Interleaving Stability of Deep Learning Algorithms

Maria Walch

August 24, 2022

Abstract

We develop a framework for using the interleaving distance to study the stability of a broad class of deep learning algorithms. We present bounds on how closely the output these algorithms produce under the presence of noise approximates the output they would produce for noiseless data. Also, we investigate the stability of the 'optimal' set of weights $\{\theta_{ln}\}$ after the instances of training $\mathcal{I}_{\text{train}}$. Furthermore, we explore the equivariance of deep learning algorithms with respect to input transformations.

1 Deep Learning

1.1 Weight Optimization Problem

For either classification or regression, we assume that our input vector $\mathbf{x}$ takes values $x_i \in \{x_1, ..., x_n\}$ in $\mathbb{R}$, thus we assume an Euclidian input space. Given a choice of weight matrices $W_{nm} \in \mathbb{R}^{n \times m}$, a deep learning algorithm solves the **weight optimization problem**. The weight optimization problem is given by a tuple $(\theta_{nl}, \mathcal{L})$, where θ_{nl} denotes the complete set of weights for each node n in each layer l; and $\mathcal{L}$ is the loss measure that is calculated by the mean over all N training examples $\{y_j, \hat{y}_j\}$. Intuitively, $\mathcal{L}$ acts as a penalty for bad outputs y with respect to the ground truth $\hat{y}$ as well as bad (initializations of) weights. Its form depends on the task that is assigned to the network. For regression, the most common form is quadratic loss, whereas for classification, one often takes cross entropy. The definition of the weight optimization problem does not include any specification of how the optimization problem is solved (most commonly backpropagation and gradient descent). This is done iteratively and the problem with weight optimization is, that one can only hope to find a local minimum of the loss measure $\mathcal{L}$, not a global one.

1.2 Feedforward Training Problem

Formally we define the **feedforward training problem** to be a function that maps the Euclidian input space to a weight optimization problem $(\theta_{nl}, \mathcal{L}, \{y_j, \hat{y}_j\})$.

Optimizing this objective involves finding a set of weights θ_{nl} that minimizes (for the regression task)

$$\mathcal{L}(W,b) = \frac{1}{2N}\sum_{j=1}^{N} ||\phi(\sum_{l} hl^t W_{lp})_j - \hat{y}_j||^2. \tag{1}$$

One entry p of the jth output vector y_j is given by $\phi(\sum_l hl^t W_{lp})$, thus the underline index in Eq. 1 denotes that all entries of a given output vector y_j are considered. The autoencoder feedforward training problem is special in the sense that the target vector is equal to the input vector, thus $\hat{y}_j = x_j$. Furthermore, the architecture is chosen such that it consists of two parts: the encoder and the decoder. The encoder performs a dimension reduction of the input data resulting in the so-called *latent representation*. Subsequently, the decoder uses the latent representation to reconstruct the output, usually in an architecturally mirrored way.

2 The Encoder

Shiebler [1] showed that

3 Feedforward Vanilla Autoencoder

A Minimal Working Example

Our minimal working example is given by a simple fully connected feedforward network with one hidden layer, as depicted in figure 1. The outputs of subsequent layers are obtained via the inter-layer connecting weights (for our minimal example in Fig. 1) as:

$$h_l^t = \phi(\sum_r x_r^t \cdot W_{rl}) \quad \text{and} \quad y_p^t = \phi(\sum_l h_l^t W_{lp}) \tag{2}$$

where h_l^t and y_k^t are the row vectors obtained from h_l and y_k via transposition.The nonlinear function ϕ is called the activation function and might vary for each layer. Given $n, m, k \in \mathbb{N}$, a three layer feedforward training algorithm constructs a k vector of outputs y_k and optimizes the real (or even complex) valued matrices of weights W_{nm} and W_{km}. The loss measure $\mathcal{L} : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$ assigns a real-valued loss value to each sample based on the difference between the model's prediction and the ground truth. The form of the associated loss function depends on the task assigned to the network. In the following, we only consider supervised learning and differentiate between two tasks: regression and classification. For regression, the quadratic loss function is most commonly used:

$$L(\hat{y}_j, y_j) = \frac{1}{2}||y_j - \hat{y_j}||^2. \tag{3}$$

The Loss measure $\mathcal{L}$ is then calculated by the mean over all N training examples $\{y_j, \hat{y}_j\}$:

$$\mathcal{L}(W, b) = \frac{1}{2N} \sum_{j=1}^{N} ||y_j - \hat{y}_j||^2, \tag{4}$$

where W denote the parameters or weights of the network and b denotes bias terms (not shown in Fig. 1). For classification, the cross entropy loss is function is most commonly used:

$$L = -(y_j \log(\hat{y}_j) + (1 - y_j) \log(1 - \hat{y}_j)). \tag{5}$$

The loss measure $\mathcal{L}$ is then taken to be the average of the loss functions for all labeled pairs:

$$\mathcal{L} = \frac{1}{N} - \sum_{j=1}^{N} (y_j \log(\hat{y}_j) + (1 - y_j) \log(1 - \hat{y}_j)). \tag{6}$$

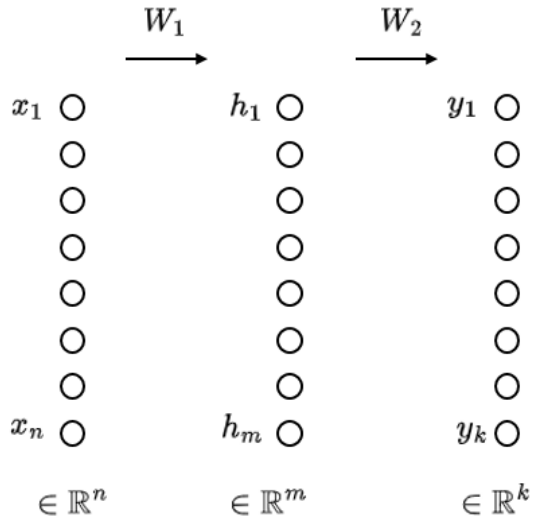

Figure 1: Simple feedforward network with three layers. Despite the graphical representation, the cardinality of nodes within the layers need not be equal, $n \neq m \neq k$. Inter-layer connections are not shown for clarity. W_1 and W_2 carry the inter-layer connectivity information for each edge, e.g. W_{ij}^1 for the weight connecting neuron i in layer 1 to neuron j in the hidden layer.