Supplementary to Kernelized Synaptic Weight Matrices

Lorenz K. Muller Julien N.P. Martel Giacomo Indiveri

Details on Network Architectures and Training Procedures

All models were implemented in Tensorflow (Abadi et al., 2015) or Theano (Theano Development Team, 2016) and (Dieleman et al., 2015).

Effective Dimensionality

The networks trained for figure 3 are Multilayer Perceptrons with layer-structure [784-500-500-10] with relu activation functions in the hidden layers and a softmax in the last layer. The cost function is the categorical cross-entropy. The learning rule is ADAM with exponential learning rate decay (factor 0.5 every 10000 steps) initialized at various levels and L_2 regularization (as described in the main paper). We use mini-batch gradient-descent with a batch size of 512. The code for this simulation is included in the supplement.

Channel Relationships

The networks trained for the results in table 1 are described here in some more detail. The two hidden layers of this network consist of 32 concatenated kernelNet layers with 28×28 and 14×14 neurons each respectively. The network structure is $[28 \times 28 - BatchNormalization - Dropout -$ KL32×28×28−KL32×14×14−10]. Batch Normalization was introduced in (Ioffe & Szegedy, 2015). All hidden units have selu activation functions, the output layers softmax. The selu activation function was introduced in (Klambauer et al., 2017). The cost function is categorical cross-entropy. The learning rule is ADAM with exponentially decaying learning rate with factor 0.95 every 2000 steps initialized at 0.01. We use mini-batch gradient descent with a batch size of 512.

MNIST Visualization

The networks trained to produce Figures 5 and 6 are an AutoEncoder and a Multilayer Perceptron that only differ in the last layer with layer structure [784 - Batch Normalization - 2000 (selu)- 2000 (selu)- 1600 (relu) - Dropout - KL2500(levelshift) - 2000(selu)- 2000 (selu)- 784/10 (sigmoid/softmax)]. In the kernelized layer we fix $\alpha_i > 0$ and the vectors $\vec{v}_i$ are not trained but fixed on a grid. The output of the Kernelized layer is scaled by a learned global factor. The cost function is the categorical cross-entropy. The learning rule is RMSProp (Tieleman & Hinton, 2012) with an exponentially decaying learning rate (factor 0.9 every 1000 steps) initialized at 0.001 / 0.0001 for the AutoEncoder / MLP. We use mini-batch gradient-descent with a batch size of 512. The code for this simulation is included in the supplement.

Pretrained Network Visualization

We pretrained a network using a standard implementation[1] of a ResNet. We cripple / augment the network as described in the main text and learn the parameters of the added layers with the same training procedure while fixing the previously learned parameters.

MovieLens Modelling

The network is fully described in the main paper. We trained the KernelNets with RPROP and the Sparse FC networks with L-BFGS-B (which respectively performed slightly better). An exception are the results on MovieLens-10M for which all results were obtained using RPROP (due to GPU memory limitations). In the L-BFGS-B training the algorithm was re-initialized every 50 parameter updates. Hyperparameter λ_2 lay in $[20, 160]$, λ_0 in $[0.004, 0.04]$. The code is available in this supplement. Note that in the code for MovieLens-10M (for comparability with the AutoRec paper) the squared error cost is scaled by 0.5 and the regularization parameters need to be scaled accordingly when training with those scripts.

[1] https://github.com/Lasagne/Recipes/blob/master/papers/deep_residual_learning/Deep_Residual_Learning_CIFAR-10.py

References

Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S., Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Goodfellow, Ian, Harp, Andrew, Irving, Geoffrey, Isard, Michael, Jia, Yangqing, Jozefowicz, Rafal, Kaiser, Lukasz, Kudlur, Manjunath, Levenberg, Josh, Mané, Dan, Monga, Rajat, Moore, Sherry, Murray, Derek, Olah, Chris, Schuster, Mike, Shlens, Jonathon, Steiner, Benoit, Sutskever, Ilya, Talwar, Kunal, Tucker, Paul, Vanhoucke, Vincent, Vasudevan, Vijay, Viégas, Fernanda, Vinyals, Oriol, Warden, Pete, Wattenberg, Martin, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Dieleman, Sander, Schlüter, Jan, Raffel, Colin, Olson, Eben, Sønderby, Søren Kaae, Nouri, Daniel, Maturana, Daniel, Thoma, Martin, Battenberg, Eric, Kelly, Jack, Fauw, Jeffrey De, Heilman, Michael, de Almeida, Diogo Moitinho, McFee, Brian, Weideman, Hendrik, Takács, Gábor, de Rivaz, Peter, Crall, Jon, Sanders, Gregory, Rasul, Kashif, Liu, Cong, French, Geoffrey, and Degrave, Jonas. Lasagne: First release., August 2015. URL http://dx.doi.org/10.5281/zenodo.27878.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Klambauer, Günter, Unterthiner, Thomas, Mayr, Andreas, and Hochreiter, Sepp. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pp. 972–981, 2017.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL http://arxiv.org/abs/1605.02688.

Tieleman, Tijmen and Hinton, Geoffrey. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.