Recurrent Neural Networks

Lecture 4

Konstantin Yakovlev [1]

[1]MIPT
Moscow, Russia

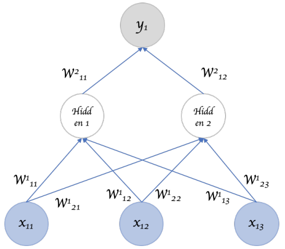
MIPT 2023

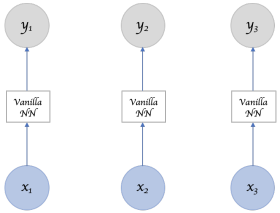
Recap

- Weight initialisation
  - Zero
  - Random
  - Xavier
- Batch Normalization
  - Layer norm
  - Instance norm
  - Group norm
- Convolutions
  - Forward
  - Backward
  - Parameters

Motivation

Input: a sequence of arbitrary length
Output: a sequence of arbitrary length
Proposition: MLP or CNN will not allow you to get a scalar output given an arbitrary sequence.





RNN for language modeling

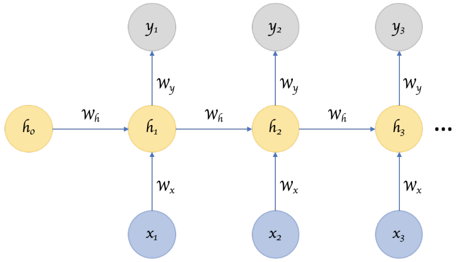
The probability of a sequence of a sequence of T words $(w_1, \dots w_T)$:

$$p(w_1, \dots, w_T) = \prod_{t=1}^{T} p(w_t | w_{<t}),$$

Notation

- $\mathbf{x}_t \in \mathbb{R}^d$ input word vector at timestep t.
- $\mathbf{W}_x \in \mathbb{R}^{D_h \times d}$ weights matrix used to condition the input word vector $\mathbf{x}_t$.
- $\mathbf{W}_h \in \mathbb{R}^{D_h \times D_h}$ weights matrix used to condition the output of the previous time-step $\mathbf{h}_{t-1}$.
- $\mathbf{h}_{t-1}$ output of the non-linear function at the previous time-step $t - 1$.
- $\sigma(.)$ activation function.
- $y_t = \mathrm{softmax}(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y)$ the output probability distribution over the vocabulary; $\mathbf{W}_y \in \mathbb{R}^{|V| \times D_h}$, $\mathbf{b}_y \in \mathbb{R}^{|V|}$.

Recurrent Neural Network (RNN)



Architecture

$$\mathbf{h}_t = \sigma(\mathbf{W}_x \mathbf{x}_t + \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{b}_h)$$

$$\mathbf{z}_t := \mathbf{W}_x \mathbf{x}_t + \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{b}_h$$

$$\mathbf{h}_t = \sigma(\mathbf{z}_t)$$

$$\mathbf{y}_t = \mathrm{softmax}(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y)$$

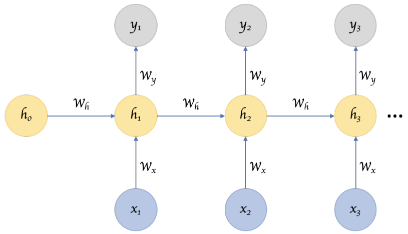
Criterion

$$\mathcal{L}(\mathbf{W}, \mathbf{b}) := -\mathbb{E}_{\mathbf{x}} \sum_{t=1}^{T} \underbrace{\sum_{j=1}^{|V|} \mathbb{I}[y_{t,j} = w_t] \log y_{t,j}}_{\mathcal{L}_t(\mathbf{x}_{<t}, \mathbf{W}, \mathbf{b})} \to \min_{\mathbf{W}, \mathbf{b}}$$

Problem

$$\nabla_{\mathbf{W}_h} \mathcal{L} = ?$$

RNN backpropagation



Deriving a gradient w.r.t. $\mathbf{W}_h$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_h} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial \mathbf{W}_h} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial \mathcal{L}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}_h} =$$
$$= \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial \mathcal{L}_t}{\partial \mathbf{h}_t} \left(\prod_{j=k+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right) \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}_h}.$$

Vanishing/Exploding gradients

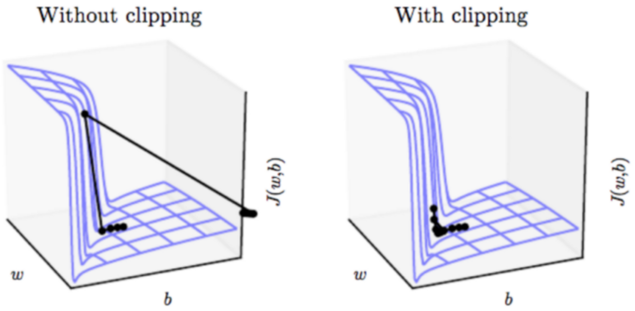
$$\|\frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}\| \leq \|\mathbf{W}_h\| \cdot \|\mathrm{diag}(\sigma'(\mathbf{z}_{j-1}))\| \leq \|\mathbf{W}_h\| \Rightarrow \| \prod_{j=k+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \| \leq \|\mathbf{W}_h\|^{t-k}.$$

Vanishing gradients: $\|\mathbf{W}_h\| < 1$. **Problem**: dras- tically reducing the learning quality of the model for far-away words.
Exploding gradients: $\|\frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}\| > 1$. **Problem**: if the gradient value grows extremely large, it causes an overflow.

Exploding gradients

Problem: if the gradient value grows extremely large, it causes an overflow.
Solution: gradient clipping.

```
$\mathbf{g} \leftarrow \frac{\partial \mathcal{L}}{\partial \mathbf{W}}$
if $\|\mathbf{g}\| \geq \text{threshold}$ then
    $\mathbf{g} \leftarrow \frac{\text{threshold}}{\|\mathbf{g}\|}\mathbf{g}$
end if
```



General overview of solutions

MLPs and Convolutional Networks

- Residual connections
- Batch normalisation
- Dropout
- Weight initialization

RNNs

- LSTM/GRU architectures
- Layer normalisation
- Dropout
- Weight parametrization

Weight Parametrization[1]

Challenge: gradient explosion/Vanishing
Solution: parametrization by unitary matrices Let $\sigma(.) = \mathrm{ReLU}(.)$

$$\|\frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}\| = \|\mathrm{diag}(\sigma'(\mathbf{z}_{j-1})) \underbrace{\mathbf{W}_h}_{\textit{orthogonal}} \| = 1$$

Therefore, we avoid exploding gradients.
Effective parametrization in the complex domain:

$$\mathbf{W}_h = \mathbf{D}_3\mathbf{R}_2\mathcal{F}^{-1}\mathbf{D}_2\mathbf{\Pi}\mathbf{R}_1\mathcal{F}\mathbf{D}_1,$$

- $\mathbf{D}$ – diagonal matrix, $\mathbf{D}_{jj} = e^{iw_j}$
- $\mathbf{R} = \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^*}{\|\mathbf{v}\|^2}$ – refection matrix
- $\mathbf{\Pi}$ – fixed radnom index permutation matrix
- $\mathcal{F}$ – Fourier transform

Interestingly, $\mathbf{D}, \mathbf{R}, \mathbf{\Pi}$ requires $\mathcal{O}(D_h)$ computations, while $\mathcal{F}$ requires $\mathcal{O}(D_h \log D_h)$. Vanilla RNN requires $\mathcal{O}(D_h^2)$ computation.
Activation function:

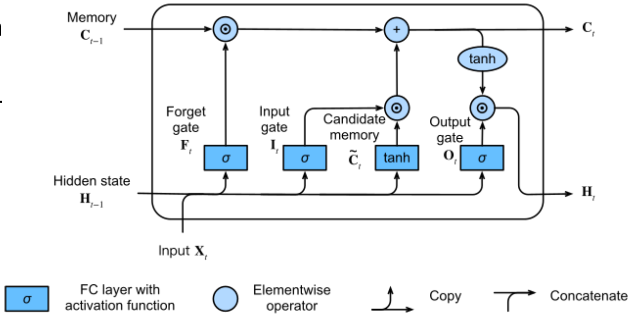
$$\sigma(z) = \mathrm{ReLU}(|z| + b)\frac{z}{|z|}.$$

[1] Arjovsky M., Shah A., Bengio Y. Unitary evolution recurrent neural networks, 2016

Long-Short-Term Memory network[2]

Challenge: RNN poorly models long-term dependencies
Solution: introduce a memorization mechanism

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}^i_x \mathbf{x}_t + \mathbf{W}^i_h \mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}^f_x \mathbf{x}_t + \mathbf{W}^f_h \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{o}_t &= \sigma(\mathbf{W}^o_x \mathbf{x}_t + \mathbf{W}^o_h \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\tilde{\mathbf{c}}_t &= \tanh(\mathbf{W}^c_x \mathbf{x}_t + \mathbf{W}^c_h \mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{c}_t &= \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tilde{\mathbf{c}}_t \\
\mathbf{h}_t &= \mathbf{o}_t \circ \tanh(\mathbf{c}_t)
\end{aligned}$$


Memory $\mathbf{C}_{t-1}$
$\mathbf{C}_t$
tanh
Forget gate $\mathbf{F}_t$
Input gate $\mathbf{I}_t$
Candidate memory $\tilde{\mathbf{C}}_t$
Output gate $\mathbf{O}_t$
σ
tanh
Hidden state $\mathbf{H}_{t-1}$
$\mathbf{H}_t$
Input $\mathbf{X}_t$
FC layer with activation function
Elementwise operator
Copy
Concatenate

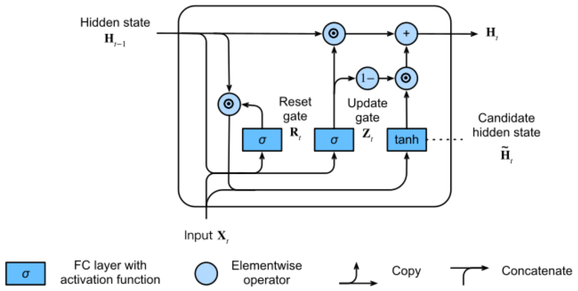
where $\mathbf{i}_i$ – input gate, $\mathbf{f}_i$ – forget gate, $\mathbf{o}_i$ – output gate, $\tilde{\mathbf{c}}_t$ – new memory cell, $\mathbf{c}_t$ – final memory cell
Note: Initialize $\mathbf{b}_f \gg 1$, $\mathbf{b}_i \gg 1$.

[2]Hochreiter S., Schmidhuber J. Long Short-Term Memory, 1997

Gated Recurrent Units[3]

Architecture:

$$\mathbf{u}_t = \sigma(\mathbf{W}_x^u \mathbf{x}_t + \mathbf{W}_h^u \mathbf{h}_{t-1} + \mathbf{b}_u)$$
$$\mathbf{r}_t = \sigma(\mathbf{W}_x^r \mathbf{x}_t + \mathbf{W}_h^r \mathbf{h}_{t-1} + \mathbf{b}_r)$$
$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{r}_t \circ \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_{t-1})$$
$$\mathbf{h}_t = (1 - \mathbf{u}_t) \circ \tilde{\mathbf{h}}_t + \mathbf{u}_t \circ \mathbf{h}_{t-1}$$



- $\mathbf{u}_t$ – update gate
- $\mathbf{r}_t$ – reset gate
- $\tilde{\mathbf{h}}_t$ – new memory

Note: initialize $\mathbf{b}_u \gg 1$, $\mathbf{b}_r \gg 1$.

[3] Cho et al. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches, 2014

Layer Normalization[4]

Challenge: batch normalization can not be applied to the case when the minibatches have to be small.

Solution: introduce layer normalization. Define $\mathrm{LN}: \mathbb{R}^D \to \mathbb{R}^D$, with two parameters $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{R}^D$.

$$\mathrm{LN}(\mathbf{z}; \boldsymbol{\alpha}, \boldsymbol{\beta})_i = \frac{z_i - \mu}{\sigma}\alpha_i + \beta_i, \ \ i = \overline{1, D},$$

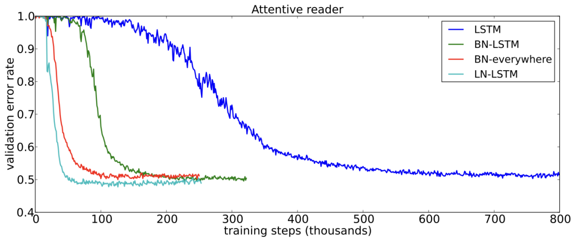
$$\mu = \frac{1}{D}\sum_{i=1}^{D} z_i, \quad \sigma = \sqrt{\frac{1}{D}\sum_{i=1}^{D}(z_i - \mu)^2}.$$

LSTM with layer normalization:

$$\begin{pmatrix} \mathbf{f}_t \\ \mathbf{i}_t \\ \mathbf{o}_t \\ \mathbf{g}_t \end{pmatrix} = \mathrm{LN}(\mathbf{W}_h \mathbf{h}_{t-1}; \boldsymbol{\alpha}_1, \boldsymbol{\beta}_1) + \mathrm{LN}(\mathbf{W}_x \mathbf{x}_t; \boldsymbol{\alpha}_2, \boldsymbol{\beta}_2) + \mathbf{b}$$

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

$$\mathbf{h}_t = \sigma(\mathbf{o}_t) \circ \tanh(\mathrm{LN}(\mathbf{c}_t; \boldsymbol{\alpha}_3, \boldsymbol{\beta}_3))$$

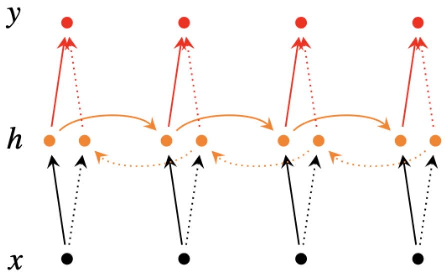


[4]Ba, Jimmy et al. Layer Normalization, 2016

Deep Bidirectional RNNs

Challenge: Consider the part-of-speech tagging task: $p(\mathbf{y}_{1:n}|\mathbf{x}_{1:n}) = \prod_{i=1}^{n} p(y_i|\mathbf{x}_{1:n})$. A vanilla RNN can not be conditioned on $\mathbf{x}_{i:n}$.
Solution: introduce a Bidirectional recurrent neural network (BRNN).

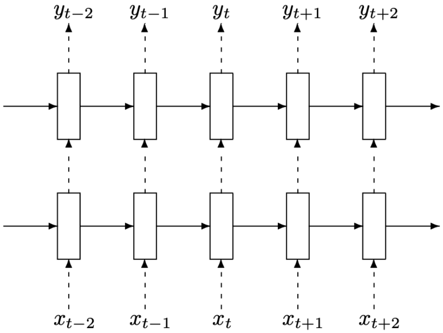
$$\vec{\mathbf{h}}_t = \sigma(\vec{\mathbf{W}}_x \mathbf{x}_t + \vec{\mathbf{W}}_h \mathbf{h}_{t-1} + \vec{\mathbf{b}})$$
$$\overleftarrow{\mathbf{h}}_t = \sigma(\overleftarrow{\mathbf{W}}_x \mathbf{x}_t + \overleftarrow{\mathbf{W}}_h \mathbf{h}_{t+1} + \overleftarrow{\mathbf{b}})$$
$$\mathbf{y}_t = \mathrm{softmax}(\mathbf{W}_y[\vec{\mathbf{h}}_t, \overleftarrow{\mathbf{h}}_t] + \mathbf{b}_y)$$

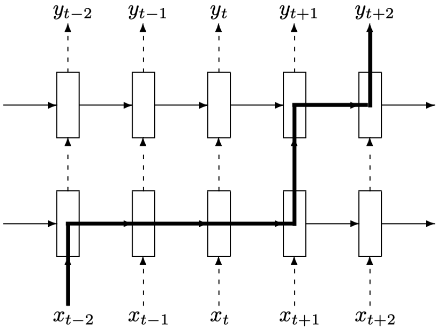


Extension: multi-layered BRNNs.

Naive dropout [5]

Idea: apply the dropout operator only to the non-recurrent connections.

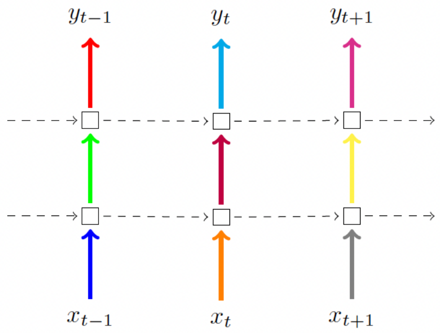




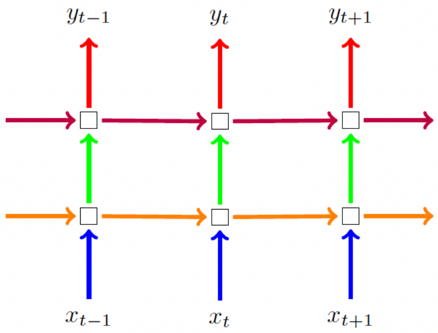
Model	Training set	Validation set
Non-regularized LSTM	71.6	68.9
Regularized LSTM	69.4	**70.5**

[5] Zaremba W. et al. Recurrent neural network regularization, 2014

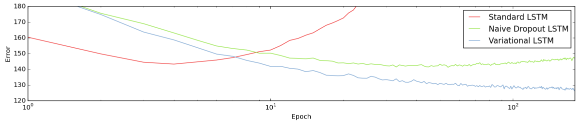
Variational Dropout[6]



(a) Naive dropout RNN



(b) Variational RNN



[6]Gal Y. et al. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks, 2016

Variational dropout: theoretical explanation

Approximate Variational Inference in Bayesian Neural Networks: Let $q(\mathbf{w})$ be ana approximating variational distribution:

$$\mathrm{KL}(q(\mathbf{w})||p(\mathbf{w}|\mathbf{X},\mathbf{Y})) \propto$$
$$-\sum_{i=1}^{n}\mathbb{E}_{q(\mathbf{w})}\log p(\mathbf{y}_i|\mathbf{f}(\mathbf{x}_i,\mathbf{w})) + \mathrm{KL}(q(\mathbf{w})||p(\mathbf{w})) \to \min_q$$

Variational Inference with RNNs:

$$\mathbf{w} = [\mathbf{m}_k]_{k=1}^{K} = [\mathbf{W}_h, \mathbf{W}_x, \mathbf{W}_y, \mathbf{b}_h, \mathbf{b}_y].$$

Approximating posterior distribution:

$$q(\mathbf{w}) = \prod_{k=1}^{K} q(\mathbf{w}_k),$$
$$q(\mathbf{w}_k) = p\mathcal{N}(\mathbf{w}_k|\mathbf{0},\sigma^2\mathbf{I}) + (1-p)\mathcal{N}(\mathbf{w}_k|\mathbf{m}_k,\sigma^2\mathbf{I})$$

Interpretation: Evaluating the model output $\mathbf{f}(\mathbf{x}_i,\hat{\mathbf{w}})$ with a sample $\hat{\mathbf{w}} \sim q(\mathbf{w})$ corresponds to randomly masking rows in each weight matrix during the forward pass if σ is small enough.

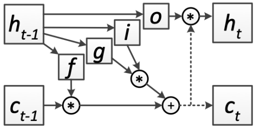
Predictive distribution:

$$p(\mathbf{y}^*|\mathbf{x}^*,\mathbf{X},\mathbf{Y}) \approx \mathbb{E}_{q(\mathbf{w})}p(\mathbf{y}^*|\mathbf{x}^*,\mathbf{w}) \approx$$
$$\frac{1}{K}\sum_{j=1}^{J} p(\mathbf{y}^*|\mathbf{x}^*,\hat{\mathbf{w}}_j), \quad \hat{\mathbf{w}}_j \sim q(\mathbf{w}).$$

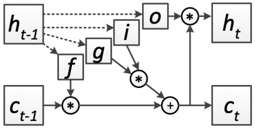
We perform dropout at test time and average results (MC dropout).

Recurrent Dropout without Memory Loss[7]

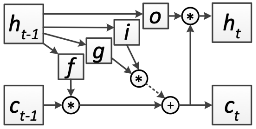
Challenge: information loss in memory cells of LSTMs when applying recurrent dropout
Solution: a novel dropout mechanism $\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ d(\tilde{\mathbf{c}}_t)$



(a) Moon et al., 2015



(b) Gal, 2015



(c) Ours

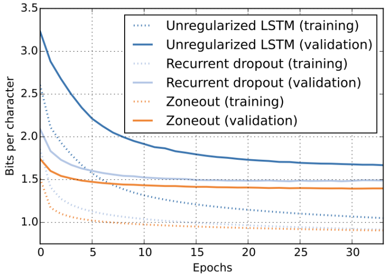
Dropout rate	Sampling	Moon et al. (2015)		Gal (2015)		Ours	
		Valid	Test	Valid	Test	Valid	Test
0.0	–	130.0	125.2	130.0	125.2	130.0	125.2
0.25	per-step	**113.0**	**108.7**	119.8	114.2	106.1	100.0
0.5	per-step	124.0	116.5	**118.3**	**112.5**	102.8	98.0
0.25	per-sequence	121.0	113.0	120.5	114.0	106.3	100.7
0.5	per-sequence	137.7	126.2	125.2	117.9	**103.2**	**96.8**

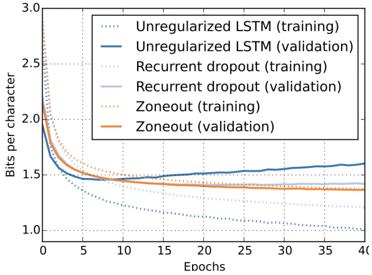
[7] Semeniuta S. et al. Recurrent Dropout without Memory Loss, 2016

Zoneout[8]

Challenge: the repeated application of the same transition operator can make the dynamics of an RNN sensitive to minor perturbations in the hidden state
Solution: regularize transition dynamics

$$\mathbf{c}_t = d_t^c \circ \mathbf{c}_{t-1} + (1 - d_t^c) \circ (\mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tilde{\mathbf{c}}_t),$$
$$\mathbf{h}_t = d_t^h \circ \mathbf{h}_{t-1} + (1 - d_t^h) \circ (\mathbf{o}_t \circ \tanh(\mathbf{c}_t))$$





[8] Krueger D. et al. ZONEOUT: Regularizing RNNs by Randomly Preserving Hidden Activations, 2017

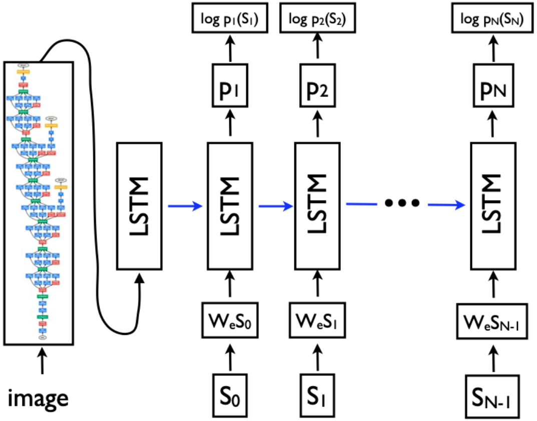
Application: Image Captioning[9]

Training:

$$-\sum_{(I,S)\in\mathfrak{D}} \log p(s_t|s_{<t}, I) \to \min_{\mathbf{W},\mathbf{b}}$$

Inference: generate a sentence given an image

- Sampling: $\hat{s}_t \sim p(s_t|s_{<t}, I)$
- Beam Search: iteratively consider the set of the k best sentences up to time t as candidates to generate sentences of size $t+1$, and keep only the resulting best k of them

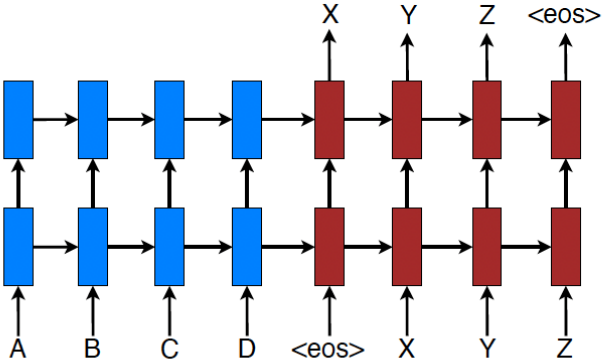


[9]Vinyals O. et al. Show and Tell: A Neural Image Caption Generator, 2015

Application: Neural Machine Translation[10]

Architecture
An encoder computes a representation **s** for each source sentence and an autoregressive decoder.

$$\log p(\mathbf{y}|\mathbf{x}) = \sum_{i=1}^{|\mathbf{y}|} \log p(y_i|\mathbf{y}_{<i}, \mathbf{s})$$



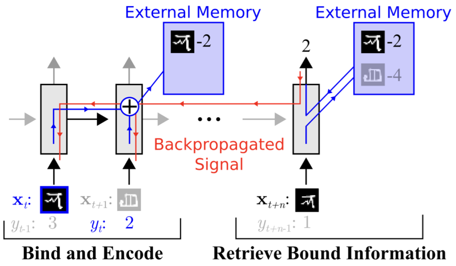
[10]Luong, et al. Addressing the Rare Word Problem in Neural Machine Translation, 2015

Application: Black-Box Meta Learning[11]

Consider a "few-shot learning" task
Challenge: fine-tuning is prone to poor learning
Solution: Memory-Augmented Neural Networks

$$\text{Supervised learning}: \quad f : x \mapsto y,$$
$$\text{Supervised meta-learning}: \quad f : (\mathfrak{D}_{\text{train}}, x) \mapsto y.$$

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{\mathfrak{D}} \sum_{t=1}^{|\mathfrak{D}|} \log \underbrace{p_{\theta}}_{\text{RNN}} (y_t | \mathbf{x}_t, \mathfrak{D}_{1:t-1}).$$



Retrieving a memory: Given a key $\mathbf{k}_t = f(\mathbf{x}_t)$

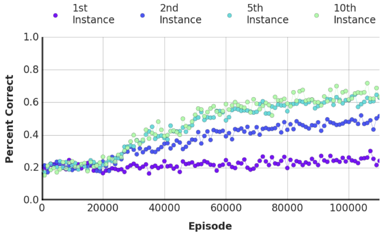
$$\mathbf{w}_t^{\text{read}}(i) = \frac{\exp(\mathrm{sim}(\mathbf{k}_t, \mathbf{M}_t(i)))}{\sum_j \exp(\mathrm{sim}(\mathbf{k}_t, \mathbf{M}_t(j)))}, \quad \mathbf{k}_t \in \mathbb{R}^d,$$

$$\mathbf{r}_t = \sum_i \mathbf{w}_t^{\text{read}}(i)\mathbf{M}_t(i), \quad \mathbf{M}_t \in \mathbb{R}^{m \times d}.$$

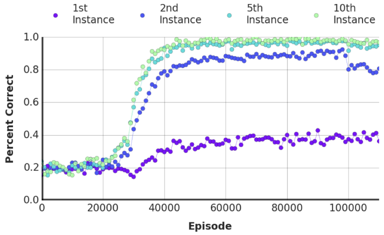
$$\mathbf{w}_t^{\text{usage}} = \gamma \mathbf{w}_{t-1}^{\text{usage}} + \mathbf{w}_t^{\text{read}} + \mathbf{w}_t^{\text{write}},$$
$$\mathbf{w}_t^{\text{write}} = \alpha \mathbf{w}_{t-1}^{\text{read}} + (1-\alpha)\mathbf{w}_{t-1}^{\text{least-used}},$$
$$\mathbf{w}_t^{\text{least-used}}(i) = \begin{cases} 0, & \mathbf{w}_t^{\text{usage}}(i) > \mathrm{bottom}_n(\mathbf{w}_t^{\text{usage}}) \\ 1, & \text{otherwise} \end{cases}$$
$$\mathbf{M}_t(i) = \mathbf{M}_{t-1}(i) + \mathbf{w}_t^{\text{write}}(i)\mathbf{k}_t$$

[11] Santoro A. et. al, One-shot Learning with Memory-Augmented Neural Networks, 2016

Memory-Augmented Neural Network: evaluation



(a) LSTM, five random classes/episode, one-hot vector labels



(b) MANN, five random classes/episode, one-hot vector labels

Model	Instance (% Correct)					
	1st	2nd	3rd	4th	5th	10th
Human	34.5	57.3	70.1	71.8	81.4	92.4
Feedforward	24.4	19.6	21.1	19.9	22.8	19.5
LSTM	24.4	49.5	55.3	61.0	63.6	62.5
MANN	**36.4**	**82.8**	**91.0**	**92.6**	**94.9**	**98.1**

Experimental setup: few-shot classification tasks of Ominiglot images.

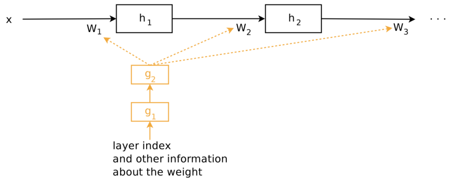
Results: The proposed MANN architecture substantially outperforms the baselines: feedforward RNN, LSTM, and human.

HyperNetworks[12]

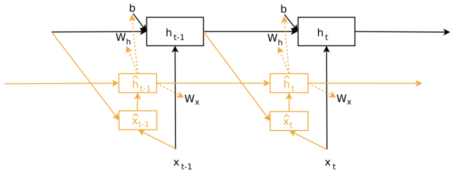
Idea: the normalization policy is fixed (ex. Layer Normalization). The learnable policy would give an increase in prediction accuracy.
Solution: adaptive weight generation with a hyper-network.

$$\mathbf{h}_t = \sigma(\mathbf{d}_h(\mathbf{z}_h) \odot \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{d}_x(\mathbf{x}_t) \odot \mathbf{W}_x \mathbf{x}_t + \mathbf{b}(\mathbf{z}_b)),$$
$$\mathbf{d}_h(\mathbf{z}_h) = \mathbf{W}_{hz}\mathbf{z}_h, \quad \mathbf{d}_x(\mathbf{z}_x) = \mathbf{W}_{xz}\mathbf{z}_x,$$
$$\mathbf{b}(\mathbf{z}_b) = \mathbf{W}_{bz}\mathbf{z}_b + \mathbf{b}_0,$$
$$\hat{\mathbf{x}}_t = [\mathbf{h}_{t-1}, \mathbf{x}_t],$$
$$\hat{\mathbf{h}}_t = \sigma(\hat{\mathbf{W}}_h \hat{\mathbf{h}}_{t-1} + \hat{\mathbf{W}}_x \hat{\mathbf{x}}_{t-1} + \hat{\mathbf{b}}),$$
$$\mathbf{z}_h = \hat{\mathbf{W}}_{hh}\hat{\mathbf{h}}_{t-1} + \hat{\mathbf{b}}_{hh},$$
$$\mathbf{z}_x = \hat{\mathbf{W}}_{hx}\hat{\mathbf{h}}_{t-1} + \hat{\mathbf{b}}_{hx},$$
$$\mathbf{z}_b = \hat{\mathbf{W}}_{hb}\hat{\mathbf{h}}_{t-1}.$$

General overview



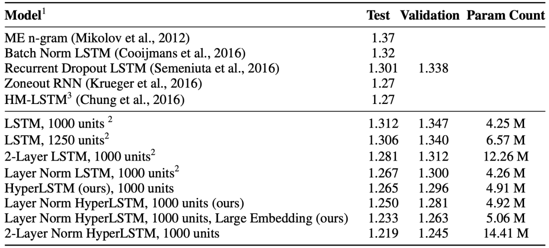
HyperRNNs architecture



[12]Ha D. et. al, HyperNetworks, 2016

HyperNetworks: evaluation

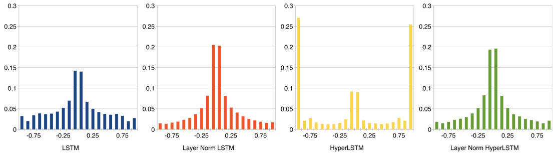
Evaluation on Penn Treebank



Model[1]	Test	Validation	Param Count
ME n-gram (Mikolov et al., 2012)	1.37		
Batch Norm LSTM (Cooijmans et al., 2016)	1.32		
Recurrent Dropout LSTM (Semeniuta et al., 2016)	1.301	1.338	
Zoneout RNN (Krueger et al., 2016)	1.27		
HM-LSTM[3] (Chung et al., 2016)	1.27		
LSTM, 1000 units [2]	1.312	1.347	4.25 M
LSTM, 1250 units[2]	1.306	1.340	6.57 M
2-Layer LSTM, 1000 units[2]	1.281	1.312	12.26 M
Layer Norm LSTM, 1000 units[2]	1.267	1.300	4.26 M
HyperLSTM (ours), 1000 units	1.265	1.296	4.91 M
Layer Norm HyperLSTM, 1000 units (ours)	1.250	1.281	4.92 M
Layer Norm HyperLSTM, 1000 units, Large Embedding (ours)	1.233	1.263	5.06 M
2-Layer Norm HyperLSTM, 1000 units	1.219	1.245	14.41 M

Layer Norm HyperLSTM significantly outperforms Layer Norm LSTM.

The normalized histogram plots of the hidden states



The normalization policy of HyperLSTM appears to be doing something very different from statistical normalization.

Mixture-of-Experts Layer[13]

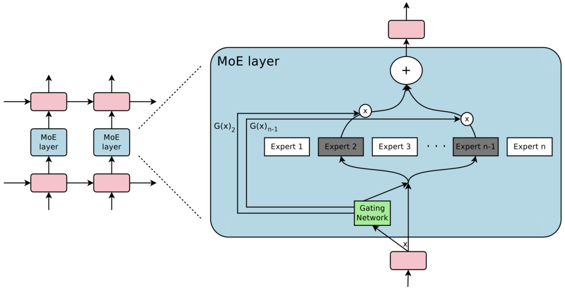
Challenge: an increase in model capacity is accompanied by an increase in computational cost.

Solution: Sparsely-Gated Mixture-of-Experts Layer (MoE).

Mixture-of-Experts layer

$$y = \sum_{i=1}^{n} \underbrace{G(x)_i}_{\text{gating network}} \cdot \underbrace{E_i(x)}_{\text{expert network}}$$

General overview



Noisy Top-k Gating Network

$$\mathbf{G(x)} = \mathrm{softmax}(\mathrm{topk}(\mathbf{xW}_g + \mathrm{softplus}(\mathbf{xW}_{\text{noise}})\epsilon)),$$
$$\epsilon \sim \mathcal{N}(\mathbf{0}_n, \mathbf{I}_n).$$

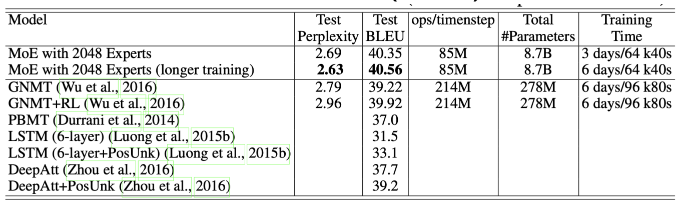
$$\mathrm{topk}(\mathbf{v}) = \begin{cases} v_i, & v_i \in \max_k(\mathbf{v}), \\ -\infty, & \text{otherwise} \end{cases}$$

The noise term helps with load balancing.

[13] Shazeer N. et. al, Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer, 2017

Mixture-of-Experts: evaluation

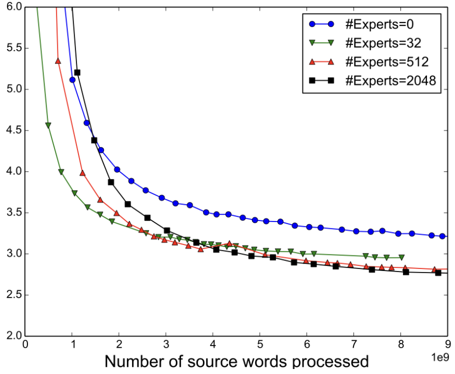
Machine translation WMT14(En-Fr)



Model	Test Perplexity	Test BLEU	ops/timenstep	Total #Parameters	Training Time
MoE with 2048 Experts	2.69	40.35	85M	8.7B	3 days/64 k40s
MoE with 2048 Experts (longer training)	**2.63**	**40.56**	85M	8.7B	6 days/64 k40s
GNMT (Wu et al., 2016)	2.79	39.22	214M	278M	6 days/96 k80s
GNMT+RL (Wu et al., 2016)	2.96	39.92	214M	278M	6 days/96 k80s
PBMT (Durrani et al., 2014)		37.0			
LSTM (6-layer) (Luong et al., 2015b)		31.5			
LSTM (6-layer+PosUnk) (Luong et al., 2015b)		33.1			
DeepAtt (Zhou et al., 2016)		37.7			
DeepAtt+PosUnk (Zhou et al., 2016)		39.2			

Each MoE layer contains 2048 feed-forward experts. There is a significant gain in BLEU score on top of the strong baselines.

Perplexity on WMT14(En-Fr)



As we increased the number of experts to approach 2048, the test perplexity of our model continued to improve.

Softmax Bottleneck[14]

Language Modeling with Softmax

$$p(x_t|c) = \mathrm{softmax}(\mathbf{h}_c^\top \mathbf{E}_w),$$

$$c = x_{<t}, \quad \mathbf{E}_w \in \mathbb{R}^{d\times|\mathcal{V}|},$$

$$\mathbf{H}_\theta = \begin{pmatrix} \mathbf{h}_{c_1}^\top \\ \dots \\ \mathbf{h}_{c_N}^\top \end{pmatrix}, \quad \mathbf{W}_\theta = \begin{pmatrix} \mathbf{w}_1^\top \\ \dots \\ \mathbf{w}_{|\mathcal{V}|}^\top \end{pmatrix},$$

$$\mathbf{A} = \|\log \pi(x_j|c_i)\|, \quad i = \overline{1, N}, j = \overline{1, |\mathcal{V}|},$$

where $\{c_i\}_{i=1}^N$ are all possible context in the natural language and $\pi(.|.)$ is data distribution.

Matrix factorization problem

$$\mathbf{H}_\theta \mathbf{W}_\theta^\top = \mathbf{A}', \quad \mathbf{A}' = \mathbf{A} + \underbrace{\mathrm{diag}(\boldsymbol{\lambda})}_{N\times N} \mathbf{1}_N \mathbf{1}_{|\mathcal{V}|}^\top.$$

Proposition (Softmax Bottleneck): if $d < \mathrm{rank}(A) - 1$ for any θ, there exists a context c such that $\pi(.|c) \neq p(.|c)$. **Solution: Mixture of Softmaxes**

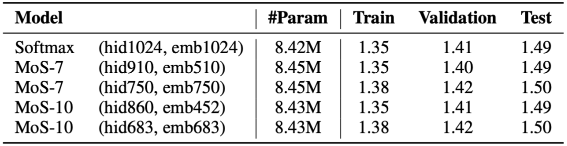
$$p(x_t|c) = \sum_{k=1}^K \pi(c)_k \mathrm{softmax}(\mathbf{h}_{c,k}^\top \mathbf{E}_w),$$

$$\sum_{k=1}^k \pi(c)_k = 1, \quad \hat{\mathbf{A}} = \underbrace{\log(\sum_{k=1}^K \mathbf{\Pi}_k \exp(\mathbf{H}_{\theta,k} \mathbf{W}_\theta^\top))}_{\text{high-rank}}.$$

[14]Yang Z. et. al, Breaking the Softmax Bottleneck: A High-Rank RNN Language Model, 2018

Softmax Bottleneck: evaluation

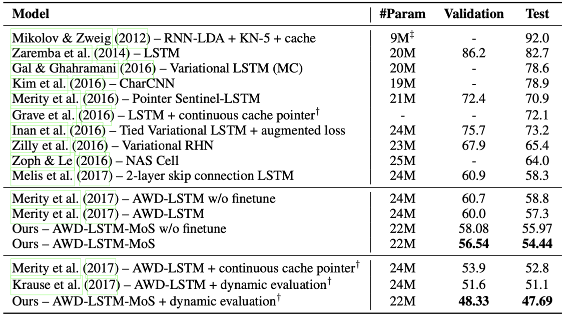
Character-level Language Modeling



Model		#Param	Train	Validation	Test
Softmax	(hid1024, emb1024)	8.42M	1.35	1.41	1.49
MoS-7	(hid910, emb510)	8.45M	1.35	1.40	1.49
MoS-7	(hid750, emb750)	8.45M	1.38	1.42	1.50
MoS-10	(hid860, emb452)	8.43M	1.35	1.41	1.49
MoS-10	(hid683, emb683)	8.43M	1.38	1.42	1.50

The models performs on par with each other, indicating that the softmax bottleteck problem diminishes when $\mathrm{rank}(A)$ is relatively small.

Language modeling on Penn Treebank



Model	#Param	Validation	Test
Mikolov & Zweig (2012) – RNN-LDA + KN-5 + cache	9M[‡]	-	92.0
Zaremba et al. (2014) – LSTM	20M	86.2	82.7
Gal & Ghahramani (2016) – Variational LSTM (MC)	20M	-	78.6
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 + continuous cache pointer[†]	-	-	72.1
Inan et al. (2016) – Tied Variational LSTM + augmented loss	24M	75.7	73.2
Zilly et al. (2016) – Variational RHN	23M	67.9	65.4
Zoph & Le (2016) – NAS Cell	25M	-	64.0
Melis et al. (2017) – 2-layer skip connection LSTM	24M	60.9	58.3
Merity et al. (2017) – AWD-LSTM w/o finetune	24M	60.7	58.8
Merity et al. (2017) – AWD-LSTM	24M	60.0	57.3
Ours – AWD-LSTM-MoS w/o finetune	22M	58.08	55.97
Ours – AWD-LSTM-MoS	22M	**56.54**	**54.44**
Merity et al. (2017) – AWD-LSTM + continuous cache pointer[†]	24M	53.9	52.8
Krause et al. (2017) – AWD-LSTM + dynamic evaluation[†]	24M	51.6	51.1
Ours – AWD-LSTM-MoS + dynamic evaluation[†]	22M	**48.33**	**47.69**

The proposed approach outperforms the baselines by a huge margin.

Summary

- RNN for Language modeling
- RNN backpropagation and related issues
- LSTM and GRU networks
- Layer Normalization and dropout mechanisms
- Applications: Neural Nachine Translation, Image Captioning, Black-Box Meta Learning
- HyperNetworks
- Mixture-of-Experts Layer
- Softmax Bottleneck