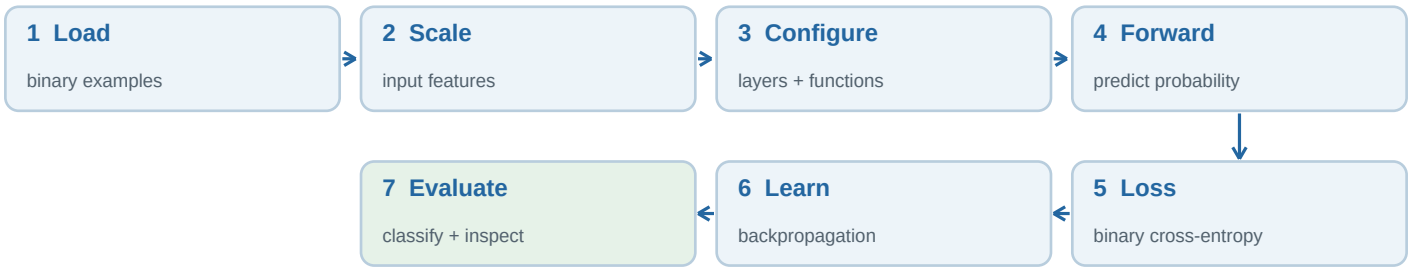
How Neural Network Training Works in AI-ANNE

From labeled examples to binary predictions | A configurable, trainable feedforward network

Prof. Dr. habil. Dennis Klinkhammer



Training repeats forward and backward passes for the selected number of epochs; evaluation uses the learned parameters.

1 Load labeled examples

$$(x^{(i)}, y^{(i)}), \quad x^{(i)} \in \mathbb{R}^m, \quad y^{(i)} \in \{0, 1\}$$

Upload numeric CSV data or select a demo. Column 1 holds the binary target; the remaining columns are input features.

2 Normalize each input feature

$$\widetilde{x}_j = \frac{x_j - \min(x_j)}{\max(x_j) - \min(x_j)}$$

Min-max scaling uses training-data extrema; constant features use a unit denominator.

3 Choose the architecture and activation functions

$$z^{(\ell)} = W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}, \quad a^{(\ell)} = f_\ell(z^{(\ell)})$$

Select 1–5 hidden layers (1–12 neurons each), their activations, training epochs and learning rate.

4 Apply nonlinear activation functions

$$\mathrm{ReLU}(z) = \max(0, z), \quad \mathrm{LeakyReLU}(z) = \max(0.01z, z),$$
$$\sigma(z) = \frac{1}{1 + e^{-z}}, \qquad \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

Hidden layers may use ReLU, Leaky ReLU, sigmoid or tanh; the single output neuron always uses sigmoid.

5 Produce a binary prediction and measure loss

$$\ell_i = -[y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$
$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \ell_i, \qquad p_i = \sigma(z_i^{(L)})$$

The sigmoid output estimates class-1 probability. Binary cross-entropy penalizes incorrect probabilities.

6 Backpropagate errors and update parameters

$$\delta^{(L)} = p - y, \quad W \leftarrow W - \eta \nabla_W \mathcal{L}, \quad b \leftarrow b - \eta \nabla_b \mathcal{L}$$

Per-example gradient updates adjust weights and biases; multiple epochs repeat training over the examples.

7 Classify new cases and inspect the network

$$\widehat{y} = \mathbf{1}[p \geq 0.5], \quad \mathrm{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Inspect the confusion matrix, accuracy and final loss; enter new features to trace activations and local feature influence.

Why it runs on modest hardware

A **small**, fully connected network trains locally in the browser with Python (Pyodide). Its learned parameters can be exported as standalone **MicroPython** code for inference on compatible devices. No GPU, external training service or large pretrained model is required.



Implementation note. AI-ANNE uses per-example gradient descent, a sigmoid output and binary cross-entropy. The displayed local feature influence compares a prediction with one feature replaced by its training-set mean; the highlighted path is illustrative, not a causal explanation.