CMC MSU Department of Algorithmic Languages
Samsung Moscow Research Center

Recent approaches to natural language processing with neural networks

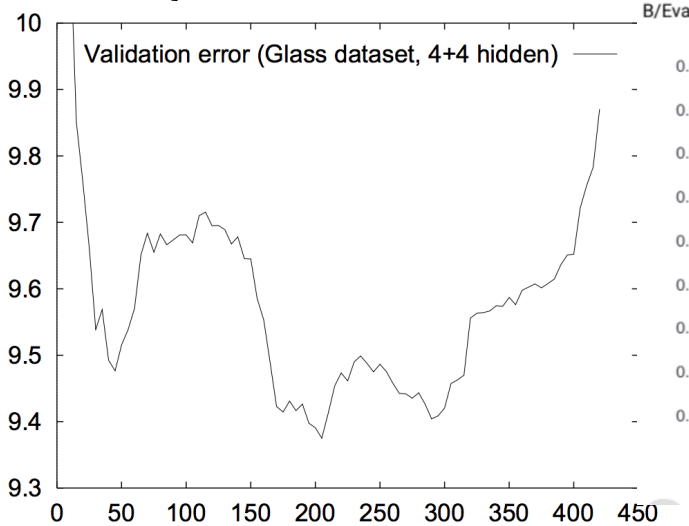
Современные подходы к обработке текстов с помощью нейронных сетей

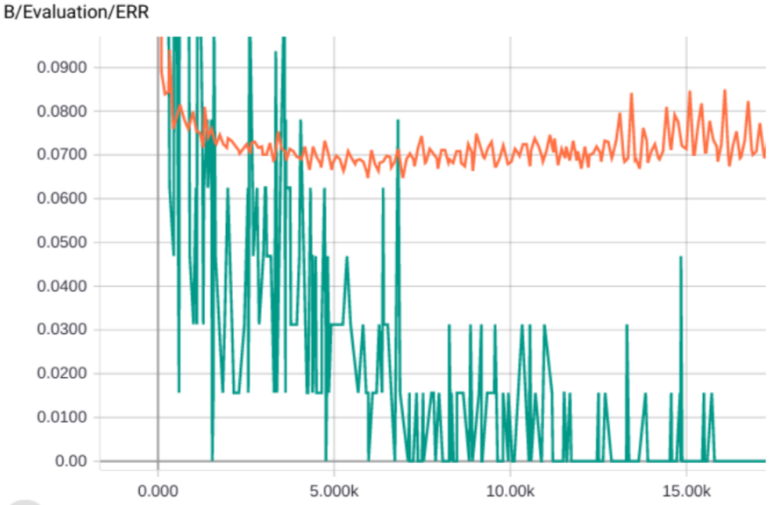
Block: Recurrent Neural Networks.
Lecture 2. Recent Approaches to Language Modeling with LSTMs.

Arefyev Nikolay
CMC MSU Department of Algorithmic Languages &
Samsung Moscow Research Center

Regularization: early stopping

1. Early stopping
   - Stop training when **accuracy** (target metric) **on validation set** starts decreasing
     - Don’t stop too early! Exact criterion is an open question.
   - Don’t use early stopping for stopping. Stop training when you have surely overfitted. But use weights giving best **validation accuracy.**
   - Very efficient: doesn’t require multiple runs to select hyperparams
   - Stability/reproducibility problems, don’t use as the only regularization technique!
   - Some experts (e.g. Andrew Ng) recommend avoiding early stopping
     - Decouple optimization and regularization subtasks [https://www.coursera.org/lecture/deep-neural-network/other-regularization-methods-Pa53F]





Figures from Lutz Prechel. Early Stopping | but when? 2012

Regularization: L2-reg / weight decay

L2-regularization:

penalize l2-norm of weights vector (excluding bias)

```
final_loss = loss + lambda * (w**2).sum() / 2
```

Weight decay

contract weights (excluding bias) towards 0 before update

```
w = w * (1 - wd)
```

- For SGD they are the same!

```
w = w - lr * (loss' + lambda * w) # wd = lr * lambda
```

- For other optimizers they are different!

Image classification (CIFAR-10)
(error, lower – better)

Method	Without amsgrad
AdamW	5.66% **WD**
Adam	6.06% L2-reg

AWD LSTM on WikiText-2 (perplexity: lower - better)

Method	Raw model	With cache pointer
Adam L2-reg	68.7/65.5	52.9/50.9
Adam + amsgrad	69.4/66.5	53.1/51.3
AdamW WD	68.9/65.7	52.8/50.9
AdamW + amsgrad	72.7/69	57/54.7

Results from Sylvain Gugger and Jeremy Howard. AdamW and Super-convergence is now the fastest way to train neural nets, 2018. https://www.fast.ai/2018/07/02/adam-weight-decay/

Regularization: Max norm constraint

- For each neuron (individually) limits it's weights (excluding bias) to have maximum L2-norm of *c*
  - clips column-wise norms of weight matrices to *c*=3-4
  - doesn't force weights to be near 0
  - [Srivastava et al, 2014] combines max-norm with dropout:

"Although dropout alone gives significant improvements, using dropout along with maxnorm regularization, large decaying learning rates and high momentum provides a significant boost over just using dropout. A possible justification is that constraining weight vectors to lie inside a ball of fixed radius makes it possible to use a huge learning rate without the possibility of weights blowing up. The noise provided by dropout then allows the optimization process to explore different regions of the weight space that would have otherwise been difficult to reach. As the learning rate decays, the optimization takes shorter steps, there by doing less exploration and eventually settles into a minimum."

Method	**Test Classification error %**
L2	1.62
L2 + L1 applied towards the end of training	1.60
L2 + KL-sparsity	1.55
Max-norm	1.35
Dropout + L2	1.25
Dropout + Max-norm	**1.05**

Table 9: Comparison of different regularization methods on MNIST.

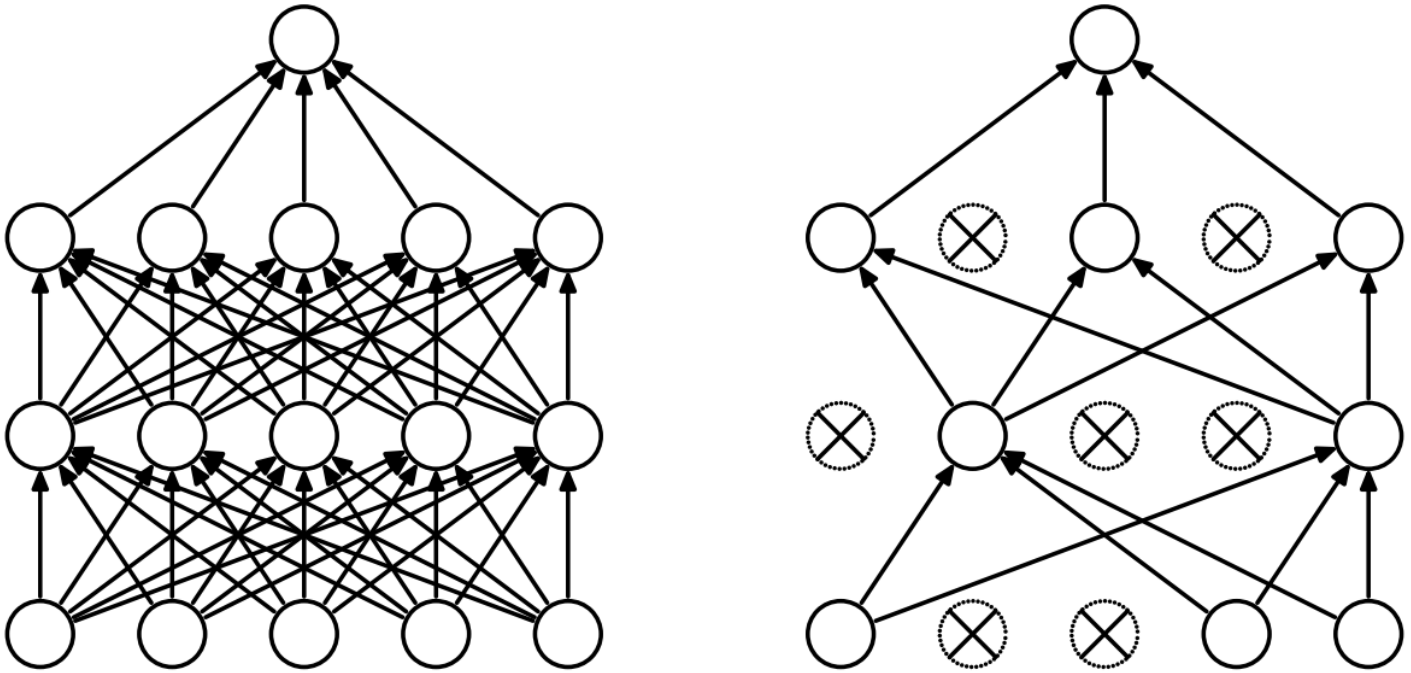
Figure from Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting, 2014

Regularization: Dropout (train)

- The most popular regularizer for Neural Nets!

  Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting, 2014

  - When training: set each neuron's output to 0 with probability 1-p
    - Hyperparameter p (keep probability) should be selected (0.5 is a common default)



(a) Standard Neural Net (b) After applying dropout.

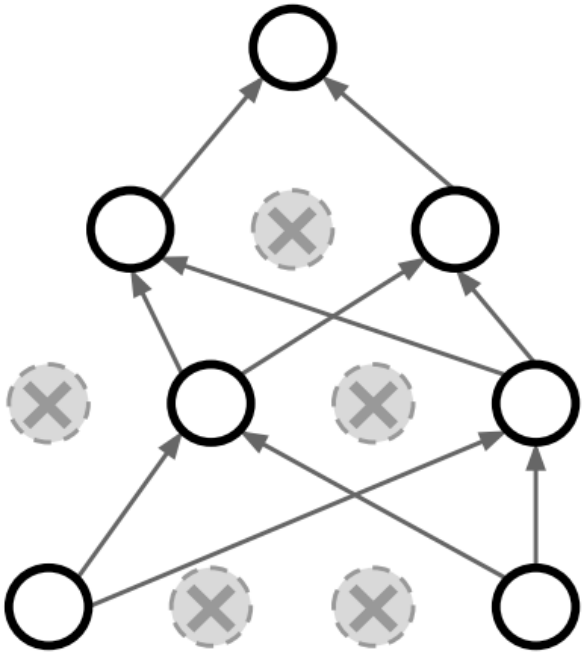
Figure from Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting, 2014

Regularization: Dropout (train)

```python
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

Figure from Fei-Fei Li & Justin Johnson & Serena Yeung. Cs231n slides, 2018

Regularization: Dropout



Forces the network to have a redundant representation;
Prevents co-adaptation of features

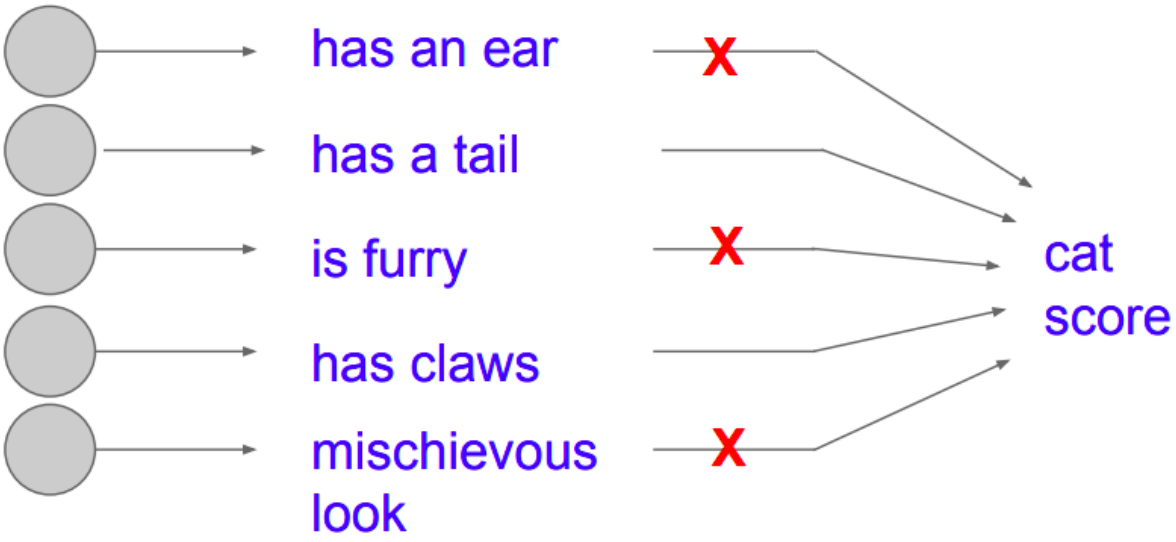
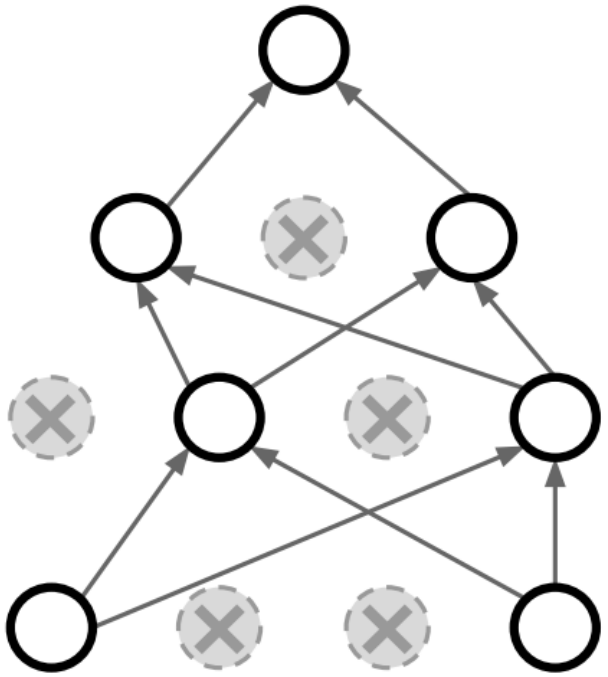


Figure from Fei-Fei Li & Justin Johnson & Serena Yeung. Cs231n slides, 2018

Regularization: Dropout



Another interpretation:

Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

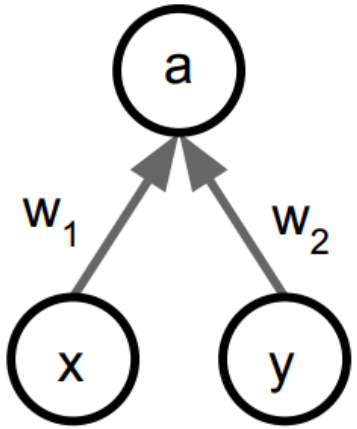
An FC layer with 4096 units has
$2^{4096} \sim 10^{1233}$ possible masks!
Only $\sim 10^{82}$ atoms in the universe...

Figure from Fei-Fei Li & Justin Johnson & Serena Yeung. Cs231n slides, 2018

Regularization: Dropout (test)

Want to approximate the integral

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$



Consider a single neuron.

At test time we have: $E[a] = w_1 x + w_2 y$

During training we have:

$$\begin{aligned} E[a] =& \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ =& \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

At test time, **multiply** by dropout probability

Figure from Fei-Fei Li & Justin Johnson & Serena Yeung. Cs231n slides, 2018

Regularization: Dropout (test)

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

At test time all neurons are active always
=> We must scale the activations so that for each neuron:
output at test time = expected output at training time

Figure from Fei-Fei Li & Justin Johnson & Serena Yeung. Cs231n slides, 2018

Regularization: Inverted Dropout

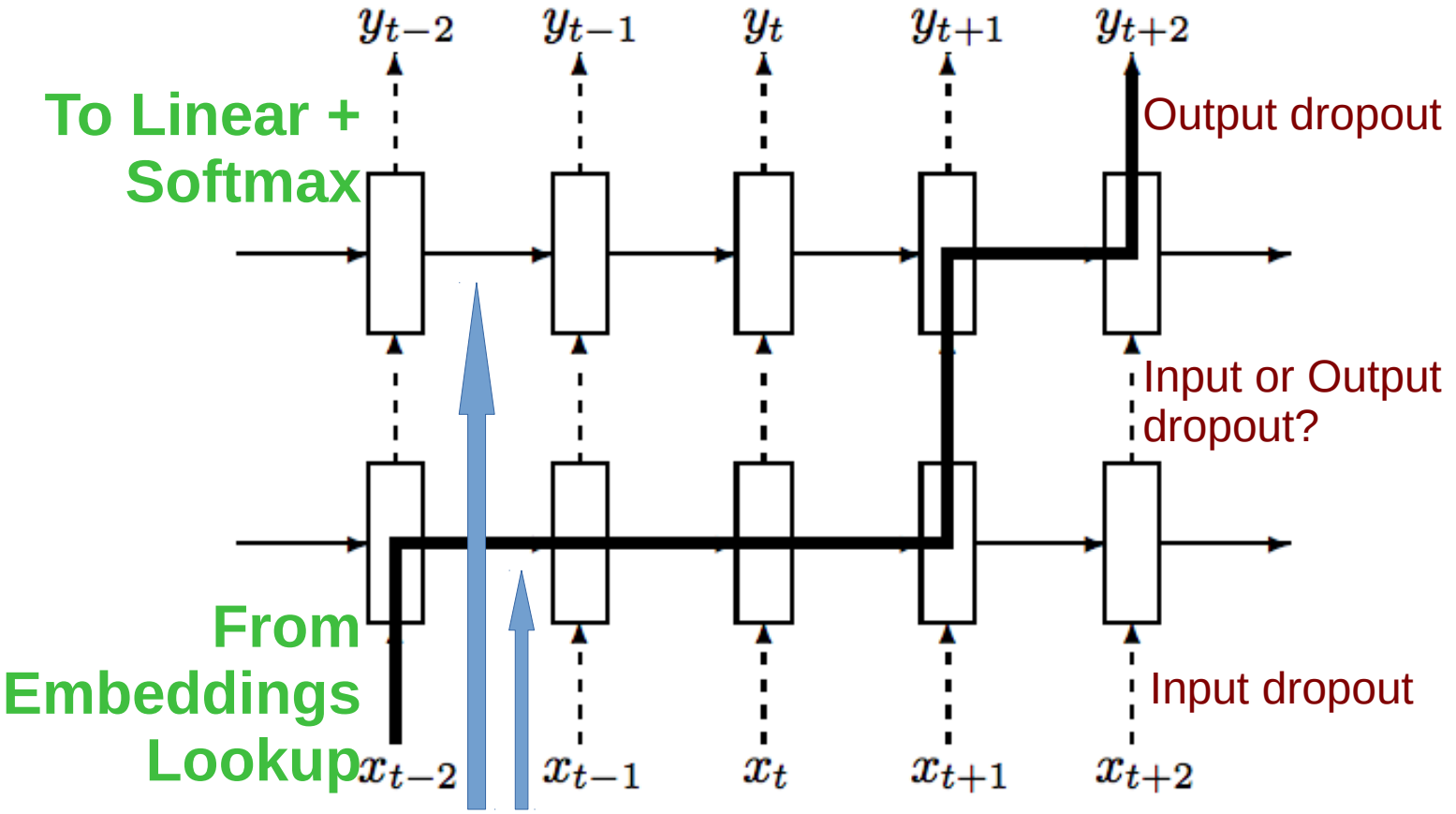
```python
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

test

Can forget keep probability used for training or even change it during training!

Figure from Fei-Fei Li & Justin Johnson & Serena Yeung. Cs231n slides, 2018

Zaremba: input/output dropout



No dropout on hidden states (recurrent dropout)!
Otherwise information stored in a hidden unit k steps ago is lost with prob p^k => long-range dependencies (large k) couldn't be learnt

Figure from Zaremba et al. Recurrent Neural Network Regularization, 2014

Zaremba hyperparameters

	small	medium	large
lstm_nlayers	2	2	2
lstm_size=embs_size	200	650	1500
dropout_keep_prob	-	0.5	0.35
init	U[-0.1, 0.1]	U[-0.05, 0.05]	U[-0.04, 0.04]
start_lr	1	1	1
const_lr_nepochs	4	6	14
div_lr/epoch	2	1.2	1.15
nepochs	13	39	55
grad_clipping	5	5	10
bs	20	20	20
tbtt_steps	20	35	35
time (PTB, 1GPU'14)	2-3h	12h	24h

Optimal size, Larger overfits!

Zaremba PTB word level perplexity

Model	Validation set	Test set
A single model		
Pascanu et al. (2013)		107.5
Cheng et al.		100.0
non-regularized LSTM	120.7	114.5
Medium regularized LSTM	86.2	82.7
Large regularized LSTM	82.2	**78.4**
Model averaging		
Mikolov (2012)		83.5
Cheng et al.		80.6
2 non-regularized LSTMs	100.4	96.1
5 non-regularized LSTMs	87.9	84.1
10 non-regularized LSTMs	83.5	80.0
2 medium regularized LSTMs	80.6	77.0
5 medium regularized LSTMs	76.7	73.3
10 medium regularized LSTMs	75.2	72.0
2 large regularized LSTMs	76.9	73.6
10 large regularized LSTMs	72.8	69.5
38 large regularized LSTMs	71.9	**68.7**
Model averaging with dynamic RNNs and n-gram models		
Mikolov & Zweig (2012)		72.9

Figure from Zaremba et al. Recurrent Neural Network Regularization, 2014

Zaremba Samples

---

the meaning of life is that only if an end would be of the whole supplier. widespread rules are regarded as the companies of refuses to deliver. in balance of the nation 's information and loan growth associated with the carrier thrifts are in the process of slowing the seed and commercial paper.

---

the meaning of life is nearly in the first several months before the government was addressing such a move as president and chief executive of the nation past from a national commitment to curb grounds. meanwhile the government invests overcapacity that criticism and in the outer reversal of small-town america.

Figure 4: Some interesting samples drawn from a large regularized model conditioned on "The meaning of life is". We have removed "unk", "N", "$" from the set of permissible words.

Which sampling approach was used? (ancestral sampling vs. beam search)

Does PTB (Wall Street Journal) says something about the meaning of life?

Figure from Zaremba et al. Recurrent Neural Network Regularization, 2014

Zaremba Other Tasks

Model	Training set	Validation set
Non-regularized LSTM	71.6	68.9
Regularized LSTM	69.4	**70.5**

Table 2: Frame-level accuracy on the Icelandic Speech Dataset. The training set has 93k utterances.

Model	Test perplexity	Test BLEU score
Non-regularized LSTM	5.8	25.9
Regularized LSTM	5.0	29.03
LIUM system		33.30

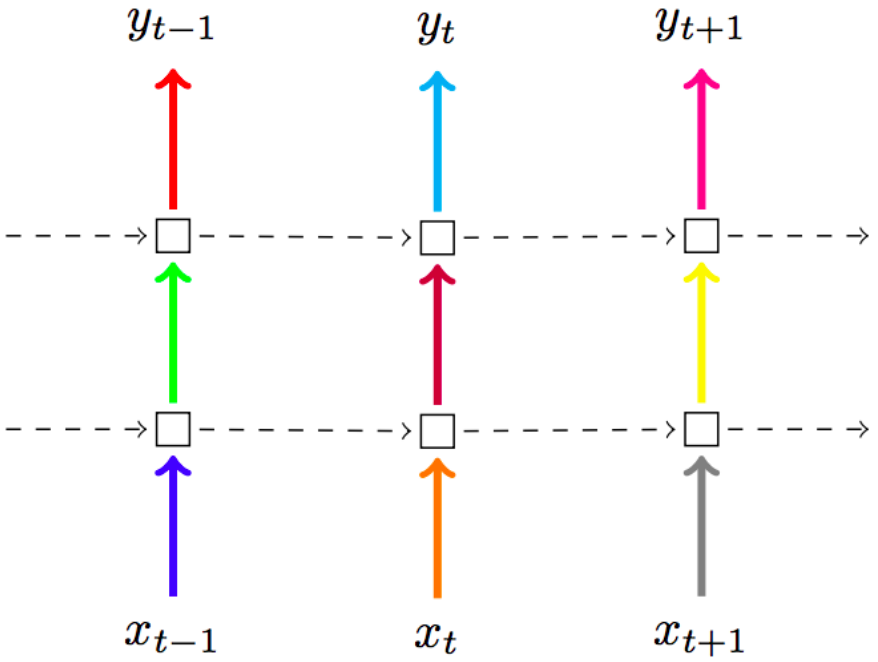
Table 3: Results on the English to French translation task.

Model	Test perplexity	Test BLEU score
Non-regularized model	8.47	23.5
Regularized model	7.99	24.3
10 non-regularized models	7.5	24.4

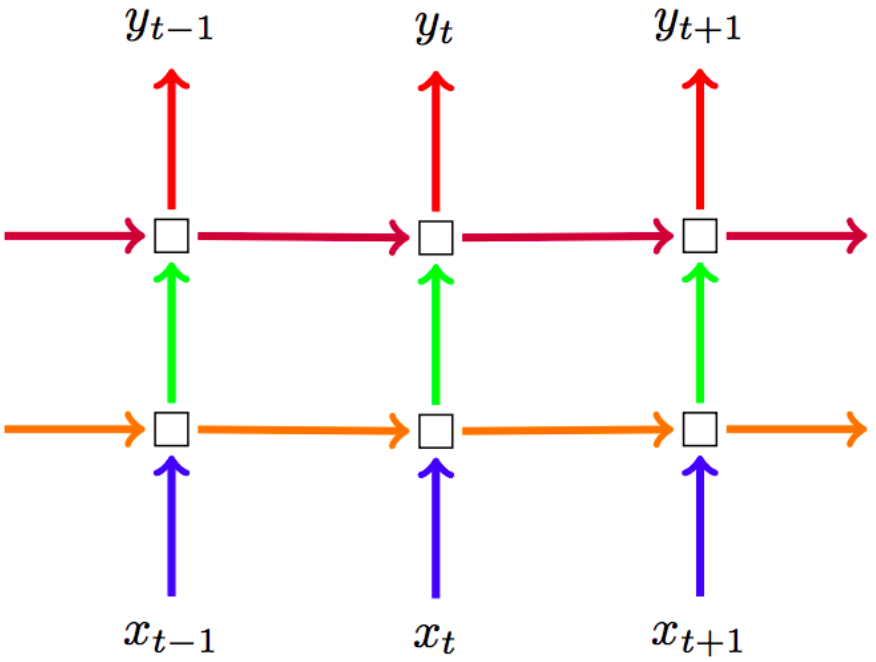
Table 4: Results on the image caption generation task.

Figure from Zaremba et al. Recurrent Neural Network Regularization, 2014

Variational RNN / Recurrent Dropout



(a) Naive dropout RNN



(b) Variational RNN

$$\begin{pmatrix} \underline{\mathbf{i}} \\ \underline{\mathbf{f}} \\ \underline{\mathbf{o}} \\ \underline{\mathbf{g}} \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} \left(\begin{pmatrix} \mathbf{x}_t \circ \mathbf{z}_{\mathbf{x}}^t \\ \mathbf{h}_{t-1} \end{pmatrix} \cdot \mathbf{W} \right)$$

$$\begin{pmatrix} \underline{\mathbf{i}} \\ \underline{\mathbf{f}} \\ \underline{\mathbf{o}} \\ \underline{\mathbf{g}} \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} \left(\begin{pmatrix} \mathbf{x}_t \circ \mathbf{z}_{\mathbf{x}} \\ \mathbf{h}_{t-1} \circ \mathbf{z}_{\mathbf{h}} \end{pmatrix} \cdot \mathbf{W} \right)$$

Zaremba:
new dropout mask (color) on each timestep
no recurrent dropout (dashed)

Gal&Ghahramani:
same dropout mask (color) on each timestep
makes possible recurrent dropout

Figures from Gal&Ghahramani. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks, 2016

Tied VS Untied Weights

Dropping features in x / h is equivalent to dropping whole rows of U / W.
Untied weights: dropping different rows x / h for different gates:

$$\underline{\mathbf{i}} = \mathrm{sigm}\left(\mathbf{h}_{t-1}\mathbf{U}_i + \mathbf{x}_t\mathbf{W}_i\right) \qquad \underline{\mathbf{f}} = \mathrm{sigm}\left(\mathbf{h}_{t-1}\mathbf{U}_f + \mathbf{x}_t\mathbf{W}_f\right)$$

$$\underline{\mathbf{o}} = \mathrm{sigm}\left(\mathbf{h}_{t-1}\mathbf{U}_o + \mathbf{x}_t\mathbf{W}_o\right) \qquad \underline{\mathbf{g}} = \tanh\left(\mathbf{h}_{t-1}\mathbf{U}_g + \mathbf{x}_t\mathbf{W}_g\right)$$

$$\mathbf{c}_t = \underline{\mathbf{f}} \circ \mathbf{c}_{t-1} + \underline{\mathbf{i}} \circ \underline{\mathbf{g}} \qquad \mathbf{h}_t = \underline{\mathbf{o}} \circ \tanh(\mathbf{c}_t)$$

Tied weights: dropping same features in x / h for different gates.

$$\begin{pmatrix} \underline{\mathbf{i}} \\ \underline{\mathbf{f}} \\ \underline{\mathbf{o}} \\ \underline{\mathbf{g}} \end{pmatrix} = \begin{pmatrix} \mathrm{sigm} \\ \mathrm{sigm} \\ \mathrm{sigm} \\ \tanh \end{pmatrix} \left(\begin{pmatrix} \mathbf{x}_t \\ \mathbf{h}_{t-1} \end{pmatrix} \cdot \mathbf{W} \right)$$

Figures from Gal&Ghahramani. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks, 2016

Variational LSTM

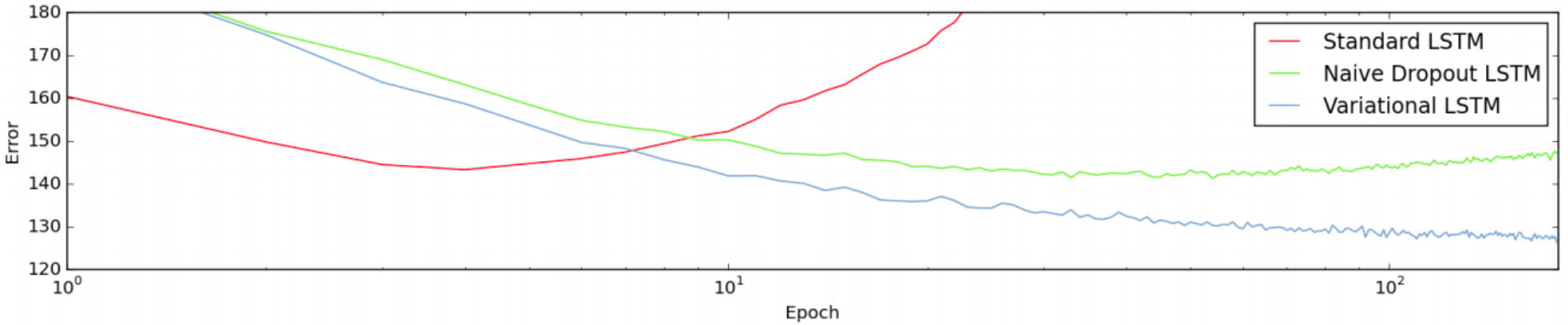


Figure 2: Medium model validation perplexity for the Penn Treebank language modelling task. Learning rate decay was reduced to assess model overfitting using dropout alone. Even with early stopping, Variational LSTM achieves lower perplexity than naive dropout LSTM and standard LSTM. Lower perplexity for all models can be achieved with learning rate decay scheduling, seen in table 1.

Figure from Gal&Ghahramani. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks, 2016

Variational RNN + Embedding Dropout

Large model + Early stopping
= Small model

	Medium LSTM			Large LSTM		
	Validation	Test	WPS	Validation	Test	WPS
Non-regularized (early stopping)	121.1	121.7	5.5K	128.3	127.4	2.5K
Moon et al. [20]	100.7	97.0	4.8K	122.9	118.7	3K
Moon et al. [20] +emb dropout	88.9	86.5	4.8K	88.8	86.0	3K
Zaremba et al. [4]	86.2	82.7	5.5K	82.2	78.4	2.5K
Variational (tied weights)	81.8 ± 0.2	79.7 ± 0.1	4.7K	77.3 ± 0.2	75.0 ± 0.1	2.4K
Variational (tied weights, MC)	–	79.0 ± 0.1	–	–	74.1 ± 0.0	–
Variational (untied weights)	81.9 ± 0.2	79.7 ± 0.1	2.7K	77.9 ± 0.3	75.2 ± 0.2	1.6K
Variational (untied weights, MC)	–	$\mathbf{78.6 \pm 0.1}$	–	–	$\mathbf{73.4 \pm 0.0}$	–

Table 1: Single model perplexity (on test and validation sets) for the Penn Treebank language modelling task. Two model sizes are compared (a medium and a large LSTM, following [4]'s setup), with number of processed words per second (WPS) reported. Both dropout approximation and MC dropout are given for the test set with the Variational model. A common approach for regularisation is to reduce model complexity (necessary with the non-regularised LSTM). With the Variational models however, a significant reduction in perplexity is achieved by using larger models.

MC (Monte Carlo) dropout is slow (note absence of WPS!)

Figure from Gal&Ghahramani. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks, 2016

Embedding Dropout

- Set to zeros whole word embeddings for randomly selected words
- Equivalent to dropout on 1-hot inputs with shared mask
- Equivalent to hiding (using 0 embeddings) from input randomly selected word types

  the wind and the wolf → _ *wind and* _ *wolf*
- Encourage model not to depend on specific words at input
- Reduce overfitting due to embeddings matrix (most parameters in LM)

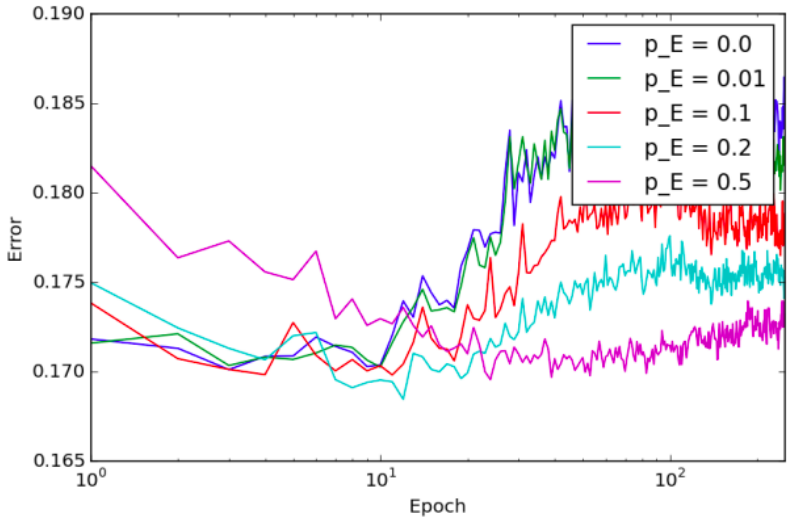


Figure 8: Test error for various embedding dropout probabilities, with sequence length 50.

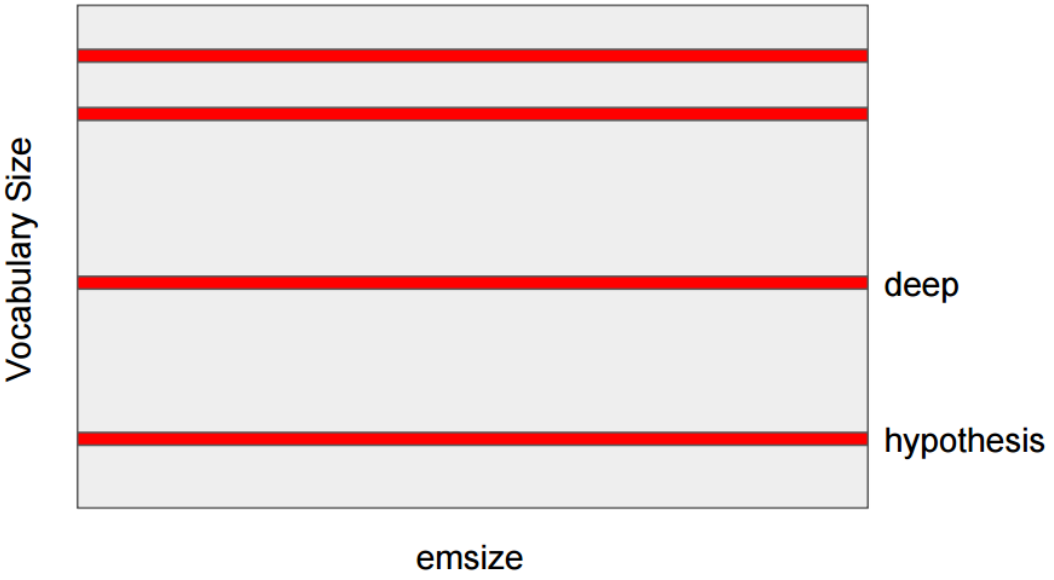


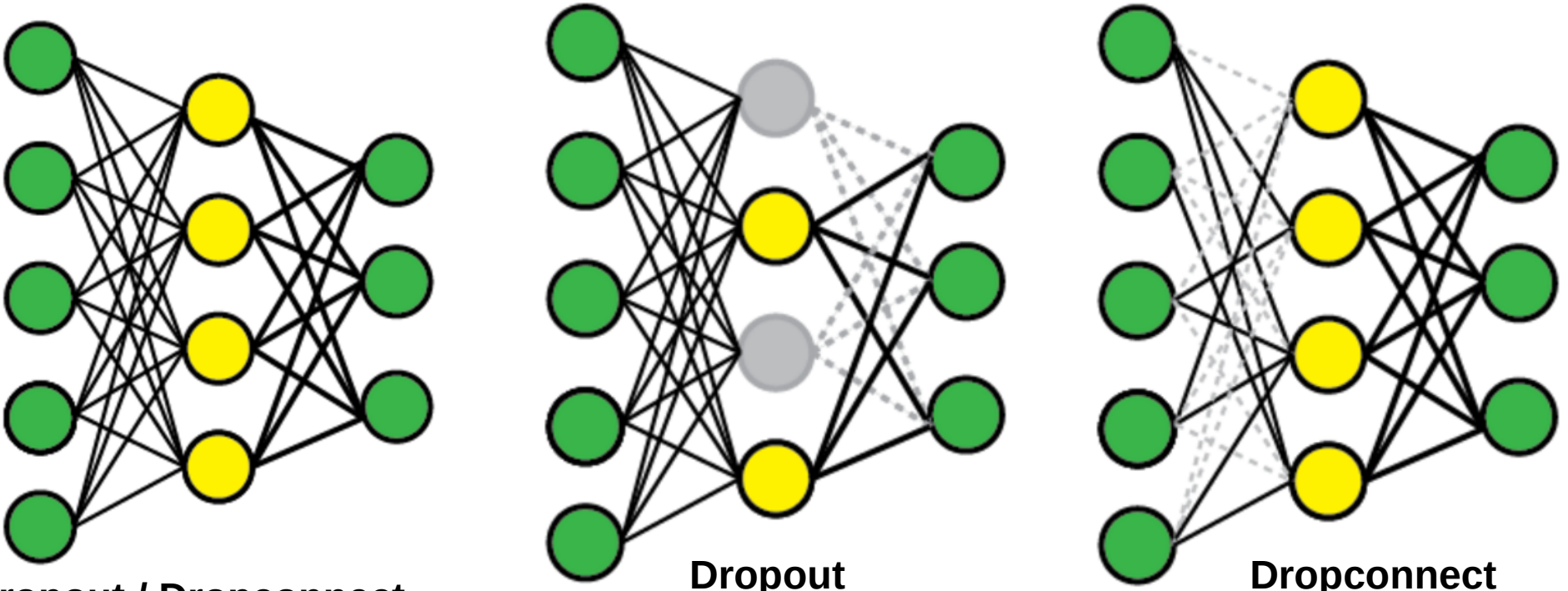
Figure from Keskar, Merity. State-of-the-Art large scale language modeling in 12 hours with a single GPU AND Gal&Ghahramani. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks, 2016

Dropconnect vs. Dropout

Sets to 0 randomly selected weights (vs. neuron outputs)

Equivalent to dropout on weight matrices

[Wan et al. Regularization of Neural Networks using DropConnect, 2013]



No Dropout / Dropconnect **Dropout** **Dropconnect**

Figure from https://stats.stackexchange.com/questions/201569/difference-between-dropout-and-dropconnect

Weight Dropped RNN

- Dropconnect applied to recurrent connections (instead of recurrent dropout)

  AWD-LSTM [Merity et al. Regularizing and Optimizing LSTM Language Models, **2018**] ←**find 2018 version on openreview.net, 2017 version contains lots of errors!**

- Applied once for each batch for U-matrices

  => same weights are dropped for all timesteps (like Variational RNN)

  => compatible with existing highly optimized LSTM implementations like NVIDIA's cuDNN LSTM (unlike Variational RNN)

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

Figure from Merity et al. Regularizing and Optimizing LSTM Language Models, 2018

Tied input/output embeddings

- a.k.a. tied softmax, shared softmax
- Same weight matrix in embedding lookup (input embeddings) and final linear+softmax layer (output embeddings)
  - **WT** (weight tying): Press et al. **2017**: Using the output embedding to improve language models
  - **RE** (reused embeddings): Inan et al. **2017**: Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling

=> ~2x reduction in number of parameters (for large vocabulary)

=> reduces overfitting, better generalization (results on test)

Input/output embeddings: word similarity

Word2vec:

- Input and output ems perform similar
- Tied embs perform worse
  - Self cooccurence, [Goldberg&Levy, 14]

	Input	Output	Tied
Simlex999	0.30	0.29	0.17
Verb-143	0.41	0.34	0.12
MEN	0.66	0.61	0.50
Rare-Word	0.34	0.34	0.23
MTurk-771	0.59	0.54	0.37

Table 2: Comparison of input and output embeddings learned by a word2vec skip-gram model. Results are also shown for the tied word2vec model. Spearman's correlation ρ is reported for five word embedding evaluation benchmarks.

A	B	$\rho(A,B)$ word2vec	$\rho(A,B)$ NNLM(S)	$\rho(A,B)$ NNLM(L)
In	Out	0.77	0.13	0.16
In	Tied	0.19	0.31	0.45
Out	Tied	0.39	0.65	0.77

Table 4: Spearman's rank correlation ρ of similarity values between all pairs of words evaluated for the different embeddings: input/output embeddings (of the untied model) and the embeddings of our tied model. We show the results for both the word2vec models and the small and large NNLM models from (Zaremba et al., 2014).

	PTB			text8		
Embedding	In	Out	Tied	In	Out	Tied
Simlex999	0.02	0.13	0.14	0.17	0.27	0.28
Verb143	0.12	0.37	0.32	0.20	0.35	0.42
MEN	0.11	0.21	0.26	0.26	0.50	0.50
Rare-Word	0.28	0.38	0.36	0.14	0.15	0.17
MTurk771	0.17	0.28	0.30	0.26	0.48	0.45

Table 3: Comparison of the input/output embeddings of the small model from (Zaremba et al., 2014) and the embeddings from our weight tied variant. Spearman's correlation ρ is presented.

Zaremba small:

- Output embs perform better, similar to tied embs
  - Sparse updates of input embs
  - and long path for gradients
- Tied embs are much more similar to output embs than to input embs

Figures from Press et al. Using the output embedding to improve language models, 17

Tied input/output embeddings

Model	Size	Train	Val.	Test
Large (Zaremba et al., 2014)	66M	37.8	82.2	78.4
Large + Weight Tying	51M	48.5	77.7	74.3
Large + BD (Gal, 2015) + WD	66M	24.3	78.1	75.2
Large + BD + WT	51M	28.2	75.8	73.2
RHN (Zilly et al., 2016) + BD	32M	67.4	71.2	68.5
RHN + BD + WT	24M	74.1	68.1	66.0

Perplexity on PTB. Figure from Press et al. Using the output embedding to improve language models, 17

		PTB		Wikitext-2	
Network	**Model**	**Valid**	**Test**	**Valid**	**Test**
Small[4] (200 units)	VD-LSTM	92.6	87.3	112.2	105.9
	VD-LSTM+AL	86.3	82.9	110.3	103.8
	VD-LSTM+RE	89.9	85.1	106.1	100.5
	VD-LSTM+REAL	86.3	82.7	105.6	98.9
Medium (650 units)	VD-LSTM	82.0	77.7	100.2	95.3
	VD-LSTM+AL	77.4	74.7	98.8	93.1
	VD-LSTM+RE	77.1	73.9	92.3	87.7
	VD-LSTM+REAL	75.7	73.2	91.5	87.0
Large[5] (1500 units)	VD-LSTM	76.8	72.6	-	-
	VD-LSTM+AL	74.5	71.2	-	-
	VD-LSTM+RE	72.5	69.0	-	-
	VD-LSTM+REAL	71.1	68.5	-	-

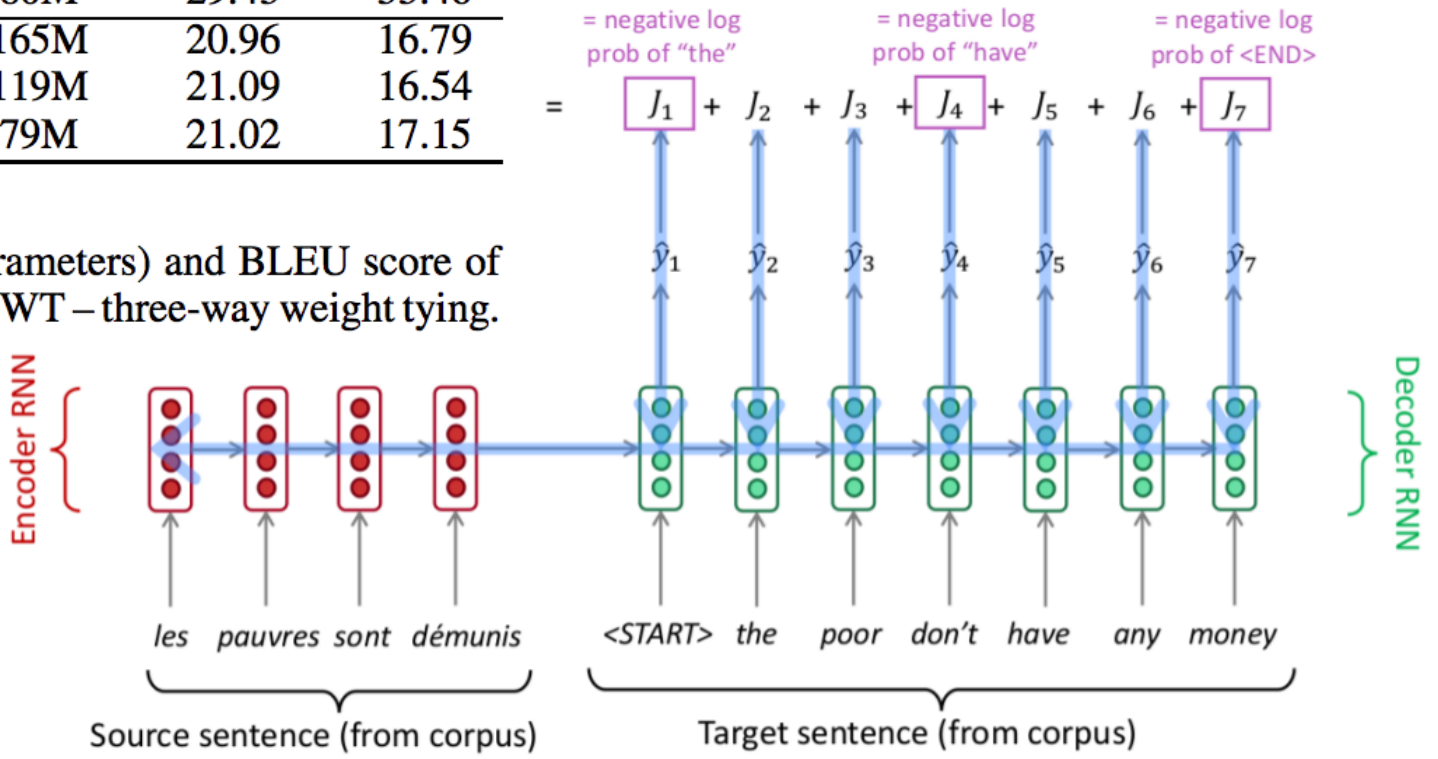
Fig. from Inan et al. Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling, 17

Tied embeddings: MT

- Weight Tying in Decoder helps also in MT
- Can even tie enc in, dec in, dec out embs (TWWT)
  - 85-90% of BPE subwords are common for En, Fr, Ge
  - Will it work for more distant languages?

		Size	Validation	Test
EN→FR	Baseline	168M	29.49	33.13
	Decoder WT	122M	29.47	33.26
	TWWT	80M	29.43	33.46
EN→DE	Baseline	165M	20.96	16.79
	Decoder WT	119M	21.09	16.54
	TWWT	79M	21.02	17.15

Table 8: Size (number of parameters) and BLEU score of various translation models. TWWT – three-way weight tying.



AR / TAR

- L2-regularization on RNN outputs / difference of adjacent RNN outputs
- Add to the loss:
  - activation regularization $\alpha L_2(m \odot h_t)$
  - temporal activation regularization $\beta L_2(h_t - h_{t+1})$

- Introduced in Revisiting activation regularization for language rnns, 2017

NT-AvSGD

- AvSGD = Average SGD
  - Instead of weights from last step of SGD take average over several last steps
  - Allows using constant lr during training!
    - Tune only lr, no need to tune lr decay!
  - Polyak and Juditsky. Acceleration of stochastic approximation by averaging, **1992**
  - Mandt, Hoffman, Blei. Stochastic gradient descent as approximate bayesian inference, 2017
- NT = Non-monotonically triggered
  - But when start averaging? When SGD starts oscillating around minimum / val perplexity doesn’t improve
  - Conservative criterion: current val perplexity is larger (worse) than minimal val perplexity over training **excluding last few steps**

NT-AvSGD

- Not much hypers tuning: L=1 (logging once an epoch), n=5 (wait at least 5 epochs before averaging)

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

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

Initialize $k \leftarrow 0$, $t \leftarrow 0$, $T \leftarrow 0$, `logs` $\leftarrow$ `[]`
while stopping criterion not met **do**
  Compute stochastic gradient $\hat{\nabla} f(w_k)$ and take SGD step (1).
  if $\texttt{mod}(k, L) = 0$ and $T = 0$ **then**
    Compute validation perplexity v.
    if $t > n$ and $v > \min_{l \in \{0, \cdots, t-n-1\}}$ `logs[l]` **then**
      Set $T \leftarrow k$
    end if
    Append v to `logs`
    $t \leftarrow t + 1$
  end if
  $k \leftarrow k + 1$
end while
return $\frac{\sum_{i=T}^{k} w_i}{(k-T+1)}$

Figure from Merity et al. Regularizing and Optimizing LSTM Language Models, 2018

AWD-LSTM

- AWD LSTM = AvSGD Weight Dropped LSTM

AWD-LSTM	PTB	WT2 (if differs)
lstm nlayers / lstm size	**3 / 1150**	
emb size	**400**	
tied softmax / var dropout	**yes**	
emb dropout prob	0.1	
lstm1 in dropout prob	0.4	0.65 ← larger vocab
lstm1/lstm2 out dropout prob	0.3/0.4	
rec dropconnect prob	**0.5**	
AR/TAR	2 / 1	
emb/other init ~U[-a,a]	0.1 / sqrt(H)	
nepochs	750	
bs	40	80
gradclip	0.25	
BPTT	~N(70,5), p=0.95	
	~N(35,5), p=0.05	

AWD-LSTM on 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	20M	–	78.6
Gal & Ghahramani (2016) - Variational LSTM	66M	–	73.4
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

Figure from Merity et al. Regularizing and Optimizing LSTM Language Models, 2018

AWD-LSTM Ablations

- Removing NT-AvSGD and Weight Dropping hurts most
  - But no recurrent regularization used instead
- Using full-sized embeddings (1150 instead of 400) hurts much!
- Ablation analysis says what helps little, not what helps much (not optimal hyperparameters)!

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-AvSGD	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

Figure from Merity et al. Regularizing and Optimizing LSTM Language Models, 2018