UNIVERSITÄT BONN

Introduction to Neural Networks

Moritz Wolter, Elena Trunz

February 24, 2025

Visual Computing Group, University of Bonn

Overview

Neural networks

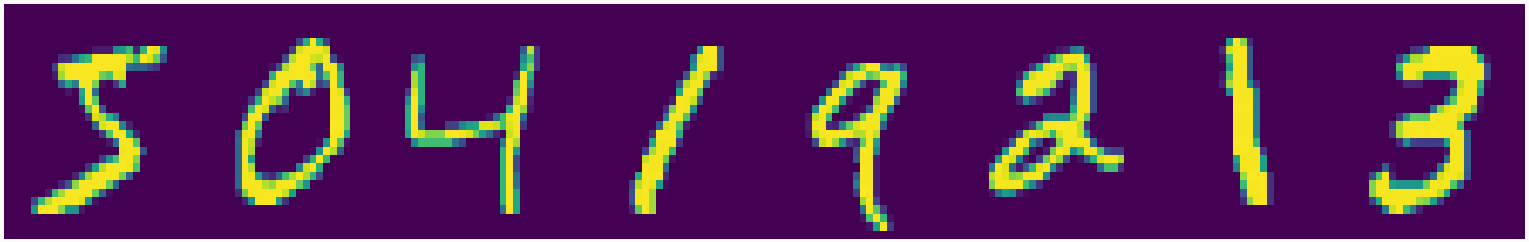
Training neural networks

Classification with neural networks

Data Augmentation

Neural networks

The wonders of the human visual system



Most humans effortlessly recognize the digits 5 0 4 1 9 2 1 3.

Biological motivation

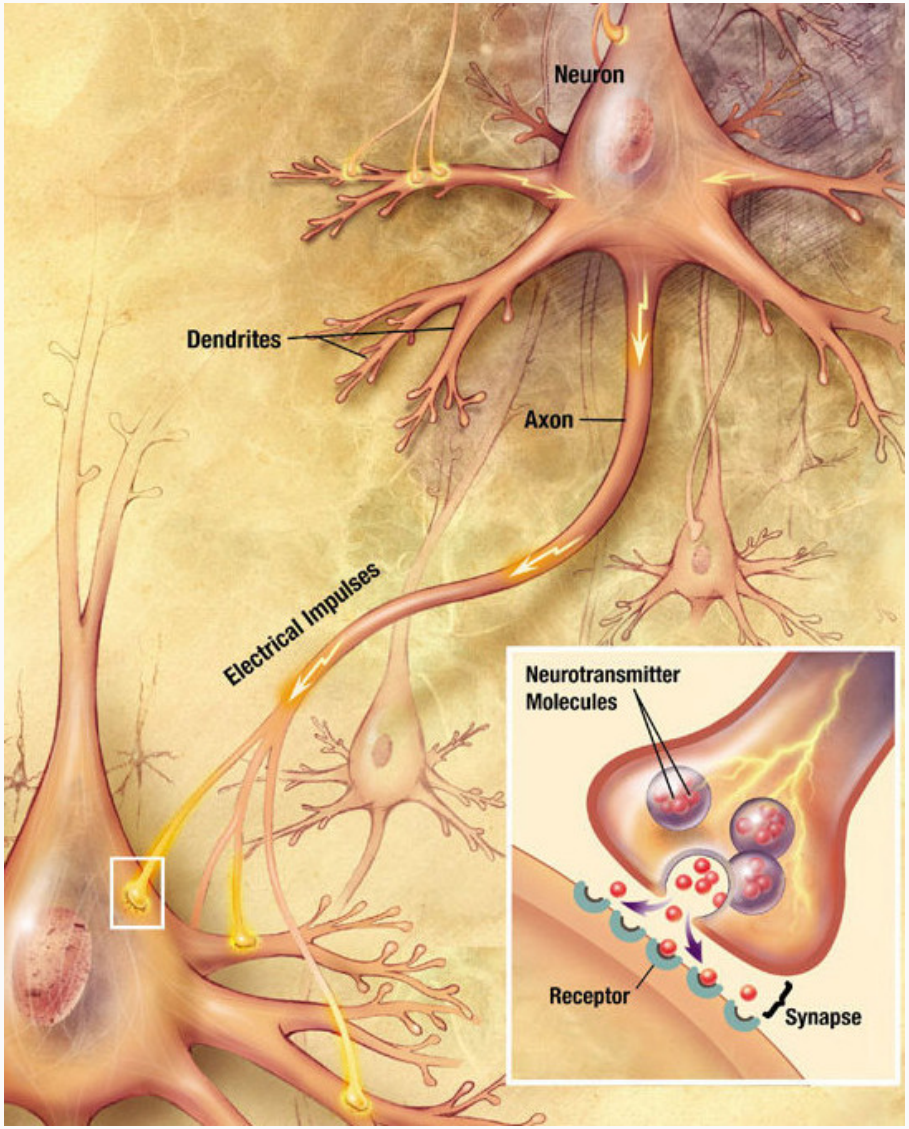
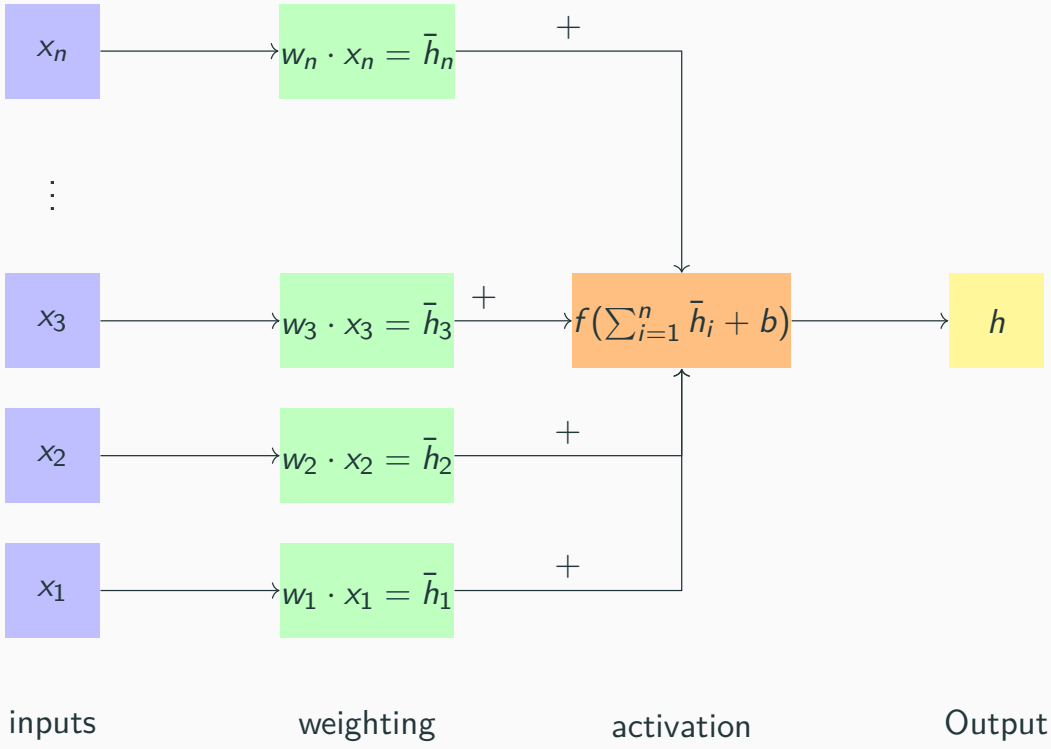


Image source: en.wikipedia.org

The perceptron

Can computers recognize digits? Mimic biological neurons,



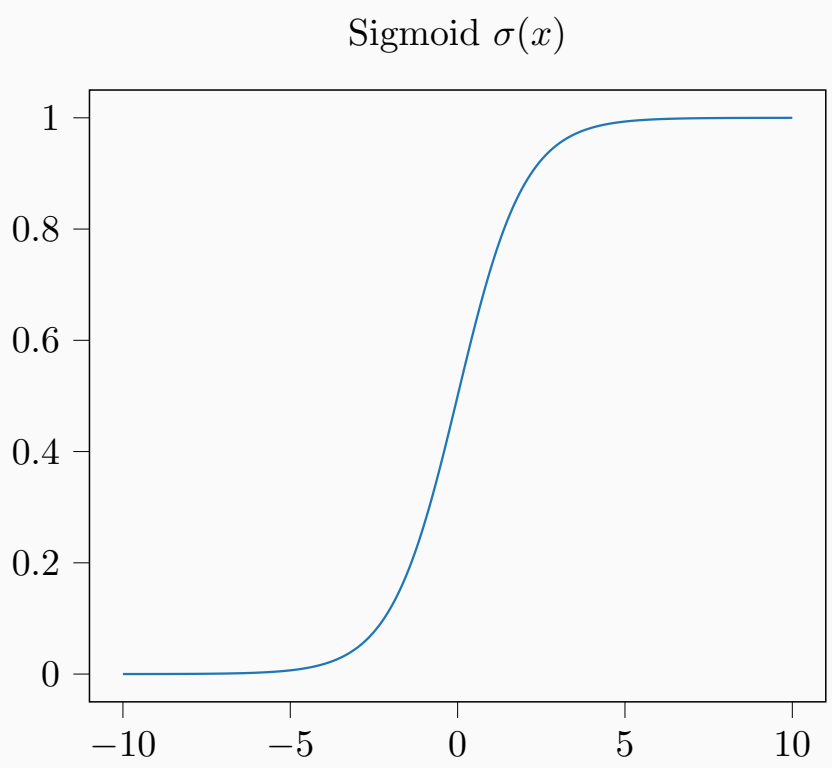
Formally a single perceptron is defined as

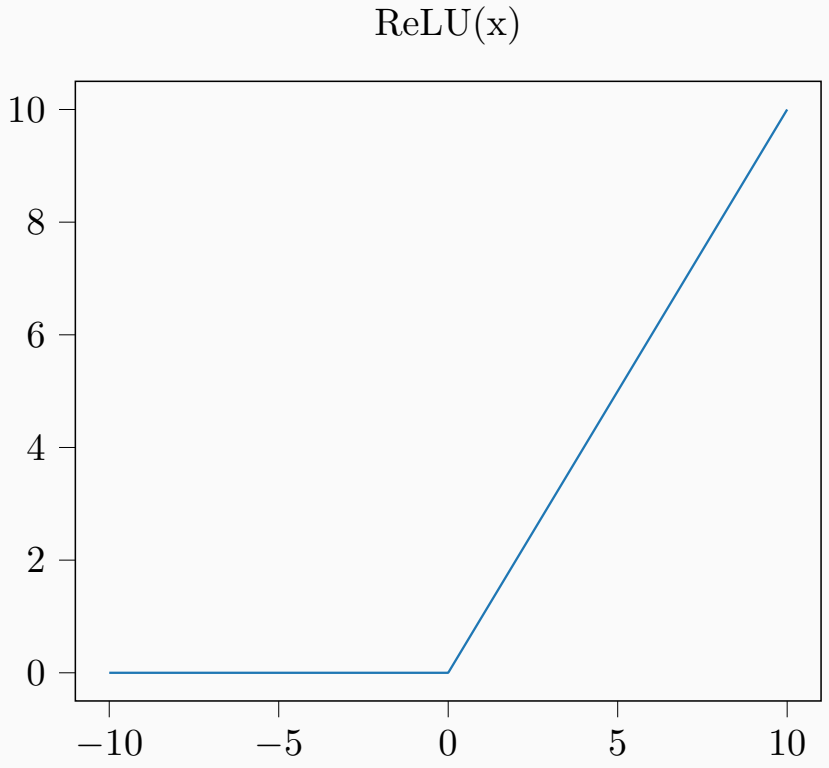
$$f(\mathbf{w}^T \mathbf{x} + b) = h \tag{1}$$

with $\mathbf{w} \in \mathbb{R}^n$, $\mathbf{x} \in \mathbb{R}^n$ and $h, b \in \mathbb{R}$.

The activation function f

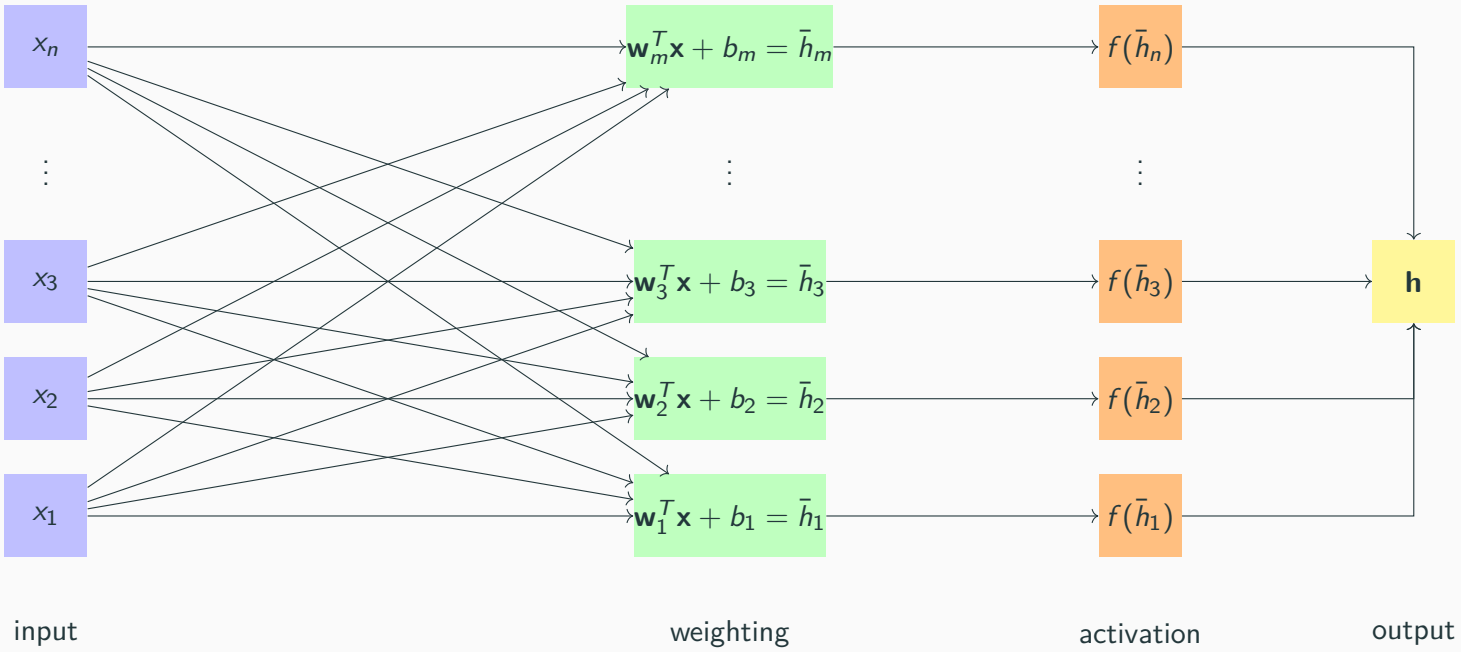
Activation functions should be differentiable and non-linear. Two popular choices for the activation function f are





Fully connected layer

Let's extend the definition to cover an array of perceptrons:



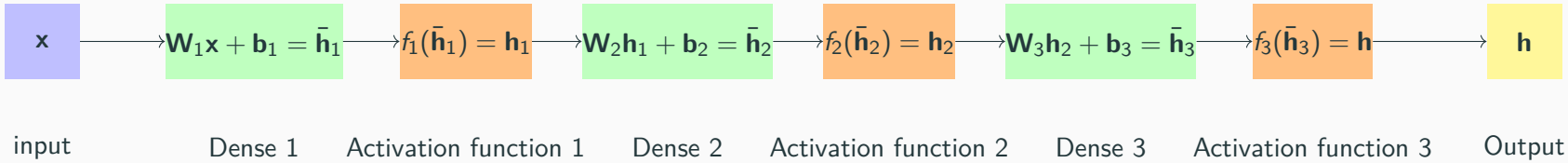
Every input is connected to every neuron. In matrix language, this turns into

$$\bar{\mathbf{h}} = \mathbf{W}\mathbf{x} + \mathbf{b}, \qquad \mathbf{h} = f(\bar{\mathbf{h}}). \tag{2}$$

With $\mathbf{W} \in \mathbb{R}^{m,n}$, $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{h}, \bar{\mathbf{h}} \in \mathbb{R}^m$.

Multi-layer networks

Stack dense (fully connected) layers and activations to create deeper networks.



The activation function used right before the output is typically chosen based on the nature of the task and often differs from those used in the hidden layers.

Output activations

- Regression: The activation function is simply an identity function, i.e. $h = \bar{h}$.
- Classification:
  - Logistic sigmoid function for binary classification:

$$\sigma_{\text{binary}}(\bar{h}) = \frac{1}{1 + \exp(-\bar{h})}$$

  - Soft-max activation for multiple classes:

$$\sigma_{\text{multi-class}}(\bar{h}) = \frac{\exp(\bar{h}_c)}{\sum_b \exp(\bar{h}_b)}$$

Training neural networks

The loss function

- To choose weights for the network, we require a quality measure.
- We already saw the mean squared error cost function:

$$C_{\text{mse}} = \frac{1}{2}\sum_{k=1}^{n}(\mathbf{h}_k - \mathbf{y}_k)^2 = \frac{1}{2}(\mathbf{h}-\mathbf{y})^T(\mathbf{h}-\mathbf{y}) \tag{3}$$

- This function measures the squared distance from each desired output. $\mathbf{y}$ denotes the desired labels, and $\mathbf{h}$ represents network output.
- We want to find the parameters $\mathbf{W}$ and $\mathbf{b}$ that minimize our loss.

Optimization

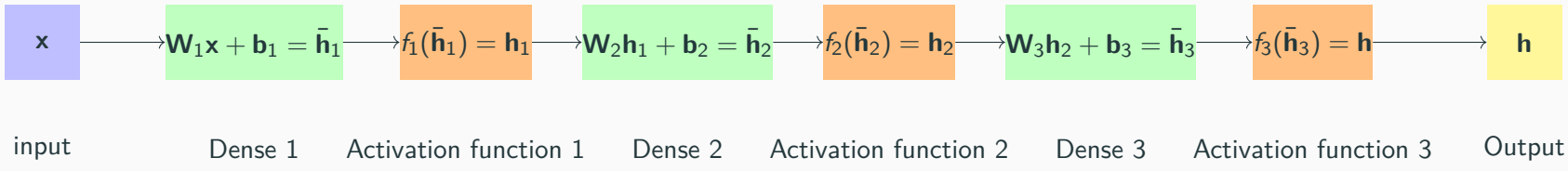
- We use gradient descent to minimize our loss function.
- That means we need to compute the gradients of the loss function with respect to each weight in the network.
- These gradients will tell us how much each weight should change to reduce the overall error.
- How can we compute the gradients efficiently?

  $\rightarrow$ We use *backpropagation*

Gradient descent with backpropagation

1. Perform a forward pass through the network.
2. Compute the gradient of the loss with respect to the output layer.
3. Backpropagate the error through the network.
4. Update the weights using the gradients.
5. Repeat steps 1) - 4) until convergence.

Step 1: Forward pass



Step 2: Gradient for output

The gradient of the mse-cost-function: Both the mean squared error loss function and our dense layer are differentiable.

$$\frac{\partial C_{\text{mse}}}{\partial \mathbf{h}} = \mathbf{h} - \mathbf{y} = \triangle_{\text{mse}} \tag{4}$$

The $\triangle$ symbol [Nie15] will re-appear. It always indicates incoming gradient information from above. If the labels are a vector of shape $\mathbb{R}^m$, $\triangle$ and the network output $\mathbf{h}$ must share this dimension.

Step 3: Backpropagate the gradient

Recall the chain rule $(g(h(x)))' = g'(h(x)) \cdot h'(x)$.

We use chain rule to backpropagate the gradient trough the activation function and the dense layer and repeat this for all layers down to the input.

Step 3: Backpropagate the gradient: Activation

From the forward pass we have: $\mathbf{h} = f(\bar{\mathbf{h}})$.

From backpropagation, we get an incoming gradient (error term) from the next layer:

$$\frac{\partial C}{\partial \mathbf{h}} = \triangle$$

We backpropagate it through the activation function:

$$\frac{\partial C}{\partial \bar{\mathbf{h}}} = \frac{\partial C}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \bar{\mathbf{h}}} = \triangle \odot f'(\bar{\mathbf{h}}),$$

where $\odot$ is the element-wise product.

Step 3: Backpropagate the gradient: Dense layer

$$\frac{\partial C}{\partial \mathbf{W}} = \frac{\partial C}{\partial \bar{\mathbf{h}}} \cdot \frac{\partial \bar{\mathbf{h}}}{\partial \mathbf{W}}, \qquad \frac{\partial C}{\partial \mathbf{b}} = \frac{\partial C}{\partial \bar{\mathbf{h}}} \cdot \frac{\partial \bar{\mathbf{h}}}{\partial \mathbf{b}}, \qquad \frac{\partial C}{\partial \mathbf{x}} = \frac{\partial C}{\partial \bar{\mathbf{h}}} \cdot \frac{\partial \bar{\mathbf{h}}}{\partial \mathbf{x}}$$

From the forward pass we know: $\bar{\mathbf{h}} = \mathbf{W}\mathbf{x} + \mathbf{b}$. Thus:

$$\frac{\partial \bar{\mathbf{h}}}{\partial \mathbf{W}} = \mathbf{x}^T, \qquad \frac{\partial \bar{\mathbf{h}}}{\partial \mathbf{b}} = 1, \qquad \frac{\partial \bar{\mathbf{h}}}{\partial \mathbf{x}} = \mathbf{W}$$

Substituting:

$$\frac{\partial C}{\partial \mathbf{W}} = (\triangle \odot f'(\bar{\mathbf{h}}))\mathbf{x}^T, \quad \frac{\partial C}{\partial \mathbf{b}} = \triangle \odot f'(\bar{\mathbf{h}}), \quad \frac{\partial C}{\partial \mathbf{x}} = \mathbf{W}^T(\triangle \odot f'(\bar{\mathbf{h}}))$$

The gradient of a dense layer

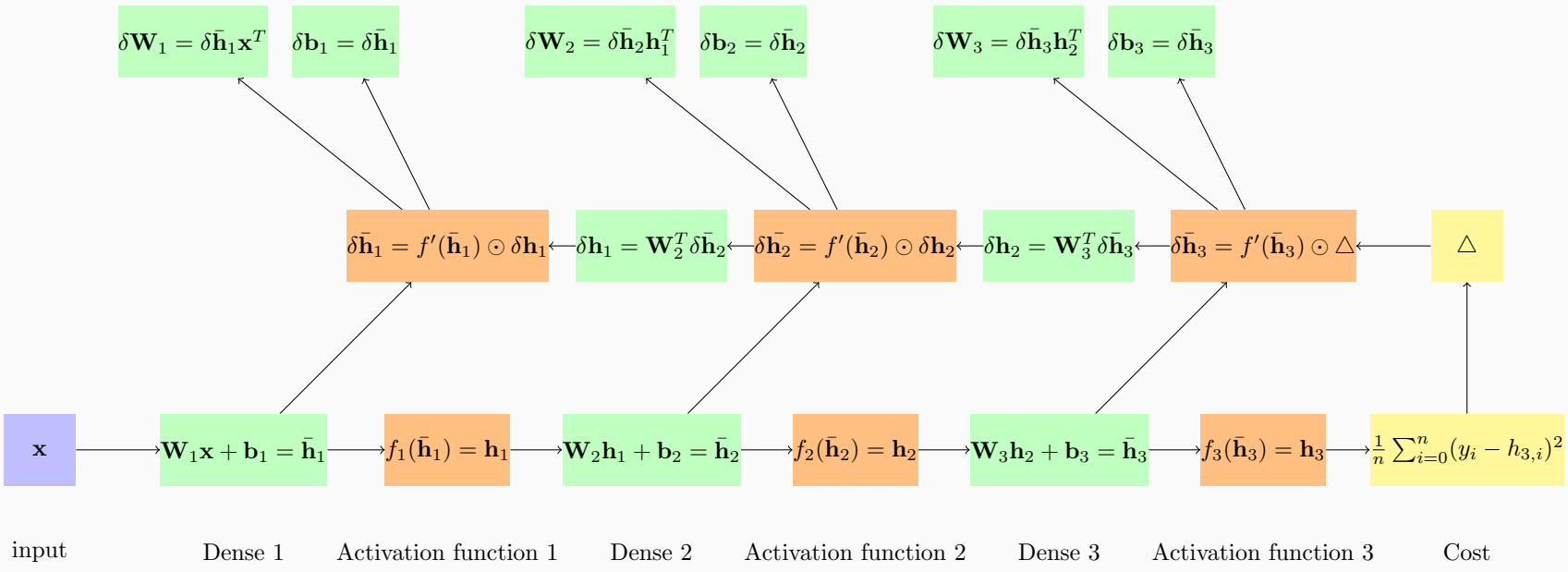
Let δ denote the cost function gradient for the value following it [Gre+16]. It holds:

$$\delta \mathbf{W} = [f'(\bar{\mathbf{h}}) \odot \triangle]\mathbf{x}^T, \qquad \delta \mathbf{b} = f'(\bar{\mathbf{h}}) \odot \triangle, \tag{5}$$

$$\delta \mathbf{x} = \mathbf{W}^T[f'(\bar{\mathbf{h}}) \odot \triangle], \tag{6}$$

with $\delta \mathbf{W} \in \mathbb{R}^{m,n}$, $\delta \mathbf{x} \in \mathbb{R}^n$ and $\delta \mathbf{b} \in \mathbb{R}^m$. **Modern libraries will take care of these computations for you!**

Backpropagation



Step 4: Updating the weights

The weights at the next time step $\tau + 1$ are given by,

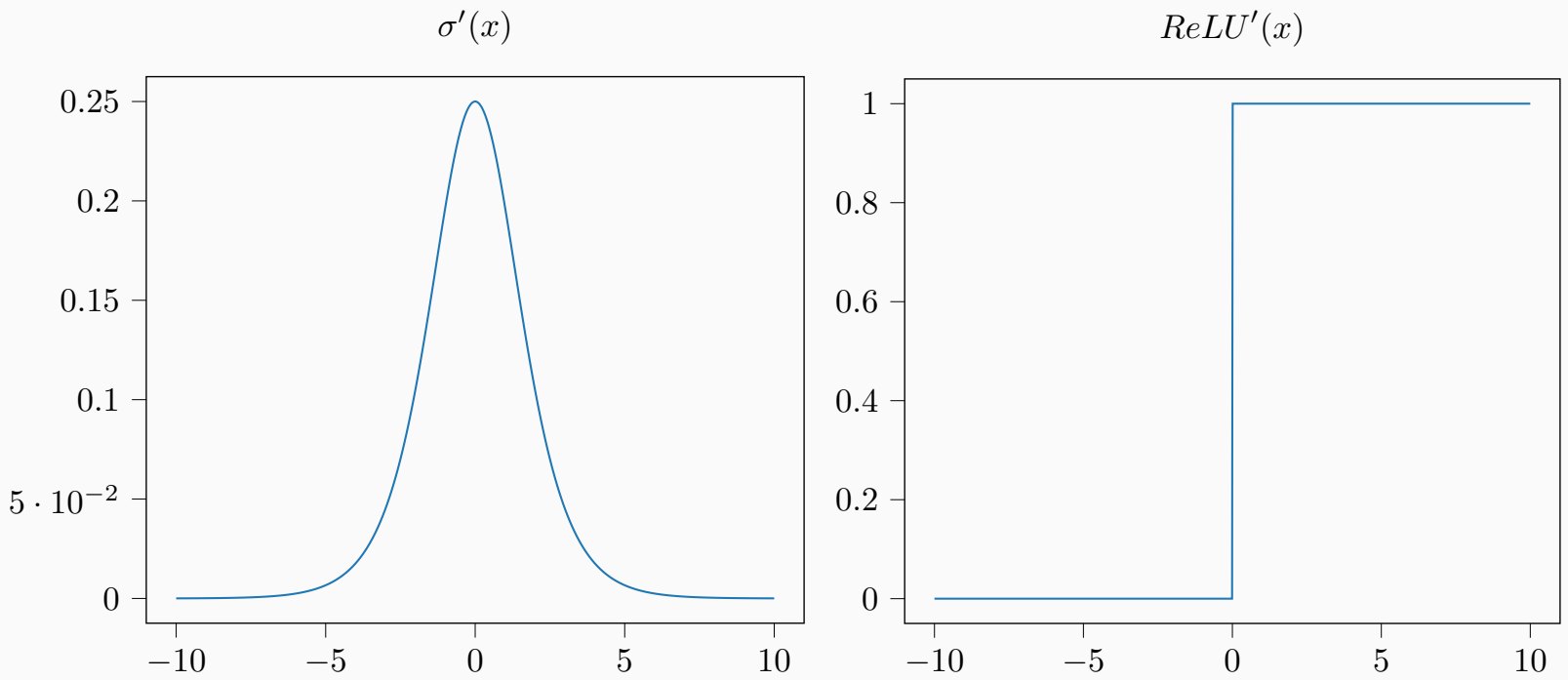
$$\mathbf{W}_{\tau+1} = \mathbf{W}_{\tau} - \epsilon \cdot \delta \mathbf{W}_{\tau},$$
$$\mathbf{b}_{\tau+1} = \mathbf{b}_{\tau} - \epsilon \cdot \delta \mathbf{b}_{\tau},$$

where the step size, also called the learning rate, is given by $\epsilon \in \mathbb{R}$. At $\tau = 0$, matrix entries are random.

Derivatives of our activation functions

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x)) \tag{7}$$

$$\mathrm{ReLU}'(x) = H(x) \tag{8}$$



Perceptrons for functions

The network components described this far already allow function learning. Given a noisy input signal $\mathbf{x} \in \mathbb{R}^n$ and a ground through output $\mathbf{y} \in \mathbb{R}^n$, define,

$$\mathbf{h}_1 = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{9}$$

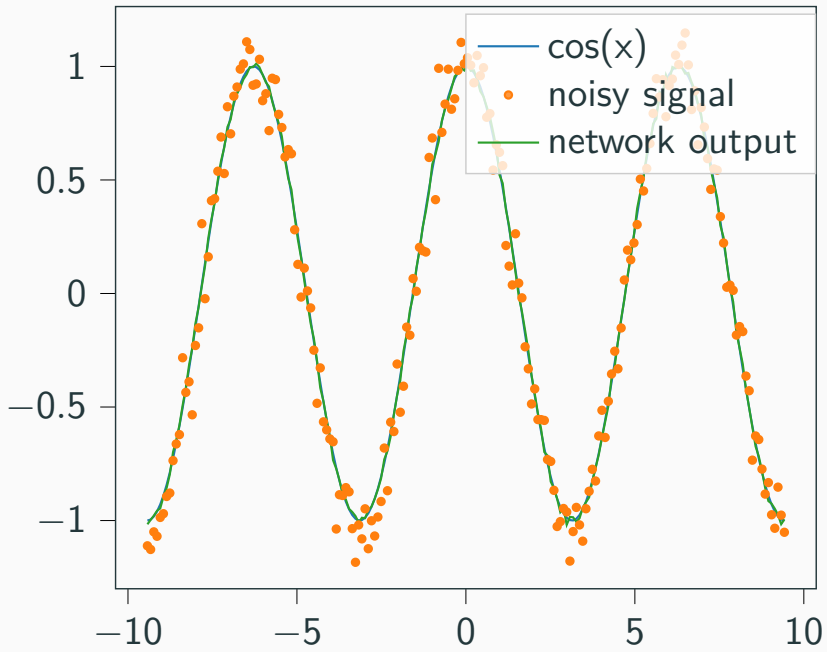
$$\mathbf{o} = \mathbf{W}_{\text{proj}}\mathbf{h}_1 \tag{10}$$

With $\mathbf{W} \in \mathbb{R}^{m,n}$ and $\mathbf{b} \in \mathbb{R}^m$. m and n denote the number of neurons and the input signal length. For signal denoising, input and output have the same length. Therefore $\mathbf{W}_{\text{proj}} \in \mathbb{R}^{n,m}$. $\mathbf{o} \in \mathbb{R}^n$ denotes the network output.

Denoising a cosine

Here, the random initialization of the weights from $\mathcal{U}[-0.1, 0.1]$ is a reasonable choice.

Optimization for 500 steps with 10 neurons, leads to the output below:



The cosine function is shown in blue, a noisy network input in orange, and a denoised network output in green.

Summary

- Artificial neural networks are biologically motivated.
- Gradients make it possible to optimize arrays of neurons.
- A single array of layers of neurons can solve tasks like denoising a cosine.

Classification with neural networks

The cross-entropy loss

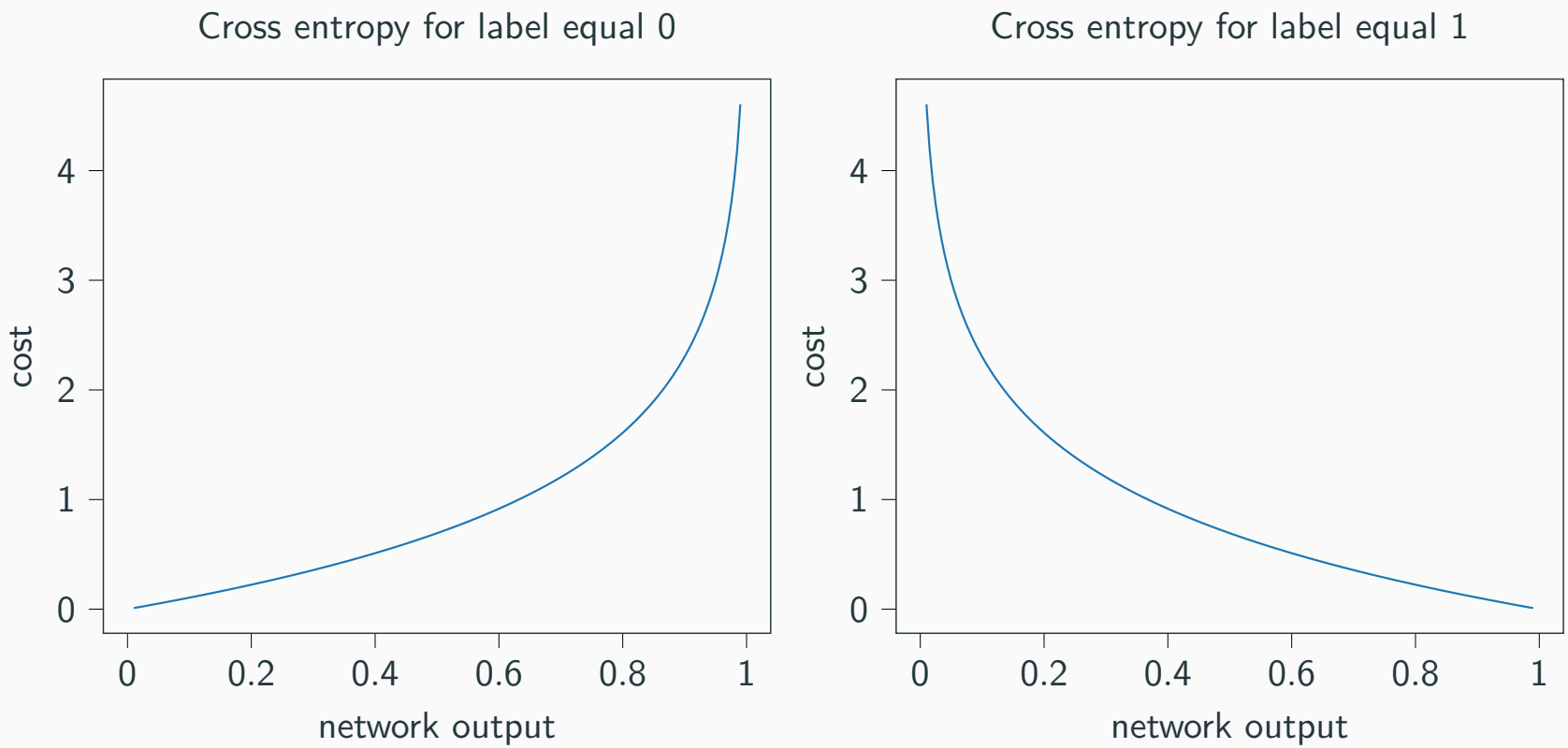
The cross-entropy loss function is defined as [Nie15; Bis06]

$$C_{\text{ce}}(\mathbf{y}, \mathbf{o}) = -\sum_{k}^{n_o}[(\mathbf{y}_k \ln \mathbf{o}_k) + (\mathbf{1} - \mathbf{y}_k) \ln(\mathbf{1} - \mathbf{o}_k)]. \tag{11}$$

With n_o the number of output neurons. $\mathbf{y} \in \mathbb{R}^{n_o}$ the desired output and $\mathbf{o} \in \mathbb{R}^{n_o}$ the network output.

Understanding how cross-entropy works

To understand cross entropy let's consider the boundary cases $y = 0$ and $y = 1$.

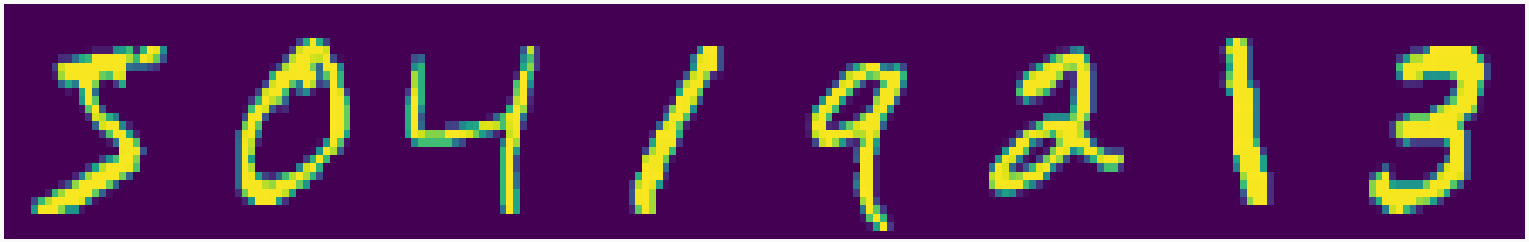


Gradients and cross-entropy

If a sigmoidal activation function produced $\mathbf{o}$, the gradients can be computed using [Nie15; Bis06]

$$\frac{\partial C_{ce}}{\partial \mathbf{h}} = \sigma(\mathbf{o}) - \mathbf{y} = \triangle_{\text{ce}} \tag{12}$$

The MNIST-Dataset



The MNIST-dataset contains 70k images of handwritten digits.

Validation and Test data splits

- To ensure the correct operation of the systems we devise, it is paramount to hold back part of the data for validation and testing.
- Before starting to train, split off validation and test data.
- The 70k MNIST samples could, for example, be partitioned into 59k training images. 1k validation images and 10k test images.

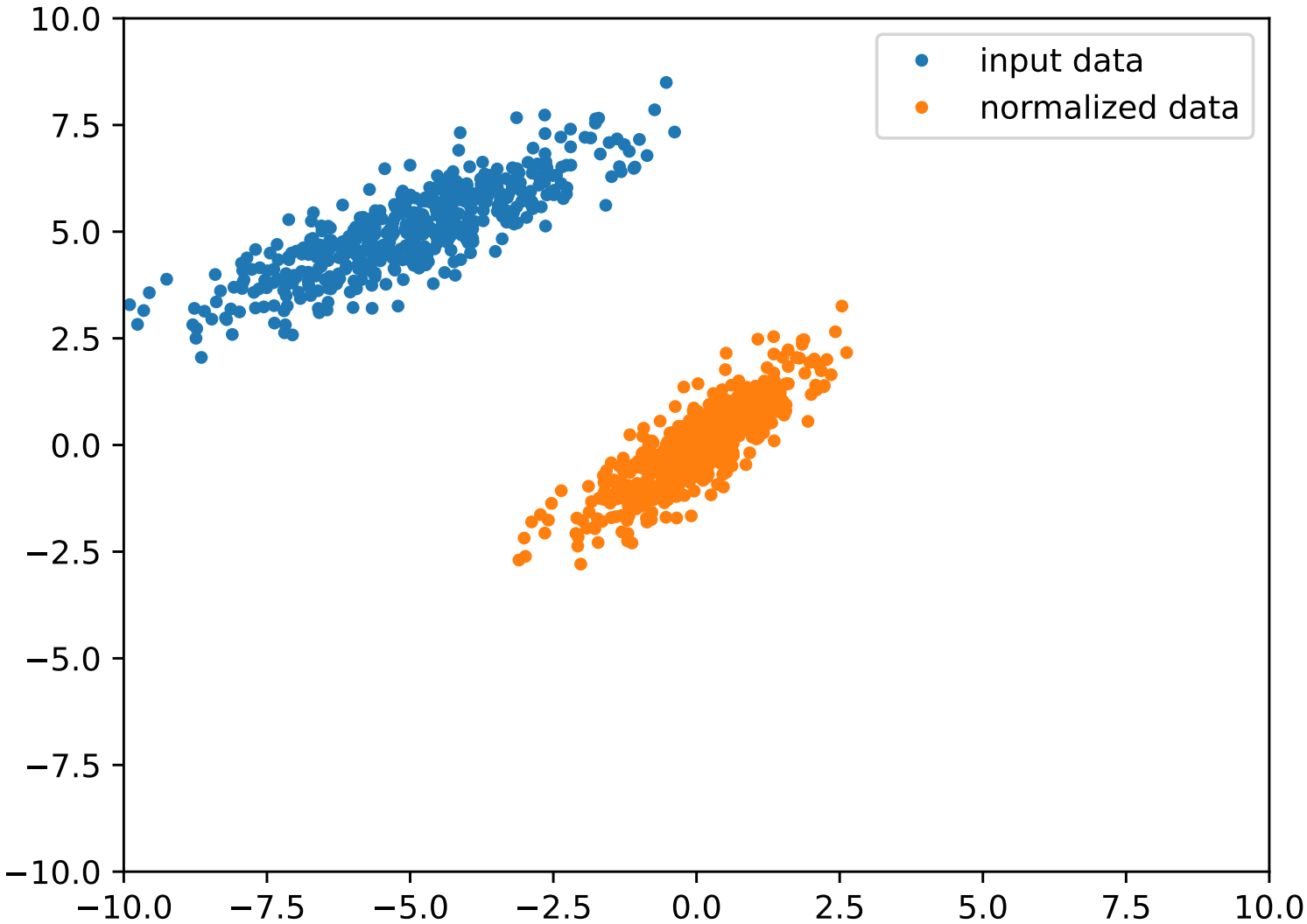
Input-preprocessing

Standard initializations and learning algorithms assume an approximately standard normal distribution of the network inputs. Consequently, we must rescale the data using,

$$x_{ij} = \frac{x_{ij} - \mu}{\sigma} \tag{13}$$

With μ and σ the training set mean and standard deviation. For all pixels, i, j up the height and width of every image.

The effect of normalization



Whitening the inputs [23]

Instead of dividing by the standard deviation, rescale the centered data with the singular values of the covariance matrix.

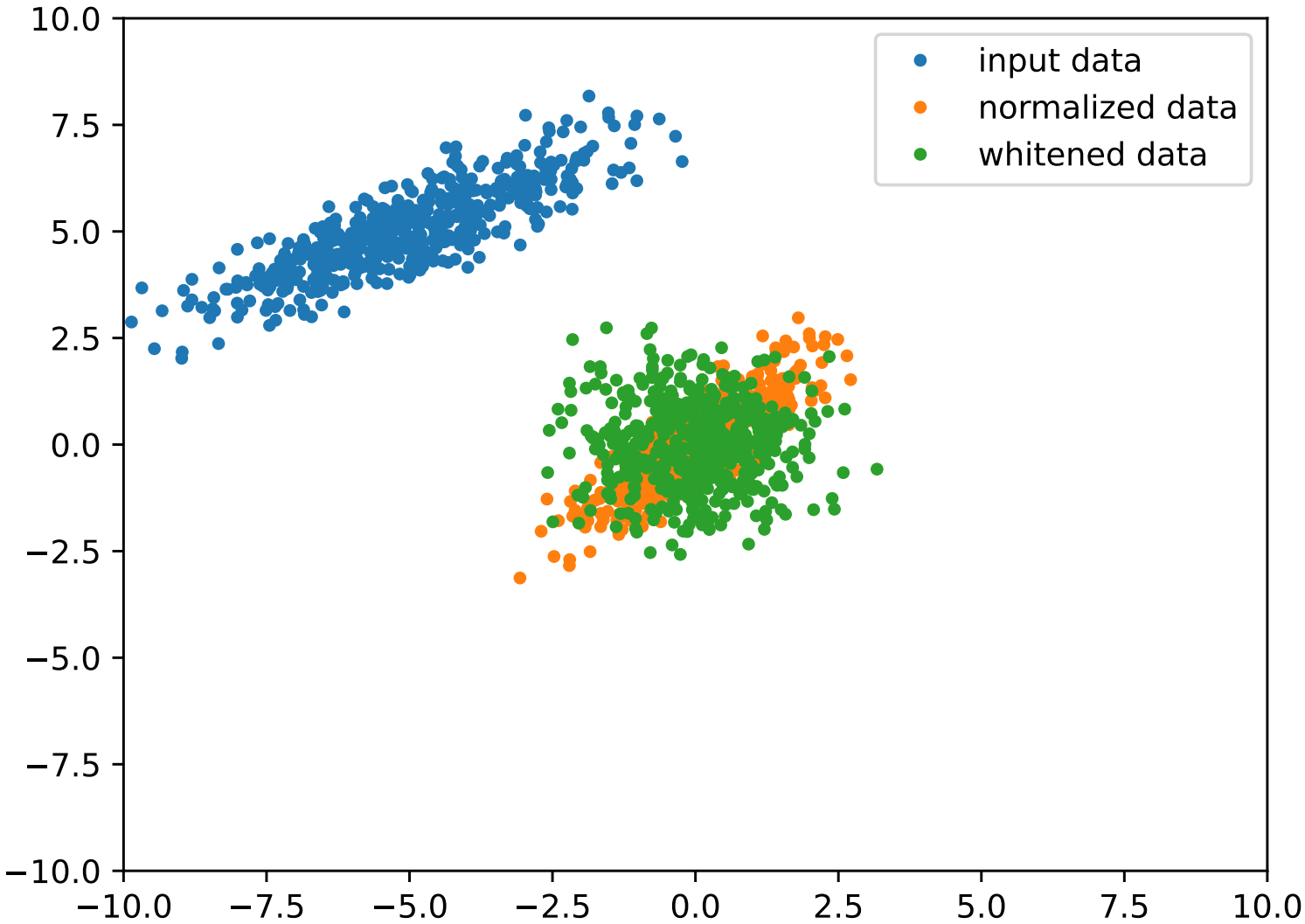
$$\mathbf{C} = \frac{1}{n}(\mathbf{x} - \mu)^T(\mathbf{x} - \mu) \tag{14}$$

With n as the total number of data points. Next we find $\mathbf{U\Sigma V} = \mathbf{C}$. After projecting the data via $\mathbf{p} = \mathbf{xU}$. Whitening now uses the singular values of $\mathbf{C}$ to rescale the data,

$$p_{ij} = \frac{p_{ij}}{\sqrt{\sigma_j} + \epsilon} \tag{15}$$

With ϵ i.e. equal to $1e^{-8}$ for numerical stability. The operation is repeated for all pixel locations i, j in the input image.

The effect of Whitening



Label-encoding

It has proven useful to have individual output neurons produce probabilities for each class. Given integer labels $1, 2, 3, 4, \cdots \in \mathbb{Z}$. One-hot encoded label vectors have a one at the label's position and zeros elsewhere. I.e.

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ \vdots \end{pmatrix}, \ldots \tag{16}$$

for the integer label sequence above.

Batching the data

Working with individual data points is not efficient in practice. Instead, we would like to process multiple (i.e. 64) samples in parallel. Add a leading batch dimension and rely on broadcasting.

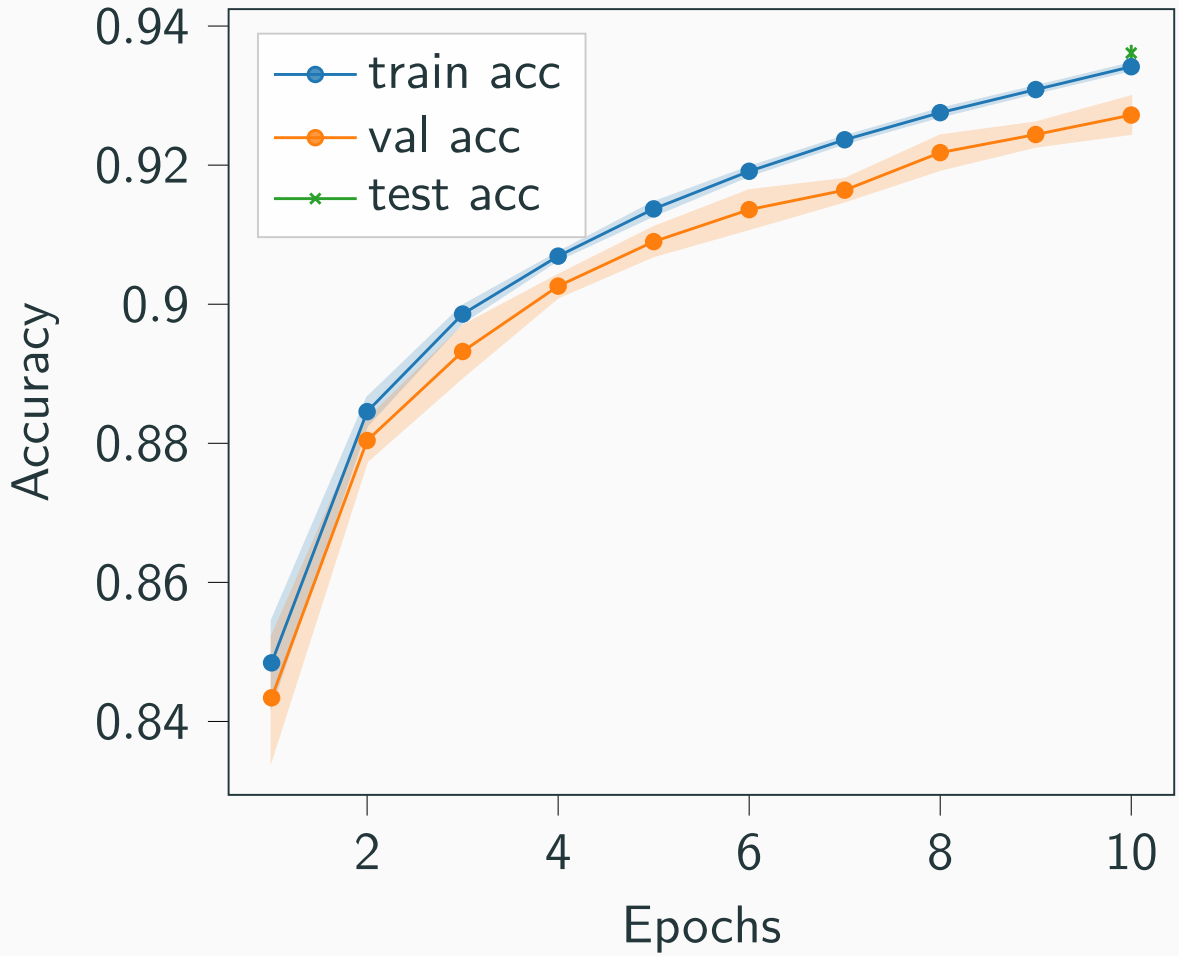
An additional mean converts the cost of a batch into a scalar. In the cross-entropy case:

$$C_{\text{ce}}(\mathbf{y}, \mathbf{o}) = -\frac{1}{n_b} \sum_{i=1}^{n_b} \sum_{k=1}^{n_o} [(\mathbf{y}_{i,k} \ln \mathbf{o}_{i,k}) + (\mathbf{1} - \mathbf{y}_{i,k}) \ln(\mathbf{1} - \mathbf{o}_{i,k})] \tag{17}$$

With n_o the number of output neurons and n_b the batch size.

MNIST-Classification

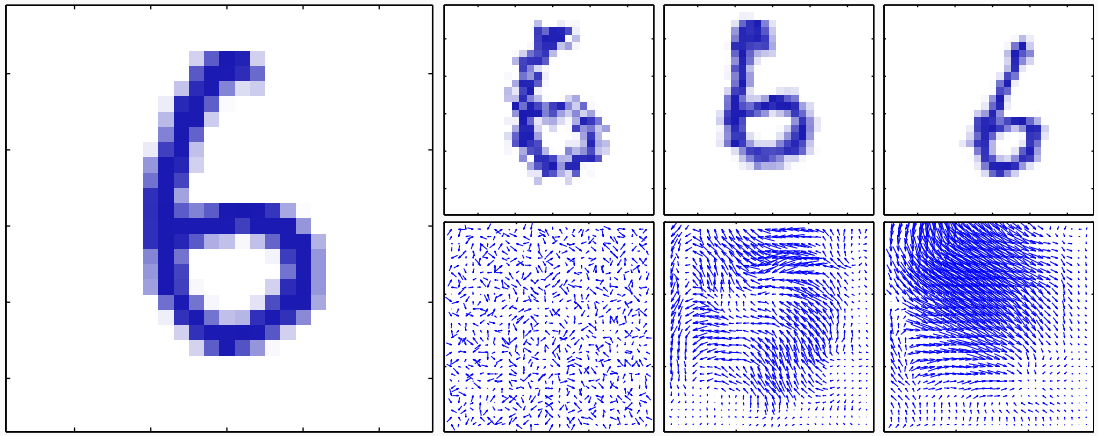
Training a three-layer dense network on mnist for five runs leads to:



Data Augmentation

Generating more training data

- A simple way to make a neural network invariant to certain transformations is to train it on data that includes these variations.
- However, collecting such data can be difficult.
- A common solution is to generate new data by applying transformations to the existing training set.



Synthetic warps of a digit generated by sampling random displacement fields (from [Bis06]).

Not all data augmentation works

The effectiveness of data synthesis depends on the application.

- In density estimation, generating new samples requires an already accurate model of the data distribution.
- For visual recognition tasks, data augmentation is effective because many image variations can be simulated.
  - Translating training images by a few pixels often improves generalization.
  - Other common transformations include rotation and scaling.
  - Care must be taken to avoid transformations that alter the true class.

Injecting noise

- In many classification and regression tasks, the model should perform well even with small random noise in the input.
- Neural networks are often sensitive to noise unless trained to handle it.
- A simple way to improve robustness is to train with noise-injected inputs.

Conclusion

- Preprocessing followed by forward passes, backward passes, and testing from the classic training pipeline.
- Using the pipeline, artificial neural networks enable computers to make sense of images.
- The optimization result depends on the initialization.
- The initialization depends on the seed of the pseudorandom number generator.
- *Seed-values must be recorded*, to allow reproduction.
- Share the results of multiple re-initialized runs, if possible.
- For visual recognition problems, data augmentation often improves performance of the models.

References

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[23] *Data-preprocessing*. https://cs231n.github.io/neural-networks-2/. Accessed: 2023-03-26. 2023.

Literature ii

[Gre+16] Klaus Greff, Rupesh K Srivastava, Jan Koutnik, Bas R Steunebrink, and Jürgen Schmidhuber. "LSTM: A search space odyssey." In: *IEEE transactions on neural networks and learning systems* 28.10 (2016), pp. 2222–2232.

[Nie15] Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA, 2015.