Neural Networks

Classification and regression

André Meichtry
Statistik-Kolloquium für Interessierte, BFH Gesundheit

2025-04-02

Neural Networks

- Very popular in the 90's in the ML and AI communities.
- Can be viewed as high-dimensional nonlinear regression models.
- Focus here on feedforward neural nets (versus recurrent neural network).

Aim

- Aim: Estimate a multivariate function that maps
  - from an p-dimensional **input space**
  - to an o-dimensional **output space**.
- Mathematically, this can be expressed as:

$$\mathbf{f}(\cdot) : \mathbb{R}^p \to \mathbb{R}^o$$

- **Classification**: For a multi-class classification problem with o classes, each dimension represents the probability of a class.
- **Regression**: For a regression problem predicting a single continuous value, $o = 1$

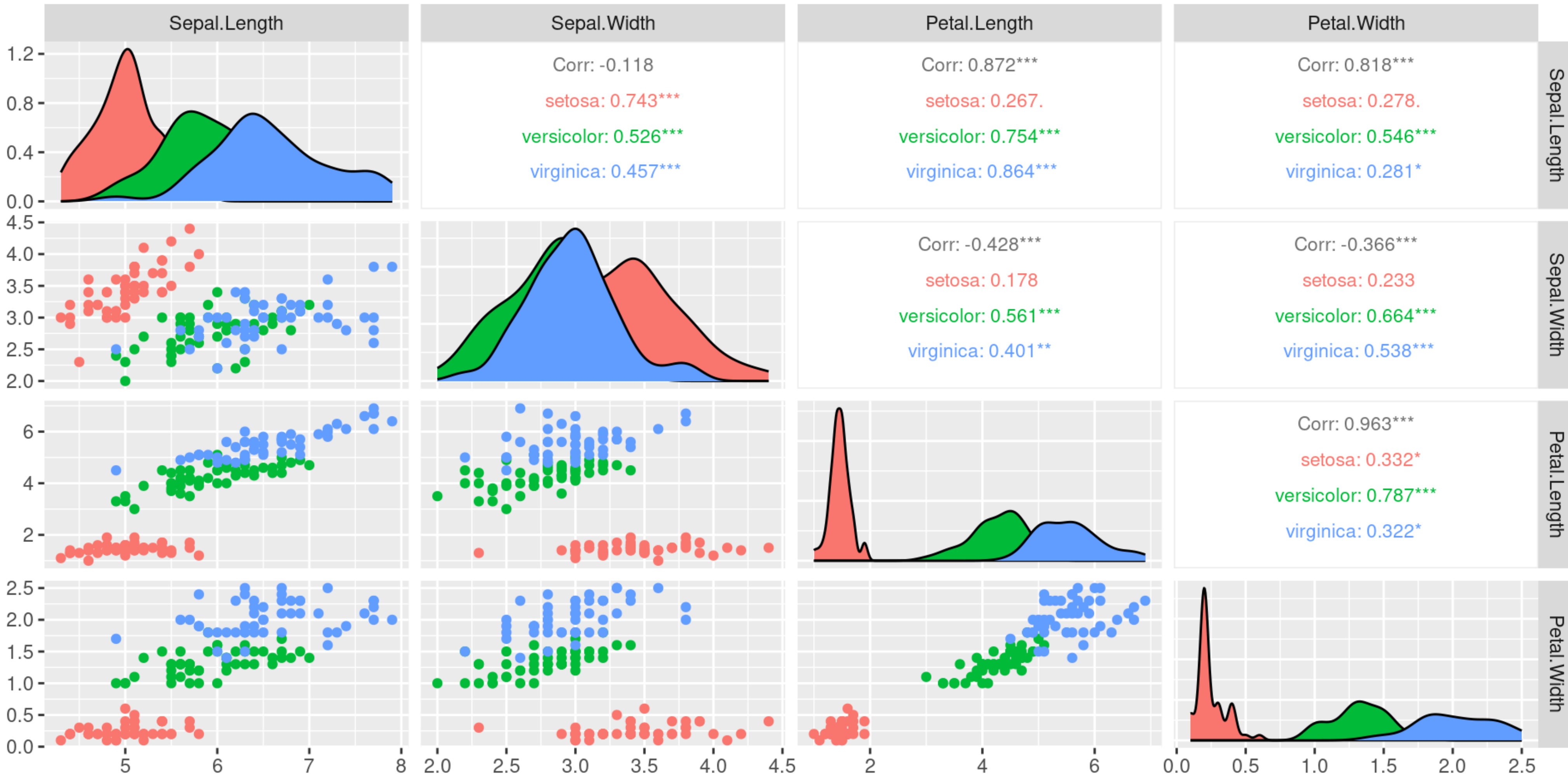
Training data

```r
psych::headTail(iris)
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1            5.1         3.5          1.4         0.2    setosa
2            4.9           3          1.4         0.2    setosa
3            4.7         3.2          1.3         0.2    setosa
4            4.6         3.1          1.5         0.2    setosa
...          ...         ...          ...         ...      <NA>
147          6.3         2.5            5         1.9 virginica
148          6.5           3          5.2           2 virginica
149          6.2         3.4          5.4         2.3 virginica
150          5.9           3          5.1         1.8 virginica
```

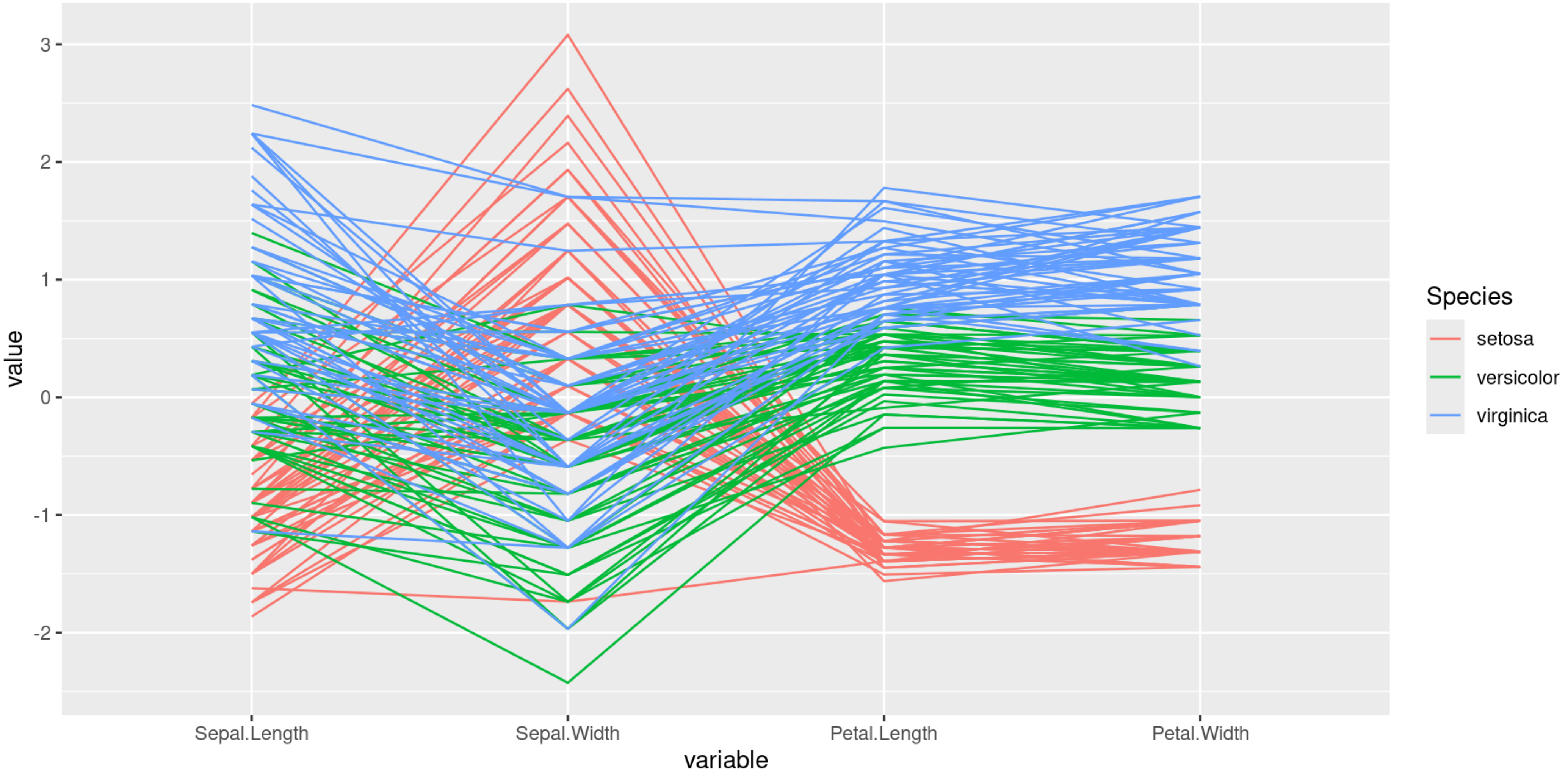
Training data

```r
library(ggplot2)
library(GGally)
ggpairs(iris, columns = 1:4, mapping = ggplot2::aes(colour = Species), upper = list(continuous = wrap("cor",
```



Training data

```r
ggparcoord(iris, 1:4, groupColumn = 5)
```

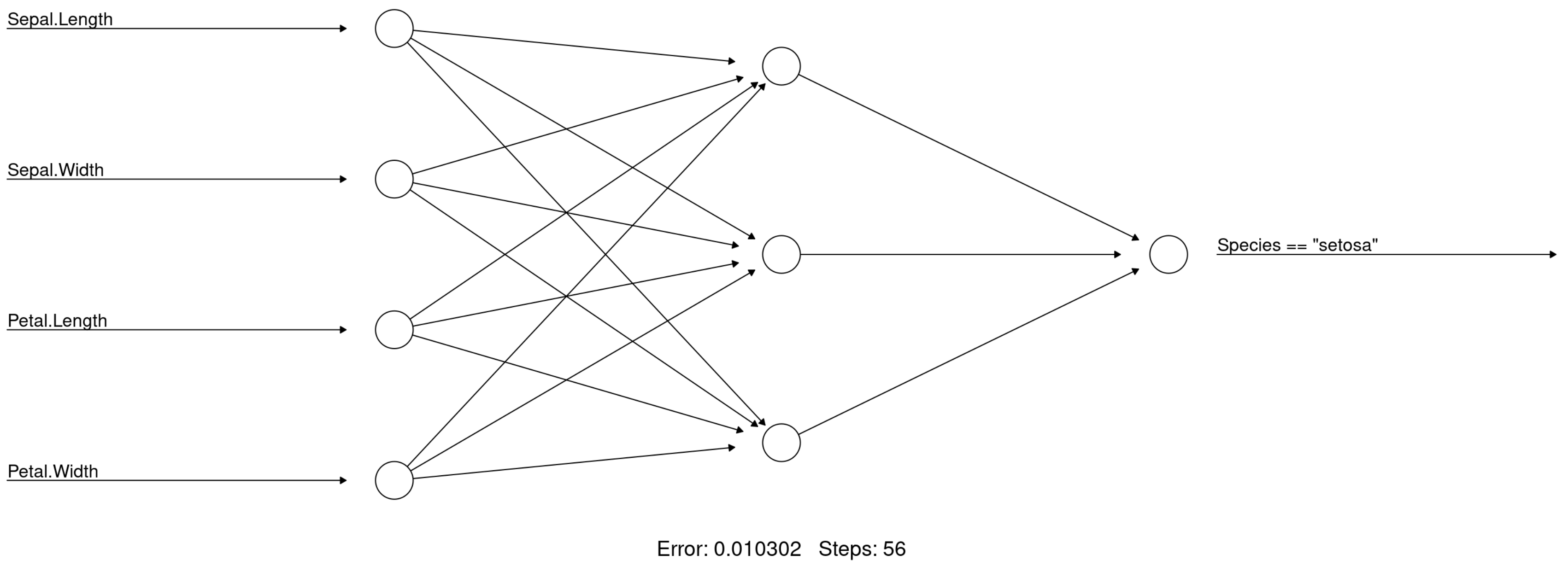


Terminology

- Input layer $\mathbf{x}$:
  - p inputs x_i, $i = 1, \ldots, p$
- Hidden layer $\mathbf{h}$:
  - q hidden neurons h_j, $j = 1, \ldots, q$
- Output layer $\mathbf{y}$:
  - o outputs y_k, $k = 1, \ldots, o$

Neural Networks in R

```r
library(neuralnet)
nn <- neuralnet(Species == "setosa" ~ ., hidden = 3, iris, linear.output = FALSE)
plot(nn, rep = "best", show.weights = FALSE, intercept = FALSE)
```



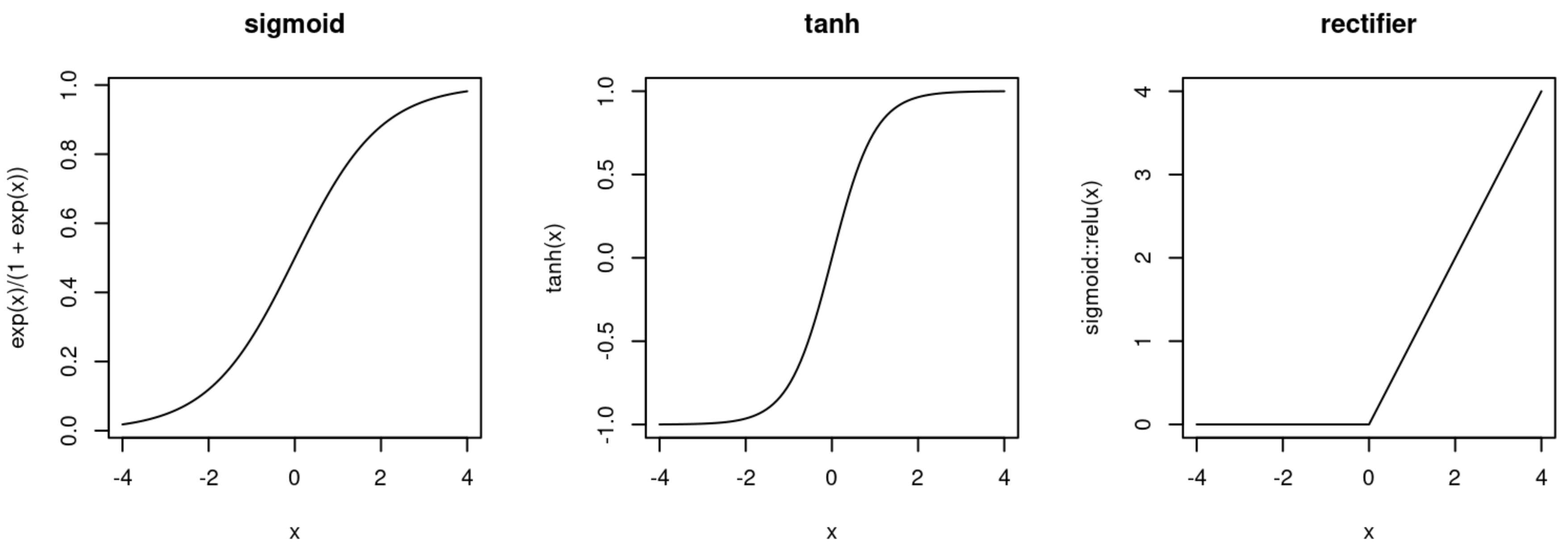
Universal Approximation Theorem

- The **universal approximation theorem** provides a
  - theoretical foundation for neural networks' ability to represent complex functions.
  - It states that a feedforward neural network with a single hidden layer containing a sufficient number of neurons can approximate any continuous function to any desired accuracy.
  - This theorem supports the idea that neural networks can learn and represent higher-level abstractions.

Non-linear Transformations

- Non-linear activation functions (such as ReLU, sigmoid, or tanh) to transform the input features.
- Allow the network to capture complex relationships and patterns that linear transformations cannot.

```r
par(mfrow = c(1, 3))
curve(exp(x)/(1 + exp(x)), from = -4, to = 4, main = "sigmoid")
curve(tanh(x), from = -4, to = 4, main = "tanh")
curve(sigmoid::relu(x), from = -4, to = 4, main = "rectifier")
```



Compositional Functions

- Composed of multiple layers, each performing a transformation on the input features.
- The output of one layer becomes the input to the next layer, creating a composition of functions, such as

$$(f_4 \circ f_3 \circ f_2 \circ f_1)(x) := f_4(f_3(f_2(f_1(x))))$$

- Each layer captures increasingly complex patterns and abstractions.

One hidden layer

- Hidden layer activation

$$\mathbf{h} = \phi_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

- Output layer activation

$$\begin{aligned}\mathbf{f}(\mathbf{x}) &= \phi_2(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2) \\ &= \phi_2(\mathbf{W}_2(\phi_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)) + \mathbf{b}_2)\end{aligned}$$

- The parameters of the model are the $\mathbf{W}$ (weight matrices) and the $\mathbf{b}$ (biases)
- $\mathbf{W}_1$ is a $q \times p$ weight matrix, $\mathbf{W}_2$ is $o \times q$, where q is the number of neurons in the hidden layer and o is the number of neurons in the output layer.
- $\mathbf{h}$ is of size q, $\mathbf{x}$ is of size p, $\mathbf{b}_1$ and $\mathbf{b}_2$ are of size q and o .
- ϕ_1 and ϕ_2 are non-linear activation functions.
- $\mathbf{f}(\mathbf{x})$ is multivariate for classification and univariate for regression.
- For regression, ϕ_2 is the identity function and we have 1 output neuron ($o = 1$).

Two hidden layers

- Hidden layer 1 activation

$$\mathbf{h}_1 = \phi_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

- Hidden layer 2 activation

$$\begin{aligned}\mathbf{h}_2 &= \phi_2(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)\\ &= \phi_2(\mathbf{W}_2(\phi_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)) + \mathbf{b}_2)\end{aligned}$$

- Output layer activation

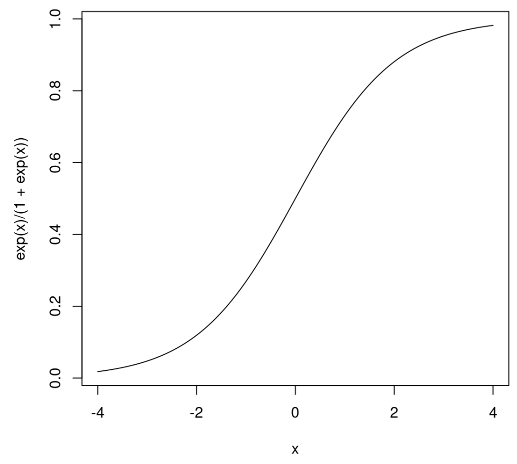
$$\begin{aligned}\mathbf{f}(\mathbf{x}) &= \phi_3(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)\\ &= \phi_3(\mathbf{W}_3(\phi_2(\mathbf{W}_2(\phi_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)) + \mathbf{b}_2) + \mathbf{b}_3))\end{aligned}$$

Two class-problem, one hidden layer

- The Softmax function converts a vector of k real numbers into a probability distribution of k possible outcomes.
- It is a generalisation of the logistic (sigmoid) function:

$$\phi(x) = \frac{\exp(x)}{1 + \exp(x)}$$

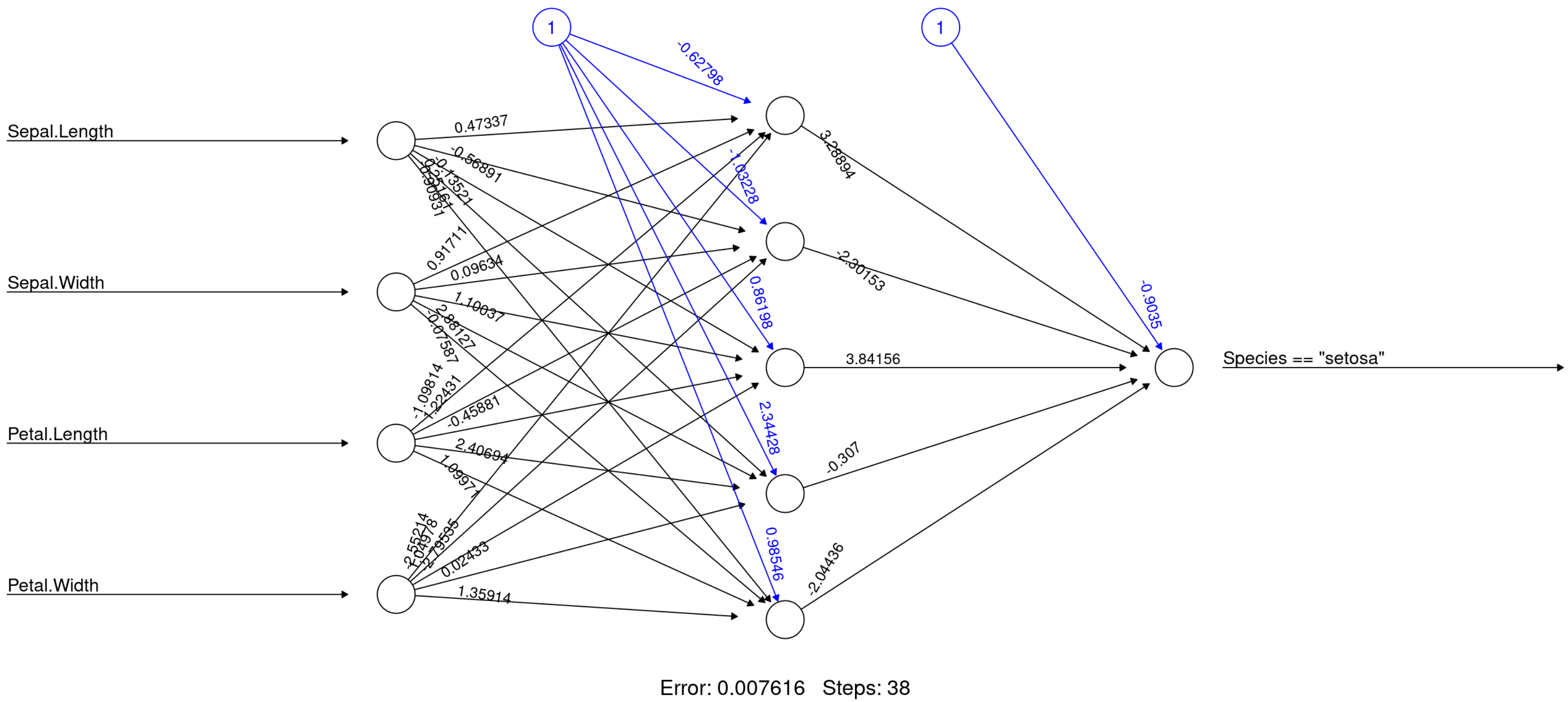
```r
curve(exp(x)/(1 + exp(x)), from = -4, to = 4)
```



Two class-problem, one hidden layer

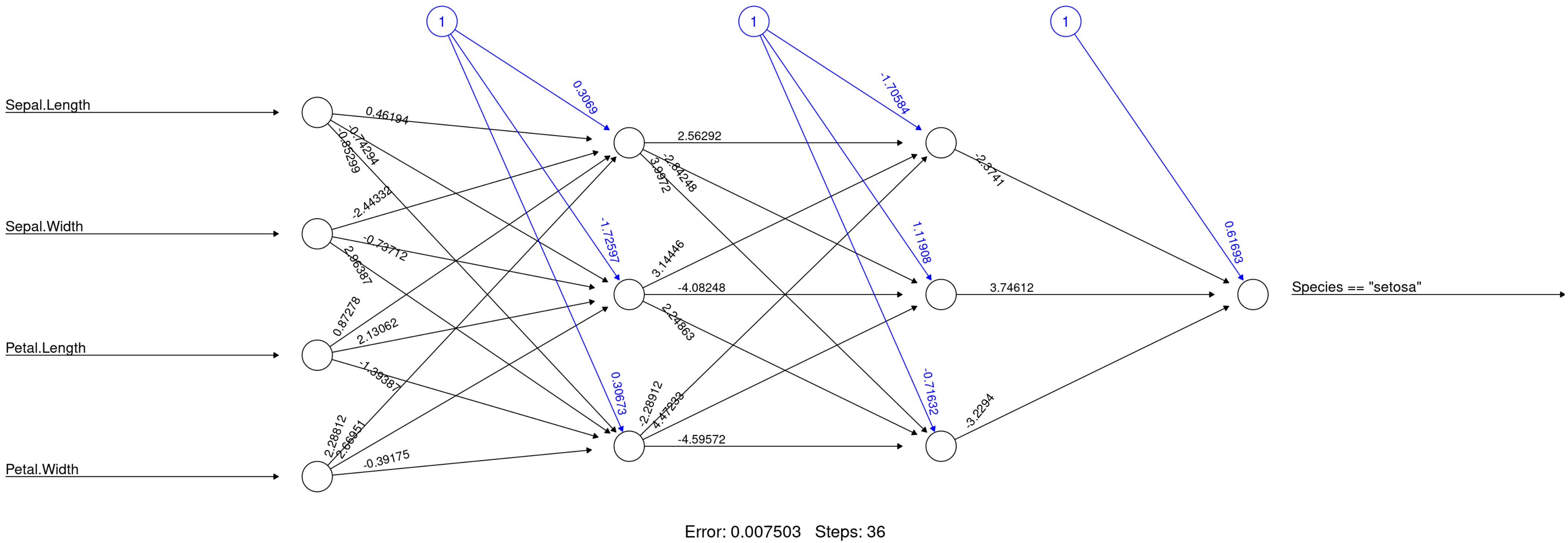
Fitted (Trained) model

```r
library(neuralnet)
nn <- neuralnet(Species == "setosa" ~ ., hidden = 5, iris, linear.output = FALSE)
plot(nn, rep = "best", show.weights = TRUE)
```



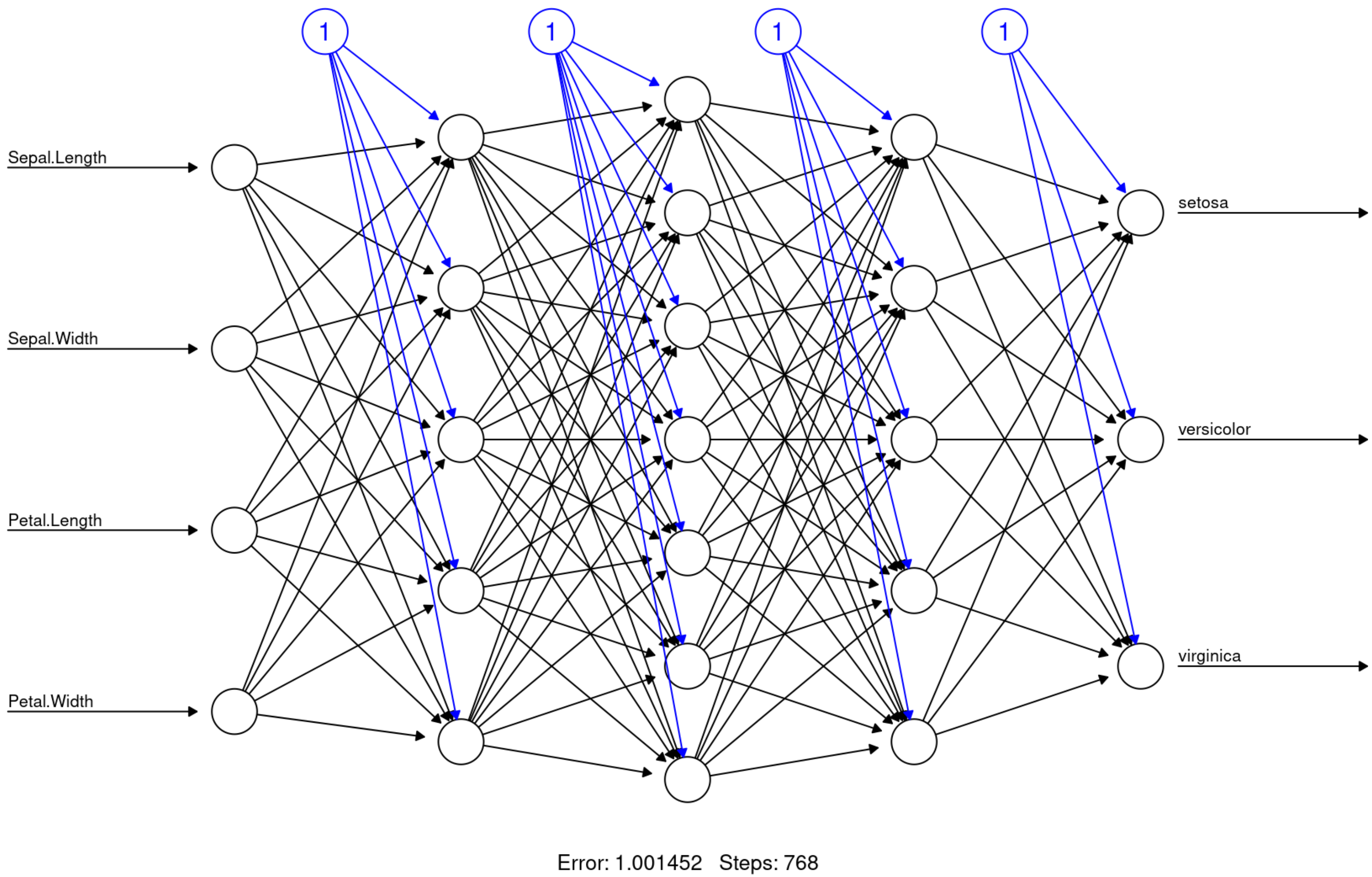
Two class-problem, two hidden layers

```r
nn <- neuralnet(Species == "setosa" ~ ., hidden = c(3, 3), iris, linear.output = FALSE)
plot(nn, rep = "best")
```



Three-class, three hidden layers

```r
nn <- neuralnet(Species ~ ., hidden = c(5, 7, 5), iris, linear.output = FALSE)
plot(nn, rep = "best", show.weights = FALSE, fontsize = 8, arrow.length = 0.15)
```



Loss calculation

- The goal is to minimize **loss**, which means making the predicted probabilities as close as possible to the actual labels.
  - If the predicted probability is close to the actual label, the loss will be small.
  - If the predicted probability is far from the actual label, the loss will be large.
- Cross-Entropy Loss is used for classification tasks.

$$\text{Cross-Entropy Loss} = -\frac{1}{p} \sum_{i=1}^{p} \sum_{k=1}^{o} y_{ik} \log(\hat{y}_{ik})$$

- **Minimizing the cross-entropy** loss is equivalent to **maximizing the log-likelihood** of the model parameters given the observed data.

Validating a model

- Validation metric (i.e. cross-entropy loss)
- Cross-validation, i.e. k-fold cross-validation
- Overfitting (performs well on training data, poorly on validation data)
- Underfitting (performs poorly on both training and validation data)
- Confusion matrix (TP, TN, FP, FN):
    - Sn, Sp, LR+, LR-, DOR, PPV, FDR, NPV, FOR, Jaccard, etc.

Validating a model

- Learning Curves
- Bias-Variance Tradeoff: Ensure that the model is neither too simple (high bias) nor too complex (high variance). This balance is crucial for optimal model performance.
- Hyperparameter Tuning (Hyperparameter are settings how the model is trained (Learning Rate, Batch Size, Network Architecture, Activation functions, Regularisation Parameters, Optimizer, Momentum etc.))
- Test on Unseen Data

Large Language Models

- Input Space Dimension (p)
  - Embedding Dimension: Each token in the input sequence is represented by a vector of size 12'288. This is the embedding dimension.
  - Each token is mapped to a 12'288-dimensional vector.
- Output Space Dimension (o)
  - Vocabulary Size: The output of the model is a probability distribution over the vocabulary. The size of this vocabulary can be 50'000 tokens or more.
  - The output vector for each token is a 50'000-dimensional vector representing the probabilities of each token in the vocabulary.
- Hidden Layers: GPT-3: 96, GPT-4: 120
- Parameters: GPT-3: 175×10^9, GPT-4: 1.8×10^{12}