Breaking the Softmax Bottleneck via Monotonic Functions

Octavian Ganea, Sylvain Gelly, Gary Bécigneul, Aliaksei Severyn

Softmax Layer (for Language Models)

- Natural language as conditional distributions

- Parametric distributions & softmax:

$$P_\theta(x|c) = \frac{\exp \mathbf{h}_c^\top \mathbf{w}_x}{\sum_{x'} \exp \mathbf{h}_c^\top \mathbf{w}_{x'}} \approx P^*(\overset{\text{next word}}{x}|\overset{\text{context}}{c})$$

Softmax Layer (for Language Models)

- Natural language as conditional distributions

- Parametric distributions & softmax:

$$P_\theta(x|c) = \frac{\exp \mathbf{h}_c^\top \mathbf{w}_x}{\sum_{x'} \exp \mathbf{h}_c^\top \mathbf{w}_{x'}} \approx P^*(\overset{\text{next word}}{x}|\overset{\text{context}}{c})$$

- Challenge: Can we always find θ s.t. for all c : $P_\theta(X|c) = P^*(X|c)$?

Softmax Layer (for Language Models)

- Natural language as conditional distributions

- Parametric distributions & softmax:

next word   context

$$P_\theta(x|c) = \frac{\exp \mathbf{h}_c^\top \mathbf{w}_x}{\sum_{x'} \exp \mathbf{h}_c^\top \mathbf{w}_{x'}} \approx P^*(x|c)$$

- Challenge: Can we always find *θ* s.t. for all *c* : $P_\theta(X|c) = P^*(X|c)$?

No, when embedding size < label cardinality (vocab size) !

What is the Softmax Bottleneck (Yang et al, ‘18) ?

- **log-P matrix**: $$\mathbf{A}_P = \begin{bmatrix} \log P(x_1|c_1) & \log P(x_2|c_1) & \dots & \log P(x_M|c_1) \\ \log P(x_1|c_2) & \log P(x_2|c_2) & \dots & \log P(x_M|c_2) \\ \vdots & \vdots & \ddots & \vdots \\ \log P(x_1|c_N) & \log P(x_2|c_N) & \dots & \log P(x_M|c_N) \end{bmatrix} \in \mathbb{R}^{N \times M}$$

Label cardinality = Vocabulary size

Breaking the Softmax Bottleneck: A High-Rank RNN Language Model, Yang et al., ICLR 2018

What is the Softmax Bottleneck (Yang et al, ‘18) ?

- **log-P matrix**: $$\mathbf{A}_P = \begin{bmatrix} \log P(x_1|c_1) & \log P(x_2|c_1) & \dots & \log P(x_M|c_1) \\ \log P(x_1|c_2) & \log P(x_2|c_2) & \dots & \log P(x_M|c_2) \\ \vdots & \vdots & \ddots & \vdots \\ \log P(x_1|c_N) & \log P(x_2|c_N) & \dots & \log P(x_M|c_N) \end{bmatrix} \in \mathbb{R}^{N \times M}$$

- Then: $\mathrm{rank}(A_{P_\Theta}) \leq d + 1$

Number of labels = Vocabulary size

Breaking the Softmax Bottleneck: A High-Rank RNN Language Model, Yang et al., ICLR 2018

What is the Softmax Bottleneck (Yang et al, ‘18) ?

- **log-P matrix**: $\mathbf{A}_P = \begin{bmatrix} \log P(x_1|c_1) & \log P(x_2|c_1) & \dots & \log P(x \\ \log P(x_1|c_2) & \log P(x_2|c_2) & \dots & \\ \vdots & & & \\ \log P(& & & \end{bmatrix}$
- Then:

But $\mathbf{A}_{P^*}$ is likely full-rank, so $\mathbf{A}_{P^*} \neq \mathbf{A}_{P_\Theta}$ when d $<<$ $min(M, N)$

Breaking the Softmax Bottleneck: A High-Rank RNN Language Model, Yang et al., ICLR 2018

Breaking the Softmax Bottleneck [1]

- MoS [1] : Mixture of **K** Softmaxes

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

Breaking the Softmax Bottleneck [1]

- MoS [1] : Mixture of **K** Softmaxes

- Improves perplexity

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

Breaking the Softmax Bottleneck [1]

- MoS [1] : Mixture of **K** Softmaxes

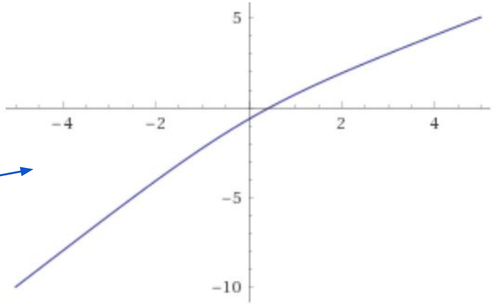
- Improves perplexity

- Slower than vanilla softmax: 2 - 6.4x
- GPU Memory: $\underbrace{\text{M x N}}_{\text{Vanilla softmax}}$ x **K** tensor

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

Breaking the Softmax Bottleneck [2]

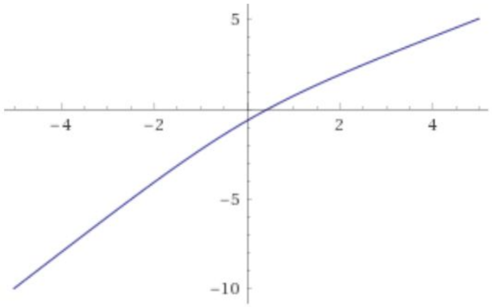
- Sig-Softmax [2] :



$$\mathrm{softmax}(\overbrace{2\mathbf{y} - \log(1 + \exp(\mathbf{y}))})$$

[2] Sigsoftmax: Reanalysis of the Softmax Bottleneck, S. Kanai et al., NIPS 2018

Breaking the Softmax Bottleneck [2]



- Sig-Softmax [2] :

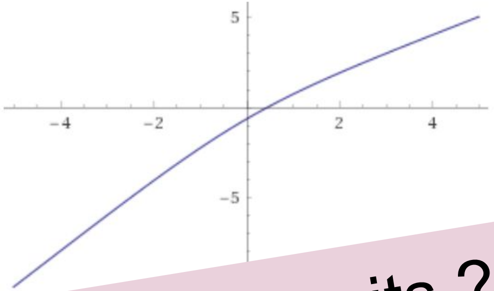
$$\mathrm{softmax}(2\mathbf{y} - \log(1 + \exp(\mathbf{y})))$$

- Small improvement over vanilla Softmax

[2] Sigsoftmax: Reanalysis of the Softmax Bottleneck, S. Kanai et al., NIPS 2018

Breaking the Softmax Bottleneck [2]

- Sig-Softmax [2] :



$$\mathrm{softmax}(2\mathbf{y} - \log(1 +$$

- tmax

Can we learn the best non-linearity to deform the logits ?

[2] Sigsoftmax: Reanalysis of the Softmax Bottleneck, S. Kanai et al., NIPS 2018

Can we do better ?

- Our idea - learn a pointwise monotonic function on top of logits:

$$p(y_i) = \frac{\exp(f(y_i))}{\sum_j \exp(f(y_j))}, \ \text{i.e. softmax}(f(\mathbf{y}))$$

Can we do better ?

- Our idea - learn a pointwise monotonic function on top of logits:

$$p(y_i) = \frac{\exp(f(y_i))}{\sum_j \exp(f(y_j))}, \text{ i.e. } \mathrm{softmax}(f(\mathbf{y}))$$

$f : \mathbb{R} \to \mathbb{R}$ should be:

Can we do better ?

- Our idea - learn a pointwise monotonic function on top of logits:

$$p(y_i) = \frac{\exp(f(y_i))}{\sum_j \exp(f(y_j))}, \text{ i.e. } \mathrm{softmax}(f(\mathbf{y}))$$

$f : \mathbb{R} \to \mathbb{R}$ should be:

1. With **unbounded image set** -- to model sparse distributions

Can we do better ?

- Our idea - learn a pointwise monotonic function on top of logits:

$$p(y_i) = \frac{\exp(f(y_i))}{\sum_j \exp(f(y_j))}, \text{ i.e. softmax}(f(\mathbf{y}))$$

$f : \mathbb{R} \to \mathbb{R}$ should be:

1. With **unbounded image set** -- to model sparse distributions
2. **Continuous** and **(piecewise) differentiable** -- for backprop

Can we do better ?

- Our idea - learn a pointwise monotonic function on top of logits:

$$p(y_i) = \frac{\exp(f(y_i))}{\sum_j \exp(f(y_j))}, \text{ i.e. softmax}(f(\mathbf{y}))$$

$f : \mathbb{R} \to \mathbb{R}$ should be:

1. With **unbounded image set** -- to model sparse distributions
2. **Continuous** and **(piecewise) differentiable** -- for backprop
3. **Non-linear** -- to break the softmax bottleneck

Can we do better ?

- Our idea - learn a pointwise monotonic function on top of logits:

$$p(y_i) = \frac{\exp(f(y_i))}{\sum_j \exp(f(y_j))}, \text{ i.e. } \mathrm{softmax}(f(\mathbf{y}))$$

$f : \mathbb{R} \to \mathbb{R}$ should be:

1. With **unbounded image set** -- to model sparse distributions
2. **Continuous** and **(piecewise) differentiable** -- for backprop
3. **Non-linear** -- to break the softmax bottleneck
4. **Monotonic** -- to preserve the ranking of logits

Can we do better ?

- Our idea - learn a pointwise monotonic function on top of logits:

$$p(y_i) = \frac{\exp(f(y_i))}{\sum_j \exp(f(y_j))}, \ \text{i.e. softmax}(f(\mathbf{y}))$$

$f : \mathbb{R} \rightarrow \mathbb{R}$ should be:

1. With **unbounded image set** -- to model sparse distributions
2. **Continuous** and **(piecewise) differentiable** -- for backprop
3. **Non-linear** -- to break the softmax bottleneck
4. **Monotonic** -- to preserve the ranking of logits
5. **Fast** and **memory efficient** -- comparable w/ vanilla softmax

Can we do better ?

- Our idea - learn a pointwise monotonic function on top of logits:

$$p(y_i) = \frac{\exp(f(y_i))}{\sum_j \exp(f(y_j))}, \text{ i.e. softmax}(f(\mathbf{y}))$$

$f : \mathbb{R} \to \mathbb{R}$ should be:

1. With **ot** -- to model sparse distributions
2. d **(piecewise) differentiable** -- for backprop
3. **n-linear** -- to break the softmax bottleneck
4. **Monotonic** -- to preserve the ranking of logits
5. **Fast** and **memory efficient** -- comparable w/ vanilla softmax

Theorem: these properties are not restrictive in terms of rank deficiency

Learnable parametric monotonic real functions

- A neural network with 1 hidden layer and positive (constrained) weights [3]

$$f(x) = \sum_{i=1}^{K} v_i \sigma(u_i x + b_i) + b, \ \text{s.t. } v_i, u_i \geq 0$$

- Universal approximator for all monotonic functions (when K is large enough !)

[3] Monotone and Partially Monotone Neural Networks, Daniels and Velikova, 2010, IEEE TRANSACTIONS ON NEURAL NETWORKS

Synthetic Experiment

- **Goal**: *separate softmax bottleneck from context embedding bottleneck*

Synthetic Experiment

- **Goal**: *separate softmax bottleneck from context embedding bottleneck*
- Sample N different categorical distributions with M outcomes:

$$P^*(\cdot|c_j) \sim \mathrm{Dir}(\alpha), \text{ for } j = 1, \ldots, N$$

Synthetic Experiment

- **Goal**: *separate softmax bottleneck from context embedding bottleneck*
- Sample N different categorical distributions with M outcomes:

$$P^*(\cdot|c_j) \sim \mathrm{Dir}(\alpha), \text{ for } j = 1, \ldots, N$$

- Goal:

$$P_\theta(x|c) = \frac{\exp \mathbf{h}_c^\top \mathbf{w}_x}{\sum_{x'} \exp \mathbf{h}_c^\top \mathbf{w}_{x'}} \approx P^*(x|c)$$

Synthetic Experiment

- **Goal**: *separate softmax bottleneck from context embedding bottleneck*
- Sample N different categorical distributions with M outcomes:

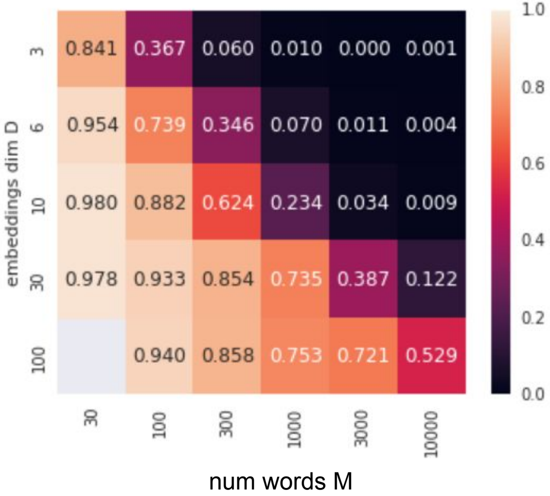
$$P^*(\cdot|c_j) \sim \mathrm{Dir}(\alpha), \text{ for } j = 1, \ldots, N$$

- Goal: $$P_\theta(x|c) = \frac{\exp \mathbf{h}_c^\top \mathbf{w}_x}{\sum_{x'} \exp \mathbf{h}_c^\top \mathbf{w}_{x'}} \approx P^*(x|c)$$
- Independent context embeddings; shared word embeddings

Synthetic Experiments - Mode Matching (α=0.01)

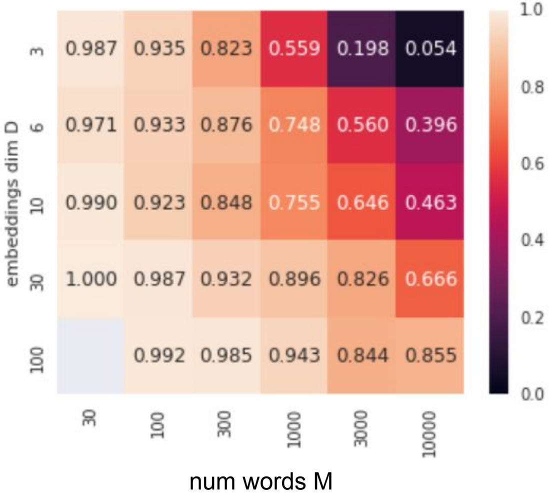
- Percentage of contexts c for which $\text{argmax}_x P^*(x|c) = \text{argmax}_x P_\Theta(x|c)$

Vanilla Softmax



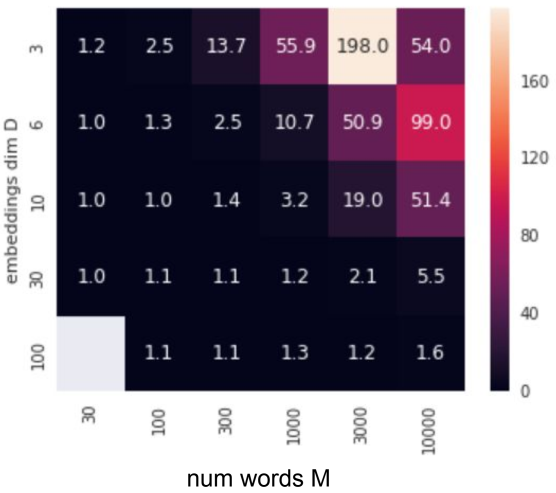
	30	100	300	1000	3000	10000
3	0.841	0.367	0.060	0.010	0.000	0.001
6	0.954	0.739	0.346	0.070	0.011	0.004
10	0.980	0.882	0.624	0.234	0.034	0.009
30	0.978	0.933	0.854	0.735	0.387	0.122
100		0.940	0.858	0.753	0.721	0.529

Monotonic fn (K=100)



	30	100	300	1000	3000	10000
3	0.987	0.935	0.823	0.559	0.198	0.054
6	0.971	0.933	0.876	0.748	0.560	0.396
10	0.990	0.923	0.848	0.755	0.646	0.463
30	1.000	0.987	0.932	0.896	0.826	0.666
100		0.992	0.985	0.943	0.844	0.855

Ratio 2nd / 1st

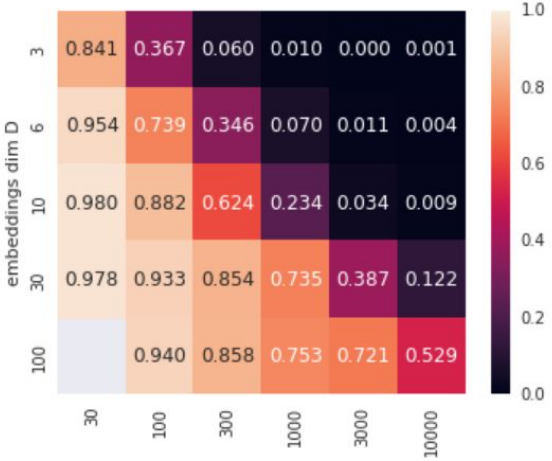


	30	100	300	1000	3000	10000
3	1.2	2.5	13.7	55.9	198.0	54.0
6	1.0	1.3	2.5	10.7	50.9	99.0
10	1.0	1.0	1.4	3.2	19.0	51.4
30	1.0	1.1	1.1	1.2	2.1	5.5
100		1.1	1.1	1.3	1.2	1.6

Synthetic Experiments - Mode Matching (α=0.01)

- Percentage of contexts c for which $\operatorname{argmax}_x P^*(x|c) = \operatorname{argmax}_x P_\Theta(x|c)$

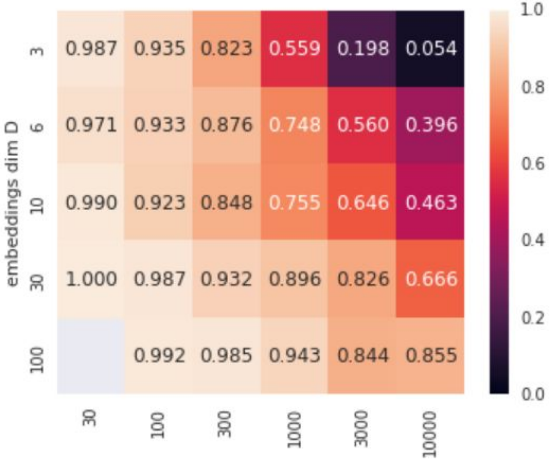
Vanilla Softmax



	30	100	300	1000	3000	10000
3	0.841	0.367	0.060	0.010	0.000	0.001
6	0.954	0.739	0.346	0.070	0.011	0.004
10	0.980	0.882	0.624	0.234	0.034	0.009
30	0.978	0.933	0.854	0.735	0.387	0.122
100		0.940	0.858	0.753	0.721	0.529

num words M

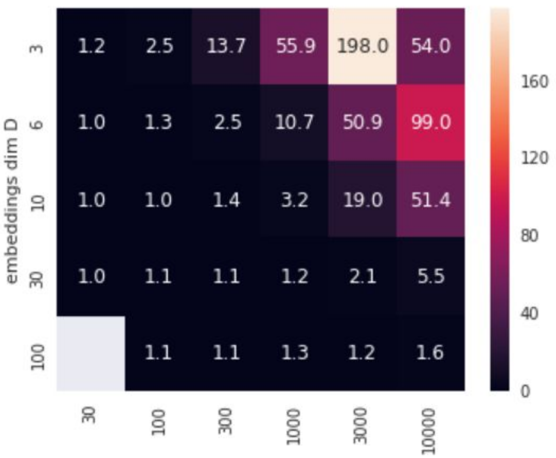
Monotonic fn (K=100)



	30	100	300	1000	3000	10000
3	0.987	0.935	0.823	0.559	0.198	0.054
6	0.971	0.933	0.876	0.748	0.560	0.396
10	0.990	0.923	0.848	0.755	0.646	0.463
30	1.000	0.987	0.932	0.896	0.826	0.666
100		0.992	0.985	0.943	0.844	0.855

num words M

Ratio 2nd / 1st



	30	100	300	1000	3000	10000
3	1.2	2.5	13.7	55.9	198.0	54.0
6	1.0	1.3	2.5	10.7	50.9	99.0
10	1.0	1.0	1.4	3.2	19.0	51.4
30	1.0	1.1	1.1	1.2	2.1	5.5
100		1.1	1.1	1.3	1.2	1.6

num words M

- Similar results for cross-entropy and other values of α

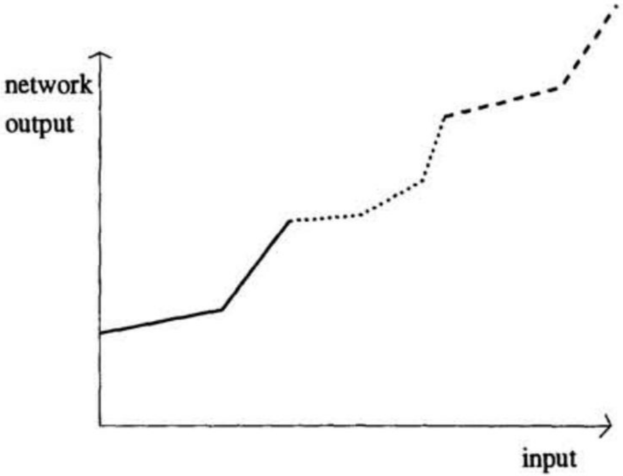
Piecewise Linear Increasing Functions (PLIF)

- NN w/ 1 hidden layer ⇒ memory hungry:
  - Tensor of size N x M x K on GPU , where K >= 1000

Piecewise Linear Increasing Functions (PLIF)

- NN w/ 1 hidden layer ⇒ memory hungry:
  - Tensor of size N x M x K on GPU , where K >= 1000

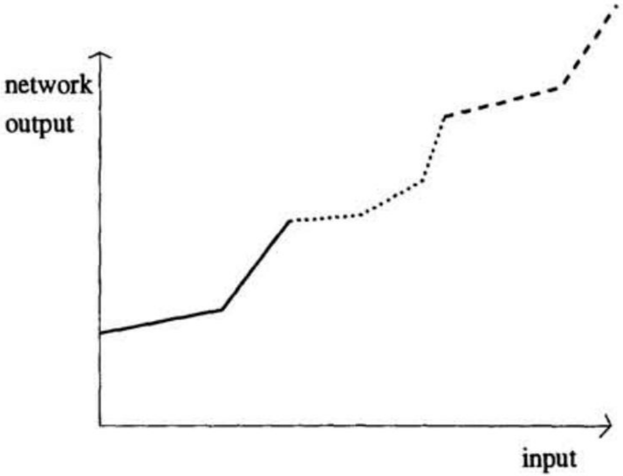
- **PLIF:**



Piecewise Linear Increasing Functions (PLIF)

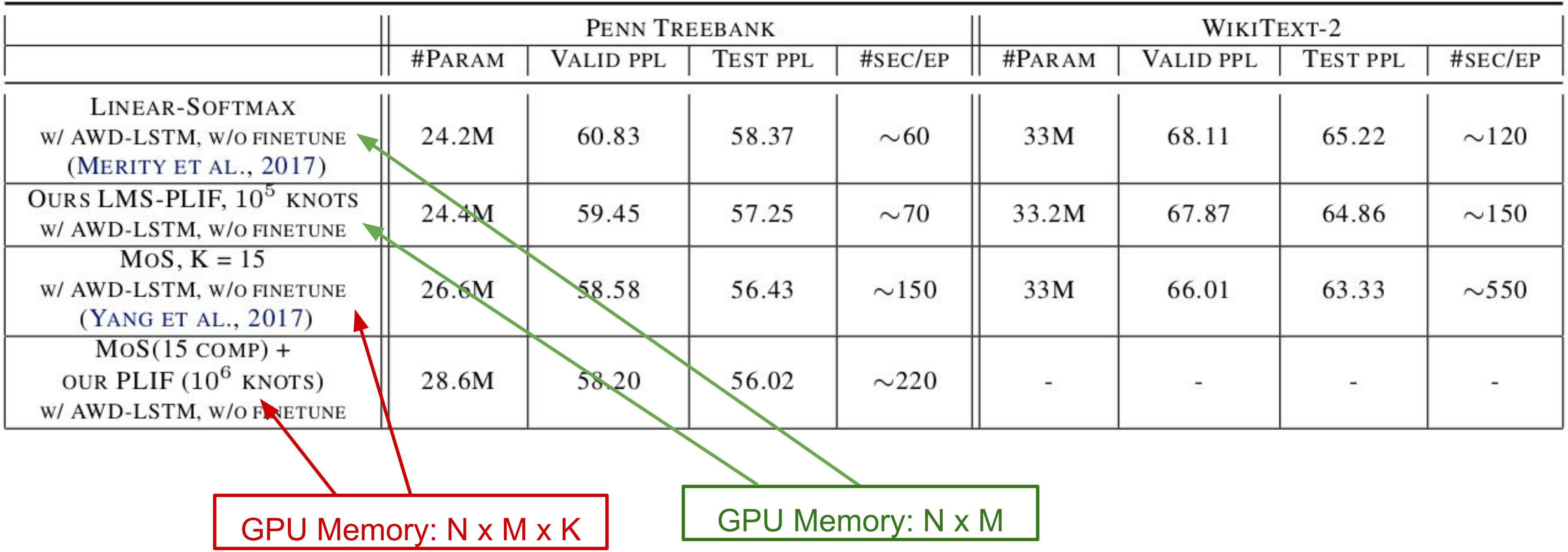
- NN w/ 1 hidden layer ⇒ memory hungry:
    - Tensor of size N x M x K on GPU , where K >= 1000

- **PLIF**:



- Forward & backward passes: just a lookup in two K dim vectors
- Memory and running time very efficient (comparable with Vanilla Softmax)

Language Modeling Results



	Penn Treebank				WikiText-2			
	#Param	Valid PPL	Test PPL	#Sec/EP	#Param	Valid PPL	Test PPL	#Sec/EP
Linear-Softmax w/ AWD-LSTM, w/o finetune (Merity et al., 2017)	24.2M	60.83	58.37	∼60	33M	68.11	65.22	∼120
Ours LMS-PLIF, 10^5 knots w/ AWD-LSTM, w/o finetune	24.4M	59.45	57.25	∼70	33.2M	67.87	64.86	∼150
MoS, K = 15 w/ AWD-LSTM, w/o finetune (Yang et al., 2017)	26.6M	58.58	56.43	∼150	33M	66.01	63.33	∼550
MoS(15 comp) + our PLIF (10^6 knots) w/ AWD-LSTM, w/o finetune	28.6M	58.20	56.02	∼220	-	-	-	-

GPU Memory: N x M x K

GPU Memory: N x M

Thank you!

Poster #23