Lecturer: Henning Sprekeler
Assistants: Robert Tjarko Lange

# Learning Dynamics in Deep Linear Networks

The solutions for these exercises (comprising source code, discussion and interpretation as an IPython Notebook) should be handed in before the **8th of November 2019 - 10.15 am** through the Moodle interface (in emergency cases send them to `robertlange0@gmail.com`).

## Exercise 1: Singular Value Mode Convergence

This exercise provides computational insight into the learning dynamics in "deep" linear networks & contrasts them with a "shallow" network. Throughout the first exercise you are given a set of feature and target matrices of second moments:

$$\mathbb{E}[xx^T] = \Sigma^x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbb{E}[yx^T] = \Sigma^{yx} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

Throughout this exercise the linear network is defined to have single hidden layer with 16 hidden units and $W^1 \in \mathbb{R}^{16\times 4}$ and $W^2 \in \mathbb{R}^{7\times 16}$:

$$\hat{y} = W^2 W^1 x \ ,$$

while the shallow network is a simple input-output mapping without any hidden layers and $W^{shallow} \in \mathbb{R}^{7\times 4}$:

$$\hat{y} = W^{shallow} x \ .$$

Furthermore, the singular value decomposition of the input-output covariance matrix can be written as:

$$\Sigma^{yx} = USV^T \ ,$$

where $S \in \mathbb{R}^{4\times 4}$ denotes the singular value diagonal matrix with non-zero elements $s_\alpha, \alpha = 1, \ldots, 4$.

1. Implement the mean weight update equations in the continuous time limit using forward Euler integration and setting $\Delta t = 0.1$. More specifically, write functions that take as an input the weight matrices and updates them according to the specified dynamics:

- For the single-hidden layer linear network:

$$\tau \frac{d}{dt} W^1 = W^{2^T} (\Sigma^{yx} - W^2 W^1 \Sigma^x) \tag{1}$$

$$\tau \frac{d}{dt} W^2 = (\Sigma^{yx} - W^2 W^1 \Sigma^x) W^{1^T} \tag{2}$$

- For the "shallow" network:

$$\tau \frac{d}{dt} W^{shallow} = \Sigma^{yx} - W^{shallow} \Sigma^x \tag{3}$$

2. Perform weight updates for the given $\Sigma^x$ and $\Sigma^{yx}$ and Gaussian initialised weight matrices (0 mean & 0.01 variance). Set the time constant $\tau = 1/\eta$ with learning rate $\eta = 0.4$ and $T = 15$ (i.e. each timestep corresponds to 10 updates). After each update of the weight matrices compute the SVD (e.g. using the NumPy implementation `np.linalg.svd()`) of the resulting weight matrix products

$$\Sigma^{\hat{y}x} = UA(t)V^T = W^2(t)W^1(t) \ .$$

   Plot the dynamics of the singular value modes $\hat{a}_\alpha(t)$ (the diagonal elements of $A(t)$) and compare them to the singular values $s_\alpha$ of the covariance matrix $\Sigma^{yx}$. What can you observe?

3. Implement the analytic solutions for the singular value mode dynamics. More specifically, write functions that take as an input the timestep as well as the singular value $s_\alpha$ and output the corresponding analytical singular value:

   - For the single-hidden layer linear network:

$$a_\alpha(t) = \frac{s_\alpha e^{2 s_\alpha t/\tau}}{e^{2 s_\alpha t/\tau} - 1 + s_\alpha / a_\alpha^0} \tag{4}$$

   - For the "shallow" network (Note that this equation was not derived in the analytical tutorial. For the interested reader, the proof follows similar steps.):

$$b_\alpha(t) = s_\alpha (1 - e^{-t/\tau}) + b_\alpha^0 e^{-t/\tau} \tag{5}$$

   Compute the analytical values of the singular values by setting $a_\alpha^0 = b_\alpha^0 = 0.001$. Again, plot them for the different time steps and compare them to the empirical ones calculated in the previous part. How well do theory and simulation align?

## Exercise 2: Deeper (Non-)Linear Networks with AutoDiff

In this exercise we will probe whether the theoretical results & insights translate to deeper linear networks as well as non-linear networks. Deep Learning is powered by reverse mode automatic differentiation, computational graphs & stochastic gradient descent algorithms. Therefore, we recommend you to use PyTorch[1] due to its easy installation, documentation & recent popularity in academic research. But please feel free to make use of any AutoDiff software you feel comfortable with (e.g. Keras, TensorFlow,

[1] If you are unfamiliar with PyTorch, you might find this introductions useful: `https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html`

JAX or even NumPy - but please use a current version). Don't worry, the trained networks are fairly small, do not involve convolutions & can therefore easily be trained on any CPU.

In the supplementary skeleton notebook (*skeleton-comp-practical.ipynb*) we provide a hierarchical data-generation process `DiffuseTreeSampler()` (see section A of skeleton) which we use throughout this exercise. Furthermore, we provide a simple single hidden layer example of a linear network (see section B of skeleton) which can guide your solutions. **Finally and most importantly, you can simply compliment the skeleton functions for part 1. and 2.**.

1. Code a variable depth Deep Linear Network class that takes as an input a list of hidden units for each layer (e.g. `hidden_units=[16, 32, 32, 16]`).

2. Complement (or rewrite if you are not using PyTorch) the learning loop defined in the skeleton (`linear_net_learning()`). This will require multiple small steps:

   a) Shuffle the data ordering at the beginning of each epoch.

   b) Perform the forward pass at each iteration for a selected feature, target pair.

   c) Calculate the corresponding loss using the Mean Squared Error (MSE) Loss.

   d) Reset the parameter gradients, perform a backwards pass to calculate the current gradients & update the parameters with the help of an optimizer object.

3. Create a dataset by creating an instance of `DiffuseTreeSampler()` and sampling from it. Afterwards, compute the SVD of $\Sigma^{yx}$ and assert that $\Sigma^{x} = \mathbf{1}$. Train a deep linear network:

   a) Define a linear network with three hidden layers (`hidden_units=[64, 128, 128]`).

   b) Instantiate an Stochastic Gradient Descent optimizer object with learning rate $\eta = 0.5$.

   c) Define the MSE loss function.

   d) Run the `linear_net_learning()` loop for 1000 epochs.

   After each epoch (loop over the entire dataset) compute the SVD of the resulting matrix of second moments $\Sigma^{\hat{y}x}$. Note that in this special case and since $x$ is the identity matrix, $\Sigma^{\hat{y}x}$ corresponds to the product of weight matrices and can be simply obtained by forward propagating through the network, i.e. $\hat{y}$. Plot the evolution of the singular values over the course of the learning epochs as well as the static singular values of $\Sigma^{yx}$.

4. Do the results hold up for both deep linear and non-linear networks? Implement a ReLU activation after every linear layer. Which singular values converge first?