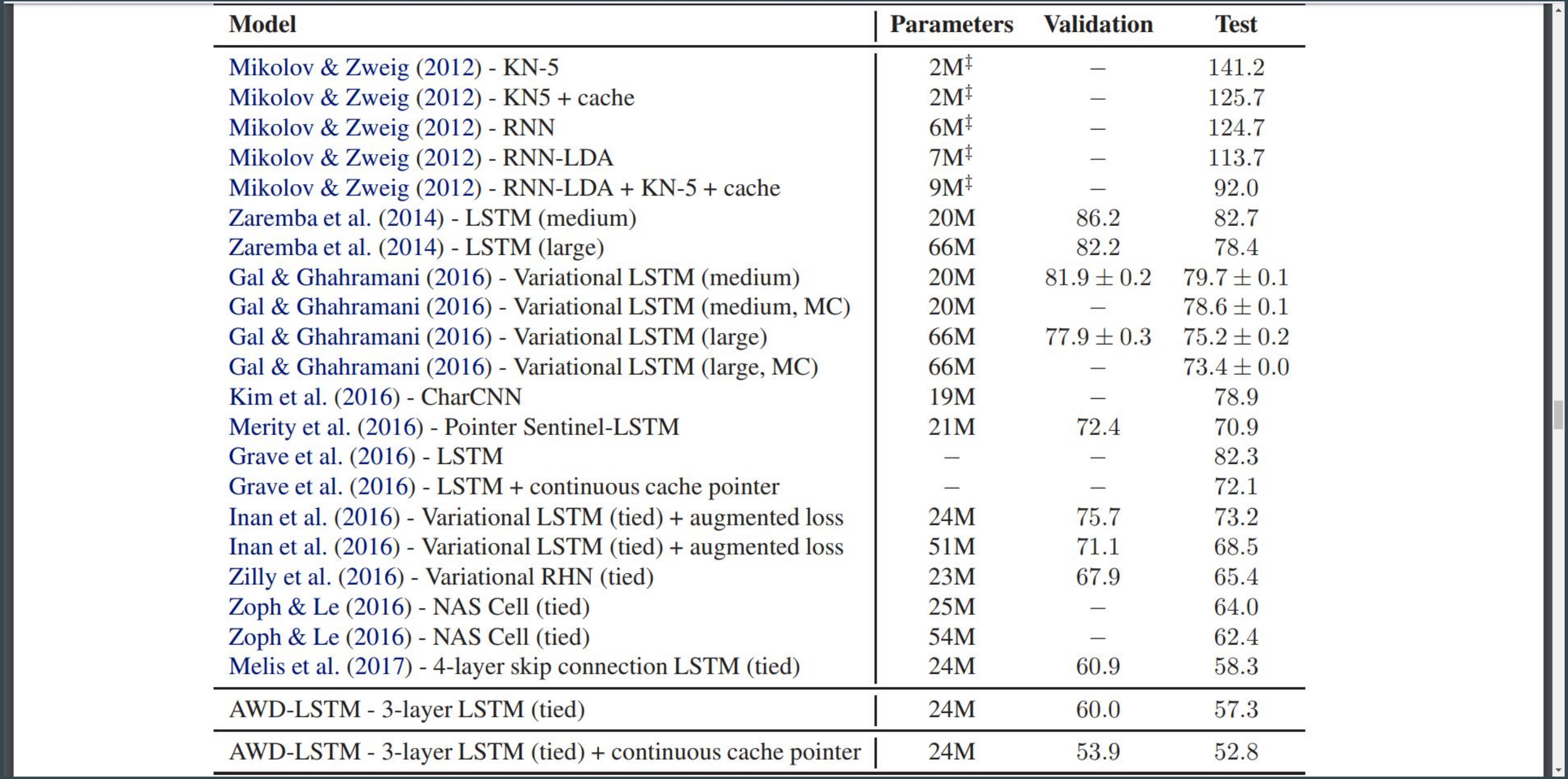
Table 2: *Accuracy on subset of the Semantic-Syntactic Word Relationship test set, using word vectors from the CBOW architecture with limited vocabulary. Only questions containing words from the most frequent 30k words are used.*

Dimensionality / Training words	24M	49M	98M	196M	391M	783M
50	13.4	15.7	18.6	19.1	22.5	23.2
100	19.4	23.1	27.8	28.7	33.4	32.2
300	23.2	29.2	35.3	38.6	43.7	45.9
600	24.0	30.1	36.5	40.8	46.6	50.4

LSTM-AWD [6]

- Weight Drop instead of drop out (compatible with cuDNN)
- SGD is better than momentum based methods in LM => Averaged SGD => NT-ASGD (trigger based on perplexity)
- Variable-length BPTT => make sure every word uses the full BPTT + needs LR rescaling
- Variational Dropout (for both input and output)
- Weight tying
- LSTM hidden size != word embedding size
- Activation Regularization + Temporal AR

Results [6]



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

References

1. On the computational power of neural nets
2. LONG SHORT-TERM MEMORY
3. LSTM: A Search Space Odyssey
4. Efficient Estimation of Word Representations in Vector Space
5. Distributed Representations of Words and Phrases and their Compositionality
6. Regularizing and Optimizing LSTM Language Models