Breaking the Activation Function Bottleneck

Through Adaptive Parameterization

Sebastian Flennerhag, Hujun Yin, John Keane & Mark Elliot
University of Manchester

Summary

Neural networks are non-linear only by virtue of a simple element-wise activation function. We develop an adaptive feed-forward layer that learns to adjust its parameterization conditional on the input. We present an adaptive LSTM that advances the state of the art for the Penn Treebank and WikiText-2 word-modeling tasks.

The Adaptive Feed-forward layer

The standard feed-forward layer, $f(\mathbf{x}) = \phi(W\mathbf{x}+\mathbf{b})$, learns an adaptive composite linear map: define $\mathbf{a} = W\mathbf{x}+\mathbf{b}$ and $\mathbf{g} = \phi(\mathbf{a})/\mathbf{a}$, with $G = \mathrm{diag}(\mathbf{g})$. Then

$$f(\mathbf{x}) = G(W\mathbf{x}+\mathbf{b}).$$

The layer learns a "prior" $(W, \mathbf{b})$ that it adapts through G. This adaptation is weak since ϕ is typically relatively linear over $p(\mathbf{a})$ and $\mathbf{g}$ shares parameters with $\mathbf{a}$. We break the activation function bottleneck by generalizing the feed-forward layer to q such adaptive blocks $\mathbf{x} \mapsto (G \circ A)(\mathbf{x})$ and parameterize the adaptation mechanism:

$$\begin{aligned}\tilde{f} &= \phi\left(D^{(q)}W^{(q-1)}\cdots W^{(1)}D^{(1)}\mathbf{x} + D^{(q+1)}\mathbf{b}\right)\\ D^{(j)} &= \mathrm{diag}(\pi^{(j)}(\mathbf{x}))\\ \pi(\mathbf{x}) &= \left[\pi^{(1)}(\mathbf{x})\ ;\ \cdots\ ;\ \pi^{(q+1)}(\mathbf{x})\right] = f(\mathbf{x})\end{aligned}$$

Policies

Singular Value Adaptation: $\tilde{f}(\mathbf{x}) = W^{(2)}DW^{(1)}\mathbf{x}$.

Learns a family of composite linear maps diagonalizable in an adaptation basis, along with a policy for selecting singular values.

IO-adaptation: $\tilde{f}(\mathbf{x}) = D^{(2)}WD^{(1)}\mathbf{x}$.

Learns to adapt the mean and variance of sub-matrices in a default parameter matrix. Can represent any other adaptation policy.

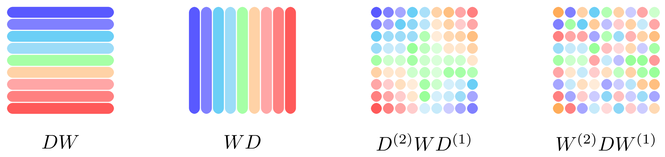

Figure 1: Adaptation policies. *Left:* output adaptation shifts the mean of each row in W; *center left:* input adaptation shifts the mean of each column; *center right:* IO-adaptation shifts mean and variance across sub-matrices; *Right:* SVA scales singular values.

Example: overcoming multi-modality

Predict y given x, with $x \sim 0.5\mathcal{N}(0.2, 0.7) + 0.5\mathcal{N}(0.8, 0.4)$ and $y = \sin(10x)$. Baseline: 2-layer feed-forward network. Adaptive model: 2-layer feed-forward network, first layer with SVA policy.

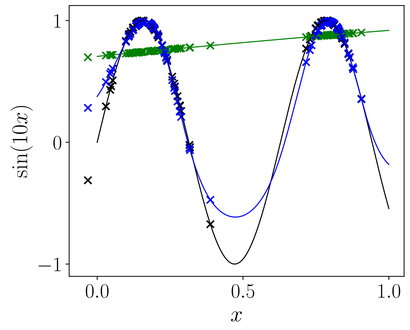

Figure 2: *Blue:* with adaptive feed-forward layer; *green:* static baseline.

The Adaptive LSTM

$$\begin{aligned}D_t^{(k)} &= U^k\mathbf{z}_t, \quad \mathbf{z}_t = m(\mathbf{v}_t, \mathbf{x}_t)\\ \mathbf{u}_t^s &= D_t^{(s,x,2)}WD_t^{(s,x,1)}\mathbf{x}_t + D_t^{(s,h,2)}VD_t^{(s,h,1)}\mathbf{h}_{t-1} + D_t^{(s,b,1)}\mathbf{b}\\ \mathbf{h}_t &= \sigma(\mathbf{u}_t^o)\cdot\tau(\mathbf{c}_t), \quad \mathbf{c}_t = \sigma(\mathbf{u}_t^f)\cdot\mathbf{c}_{t-1} + \sigma(\mathbf{u}_t^i)\cdot\tau(\mathbf{u}_t^z).\end{aligned}$$

m can be any parameterized model; for instance, a feed-forward layer or an LSTM. When stacking aLSTM layers, we develop a hybrid RHN-LSTM to ensure the adaptive model has full memory.

We test the aLSTM on Penn Treebank and WikiText-2. On Penn Treebank, the aLSTM outperforms the state of the art AWD-LSTM in 144 epochs, compared to ~500. On Wikitext-2, the aLSTM outperforms the AWD-LSTM in 160 epochs, compared to ~700.

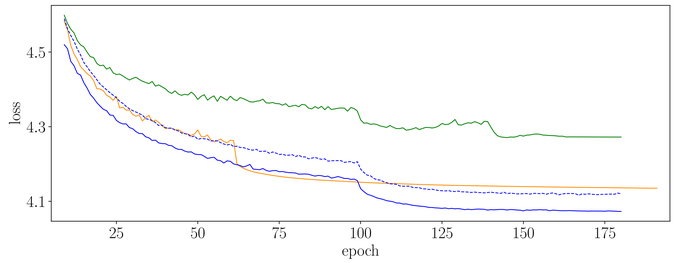

Figure 3: Val. curves on Penn Treebank. *Blue:* aLSTM; *orange:* AWD-LSTM; *green:* LSTM.

Model	Valid	Test
LSTM [10]	82.2	78.4
AWD-LSTM [8]	60.0	57.3
TG-SC LSTM [7]	60.9	58.3
aLSTM	**59.6**	**57.0**

Model	Valid	Test
VD-LSTM [6]	91.5	87.7
AWD-LSTM [8]	68.6	65.8
TG-SC LSTM [7]	69.1	65.9
aLSTM	**67.7**	**64.8**

Table 1: Perplexities (lower is better) on Penn Treebank and WikiText-2.

References

[1] Luca Bertinetto, João F Henriques, Jack Valmadre, Philip Torr, and Andrea Vedaldi. Learning feed-forward one-shot learners. In *Advances in neural information processing systems*, pages 523–531, 2016.

[2] Misha Denil, Babak Shakibi, Laurent Dinh, Nando De Freitas, et al. Predicting parameters in deep learning. In *Advances in neural information processing systems*, pages 2148–2156, 2013.

[3] David Ha, Andrew Dai, and Quoc V Le. HyperNetworks. *International Conference on Learning Representations*, 2017.

[4] David Ha and Douglas Eck. A neural representation of sketch drawings. In *International Conference on Learning Representations*, 2018.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–80, 1997.

[6] Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. In *International Conference on Learning Representations*, 2017.

[7] Gábor Melis, Chris Dyer, and Phil Blunsom. On the State of the Art of Evaluation in Neural Language Models. In *International Conference on Learning Representations*, 2018.

[8] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations*, 2018.

[9] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

[10] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent Neural Network Regularization. In *International Conference on Learning Representations*, 2015.