

[LE] [RO] [LE,RO]1 noheader

*./img/01\_intro/./img/02\_ern/./img/03\_gcal/./img/04\_char/./img/05\_n/./img/06\_summary/*

## Contents

*./chp/01<sub>i</sub>ntro/chp.tex*

*./chp/02<sub>ern</sub>/chp.tex*

*./chp/03<sub>h</sub>gal/chp.tex*

*./chp/04<sub>char</sub>/chp.tex*

*./chp/05<sub>n</sub>n/chp.tex*

*./chp/06\_summary/chp.tex*



# 1 Gradient Descent and Backpropagation

The training process is guided by a so-called loss (or objective) function  $L$ . Since neural networks can be trained for different purposes (classification, regression, reconstruction, data generation, etc.) a multitude of loss functions can be found in the literature. For a given input, the loss function takes the activations of the network's output layer and a corresponding desired output (ground truth) as input and generates a single real number to be minimized. For reconstruction, for example, the mean-squared error function

$$L_{MSE}(\vec{x}, \vec{y}) = \sum_n (x_i - y_i)^2 \quad (1)$$

is often used. If the desired output (truth) has been generated in advance by experts or external sources one speaks of supervised learning. One variant of supervised learning is self-supervised learning in which the ground truth and input data are the same, cf. Sec. ??.

Calculating the gradient of  $L$  with respect to each network parameter shows the impact that changing the parameters has on the loss. Thus, the training process is the optimization problem of minimizing the loss by varying the network parameters. The most effective way to do so is to change a parameter  $p$  in the opposite direction of the loss's gradient with respect to  $p$ , i.e. in the direction of steepest (gradient) descent:

$$p' = p - \eta \nabla_p L, \quad (2)$$

with a (small) step size  $\eta$ , called learning rate. According to this update rule, the network is guaranteed to find a (local) minimum of the loss function. The process is illustrated in Fig. ??.

[height=8cm]gradient<sub>descent</sub>.png

Figure 1: Visualization of the gradient descent algorithm for a model with parameters  $\Theta_0$  and  $\Theta_1$ . Figure taken from Ref. [?].

As it is desired for neural networks to generalize on a population of data describing a problem, the learning process needs to take a sufficiently large (representative) subset of the population into account. The network parameters are then updated via the average of gradients for all samples in this (training) set. Thereby, the network has to consider the whole population

for each training step. Since this updating procedure requires a lot of calculation and memory usage, a variant called *stochastic* gradient descent (SGD) is commonly used. SGD considers small, randomly selected, subsets of the training data (usually 32 or 64 samples per subset), called mini-batches to calculate the average gradient and update the parameters. Therefore, the network is able to learn faster, but also less precisely when converging towards the minimum.

The gradients of the loss function are calculated by tracking the data transformations of the forward pass and subsequently calculating partial derivatives  $\delta L/\delta b$  and  $\delta L/\delta w$  at each node. As the transformation of a data point  $\vec{x}$  in the network can mathematically be represented by function applications  $a(\vec{x})$  at each layer, a succession of layers is simply a composition of functions  $a_n \circ \dots \circ a_1 \circ a_0(\vec{x})$ , starting from the output layer, exploiting the chain rule of multi-variate calculus and the compositional nature of the network layers. This procedure, referred to as backward propagation is illustrated in Fig. ??; the loss is labeled  $E$ , the activation  $y_i$  and weights denoted as  $\delta E/\delta w$ . Note that biases are ignored.

[height=8cm]backprop.png

Figure 2: Illustration of forward and backward pass and the components of the computational graph used during neural network training process. Biases are ignored. Figure taken from Ref. [?].

The backpropagation algorithm can be developed by starting from the loss function  $L$  and successively applying the chain rule. At the beginning, the error term of a neuron in the output layer  $L$  is defined as

$$\delta_i^L = \frac{\partial L}{\partial z_i^L} = \frac{\partial L}{\partial a_i^L} \frac{\partial a_i^L}{\partial z_i^L}, \quad (3)$$

for the non-linear activation  $a_i^l = \sigma(z_i^l)$  and  $z_i^l = W^l \vec{a}^{l-1} + b_i$ . Thus, the rule for computing the previous layer's error can be found:

$$\delta_i^{L-1} = \frac{\partial L}{\partial z_i^{L-1}} = \underbrace{\frac{\partial L}{\partial a_i^L} \frac{\partial a_i^L}{\partial z_i^L}}_{\delta_i^L} \frac{\partial z_i^L}{\partial \vec{a}^{L-1}} \frac{\partial \vec{a}^{L-1}}{\partial z_i^{L-1}} = \delta_i^L W^L \sigma'(z^{L-1}). \quad (4)$$

Thus for any layer  $l$  one obtains:

$$\delta_i^l = (W^{l+1})^T \vec{\delta}^{l+1} \sigma'(z^l). \quad (5)$$

The partial derivatives of the loss with respect to the biases and weights of any layer can be found in a similar manner:

$$\frac{\partial L}{\partial w_{ij}^l} = \underbrace{\frac{\partial L}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l}}_{\delta_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l} = \delta_i^l a_j^{l-1} \quad (6)$$

for the weights and

$$\frac{\partial L}{\partial b_i^l} = \underbrace{\frac{\partial L}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l}}_{\delta_i^l} \underbrace{\frac{\partial z_i^l}{\partial b_i^l}}_1 = \delta_i^l \quad (7)$$

for the biases. Eq. ?? and Eq. ?? can be written in vector form using the Hadamard product  $\odot$  (element-wise multiplication) and the gradient of the loss with respect to an activation  $\nabla_{a^L} L$ :

$$\vec{\delta}^L = \nabla_{a^L} L \odot \sigma'(\vec{z}^L) \quad (8)$$

$$\vec{\delta}^l = ((\mathbf{W}^{l+1})^T \vec{\delta}^{l+1}) \odot \sigma'(\vec{z}^l). \quad (9)$$

## 2 Principle Component Analysis

While Fig. ?? schematically shows a deep undercomplete autoencoder, there exists an even simpler type with a single hidden layer. By considering linear neuron activation  $\vec{a}(\vec{x}) = \mathbf{W}\vec{x} + \vec{b}$ , it turns out that this simple network learns an encoding function similar to a dimensionality reduction technique called principal component analysis (PCA). In PCA the original feature axes of a data set are swapped with new ones (as linear combinations of the originals) that are aligned with the directions of greatest variance in the data [?]. Mathematically, this is accomplished by eigendecomposition  $\mathbf{C} = \mathbf{P}\lambda\mathbf{P}^T$  of the unbiased covariance matrix  $\mathbf{C}$  associated with data points  $\vec{x}_i$  contained in the data set  $\mathbf{X}$ .  $\mathbf{C}$  is defined as

$$\mathbf{C} = \frac{1}{m-1} \mathbf{X}\mathbf{X}^T. \quad (10)$$

The eigenvectors  $\vec{v}_i$  corresponding to the largest eigenvalues  $\lambda_i$  resemble the  $n$  principal components, where  $m$  is the number of input/output dimensions and  $n \leq m$ . The eigenvectors and eigenvalues can be found according to

$$\mathbf{C} \cdot \mathbf{P} = \mathbf{P} \cdot \lambda, \quad (11)$$

with  $\mathbf{P}$ , the matrix containing the eigenvectors  $\vec{v}_i$  in its columns and  $\lambda = \text{diag}(\lambda_i)$ :  $\mathbf{P} = (\vec{v}_1 \dots \vec{v}_n)_{n \times m}$

$$v_{1,2}v_{2,2}\cdots v_{n,2}$$

$$\vdots\cdots\vdots$$

$$v_{1,m}v_{2,m}\cdots v_{n,m} \text{ and } \lambda = (\lambda)_1 0\cdots 0$$

$$0\lambda_2\cdots 0$$

$$\vdots\cdots\vdots$$

$00\cdots\lambda_n$ . Thus, the transformation from  $m$  inputs to  $n$  latent features is accomplished by the transformation  $e(\vec{x}) = \mathbf{P}^T \vec{x}$ , and the reverse transformation into input space via  $d(e(\vec{x})) = \mathbf{P}\mathbf{P}^T \vec{x}$ . When  $n < m$ , then  $\mathbf{C} \neq \mathbf{P}\lambda\mathbf{P}^T$  and  $d(e(\vec{x}))$  cannot preserve all information anymore:  $d(e(\vec{x})) \neq \vec{x}$ . It is noteworthy to mention that for PCA to succeed principal features must exist in the data to begin with, i.e. correlation in the features of the input data is presumed.

Both PCA and the linear undercomplete autoencoder are looking for the same linear subspace. The only difference is that the basis vectors of autoencoders are not necessarily orthogonal, while the eigenvectors of the (symmetric) covariance matrix for PCA are by principle always orthogonal. [?] Furthermore, PCA requires the eigenvectors to be normalized, thus yielding a unique solution while for autoencoder training there exists no such precondition.

### 3 Neural network model architectures

### 4 Neural network training progression

### 5 16-Split weight distributions

Table 1: **CAE**: Baseline convolutional autoencoder

Layer/Operation	Kernel	Features	Stride	Padding	Activation
Input shape $28 \times 28 \times 1$					
Convolution 2D	$3 \times 3$	32 maps	$2 \times 2$	same	ReLU
Convolution 2D	$3 \times 3$	64 maps	$2 \times 2$	same	ReLU
Flatten to $16 \times 1$					
Densely connected	N/A	16 nodes	N/A	N/A	Sigmoid
Densely connected	N/A	392 nodes	N/A	N/A	ReLU
Reshape to $7 \times 7 \times 8$					
Transposed convolution 2D	$3 \times 3$	64 maps	$2 \times 2$	same	ReLU
Transposed convolution 2D	$3 \times 3$	32 maps	$2 \times 2$	same	ReLU
Transposed convolution 2D	$3 \times 3$	1 map	$1 \times 1$	same	Linear
Output shape $28 \times 28 \times 1$					

Table 2: **FCAE**: Fully convolutional autoencoder

Layer/Operation	Kernel	Features	Stride	Padding	Activation
Input shape $28 \times 28 \times 1$					
Convolution 2D	$3 \times 3$	32 maps	$2 \times 2$	same	ReLU
Convolution 2D	$3 \times 3$	64 maps	$2 \times 2$	same	ReLU
Convolution 2D	$4 \times 4$	1 map	$1 \times 1$	valid	Sigmoid
Flatten to $16 \times 1$					
Reshape to $4 \times 4 \times 1$					
Transposed convolution 2D	$4 \times 4$	1 maps	$1 \times 1$	valid	ReLU
Transposed convolution 2D	$3 \times 3$	64 maps	$2 \times 2$	same	ReLU
Transposed convolution 2D	$3 \times 3$	32 maps	$2 \times 2$	same	ReLU
Transposed convolution 2D	$3 \times 3$	1 map	$1 \times 1$	same	Linear
Output shape $28 \times 28 \times 1$					

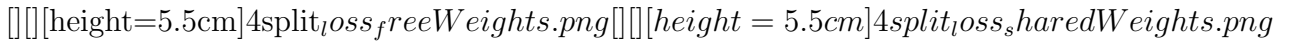


Figure 3: 4-split reconstruction loss for (a) free weights (b) shared weights trained over 50 epochs.

Table 3: **4S**: 4-Split

Layer/Operation	Kernel	Features	Stride	Padding	Activation
Input shape $4 \times 14 \times 14 \times 1$					
Convolution 2D	$3 \times 3$	32 maps	$2 \times 2$	same	ReLU
Convolution 2D	$3 \times 3$	64 maps	$2 \times 2$	same	ReLU
Flatten to $4 \times 1$					
Densely connected	N/A	4 nodes	N/A	N/A	Sigmoid
Concatenate from $4 \times 4$ to $16 \times 1$					
Densely connected	N/A	392 nodes	N/A	N/A	ReLU
Reshape to $7 \times 7 \times 8$					
Transposed convolution 2D	$3 \times 3$	64 maps	$2 \times 2$	same	ReLU
Transposed convolution 2D	$3 \times 3$	32 maps	$2 \times 2$	same	ReLU
Transposed convolution 2D	$3 \times 3$	1 map	$2 \times 2$	same	Linear
Output shape $28 \times 28 \times 1$					

Table 4: **F4S**: Fully convolutional 4-Split

Layer/Operation	Kernel	Features	Stride	Padding	Activation
Input shape $4 \times 14 \times 14 \times 1$					
Convolution 2D	$3 \times 3$	32 maps	$2 \times 2$	same	ReLU
Convolution 2D	$3 \times 3$	64 maps	$2 \times 2$	same	ReLU
Convolution 2D	$3 \times 3$	1 map	$1 \times 1$	valid	Sigmoid
Flatten to $4 \times 1$					
Concatenate from $4 \times 4$ to $16 \times 1$					
Reshape to $4 \times 4 \times 1$					
Transposed convolution 2D	$4 \times 4$	1 map	$1 \times 1$	valid	ReLU
Transposed convolution 2D	$3 \times 3$	64 maps	$2 \times 2$	same	ReLU
Transposed convolution 2D	$3 \times 3$	32 maps	$2 \times 2$	same	ReLU
Transposed convolution 2D	$3 \times 3$	1 map	$1 \times 1$	same	Linear
Output shape $28 \times 28 \times 1$					

Table 5: **16S**: 16-Split

Layer/Operation	Kernel	Features	Stride	Padding	Activation
Input shape $16 \times 14 \times 14 \times 1$					
Convolution 2D	$3 \times 3$	32 maps	$2 \times 2$	same	ReLU
Convolution 2D	$3 \times 3$	64 maps	$2 \times 2$	same	ReLU
Flatten to $4 \times 1$					
Densely connected	N/A	4 nodes	N/A	N/A	Linear
Batch Normalization					
Activation	N/A	N/A	N/A	N/A	Sigmoid
<i>Collect <math>16 \times 4</math> outputs after 16 encoder iterations.</i>					
Concatenate from $16 \times 4$ to $48 \times 1$ ( <i>16-Split encoder output</i> )					
Densely connected	N/A	1568 nodes	N/A	N/A	ReLU
Reshape to $14 \times 14 \times 8$					
Transposed convolution 2D	$3 \times 3$	64 maps	$2 \times 2$	same	ReLU
Transposed convolution 2D	$3 \times 3$	32 maps	$2 \times 2$	same	ReLU
Transposed convolution 2D	$3 \times 3$	1 map	$1 \times 1$	same	Linear
Output shape $56 \times 56 \times 1$					

Table 6: **16S-C**: 16-Split classifier head

Layer/Operation	Kernel	Features	Stride	Padding	Activation
Input shape $48 \times 1$ from <i>16-Split encoder output</i>					
Densely connected	N/A	256 nodes	N/A	N/A	ReLU
Densely connected	N/A	128 nodes	N/A	N/A	ReLU
Densely connected	N/A	64 nodes	N/A	N/A	ReLU
Densely connected	N/A	10 nodes	N/A	N/A	Softmax
Output shape $10 \times 1$					

Table 7: **16S-R**: 16-Split regressor head

Layer/Operation	Kernel	Features	Stride	Padding	Activation
Input shape $48 \times 1$ from <i>16-Split encoder output</i>					
Densely connected	N/A	256 nodes	N/A	N/A	ReLU
Densely connected	N/A	128 nodes	N/A	N/A	ReLU
Densely connected	N/A	64 nodes	N/A	N/A	ReLU
Densely connected	N/A	2 nodes	N/A	N/A	Linear
Output shape $2 \times 1$					

## Danksagung (Acknowledgement)

An dieser Stelle möchte ich mich bei allen bedanken, die zur Verwirklichung dieser Arbeit beigetragen haben.

Zunächst möchte ich mich bei Prof. Dr. Michael Wick für die Betreuung dieser Thesis bedanken. Sie standen mir stets ..

On the same note, I would like Dr. André David Tinoco Mendes for his excellent supervision during my time at CERN. Your guidance and support ..

Many thanks to ..

The following people gave valuable comments to various parts of this thesis draft:

Dr. André David Tinoco Mendes reviewed the entire thesis, pro..., Dr. Thorben Quast has reviewed chapters 1–3 and 5–6 and gave stimulating feedback on structure and scientific accuracy. Especially his comments to introduction, summary and the particulars on the theory of neural networks proofed to be invaluable. Dr. Arnaud Steen reviewed Ch. 4. His comments on technical details helped to clarify many aspects of the testing setup. Thanks to Dr. David Barney for reviewing XX of this thesis and providing constructive criticism. Your input helped to make the statements of this thesis more clear and accessible.

The chip testing efforts would not be possible without the collaboration of many people at CERN, DESY, FNAL, LLR, and university of Minnesota.