3.8 Neural Networks

3.8.1 Intro

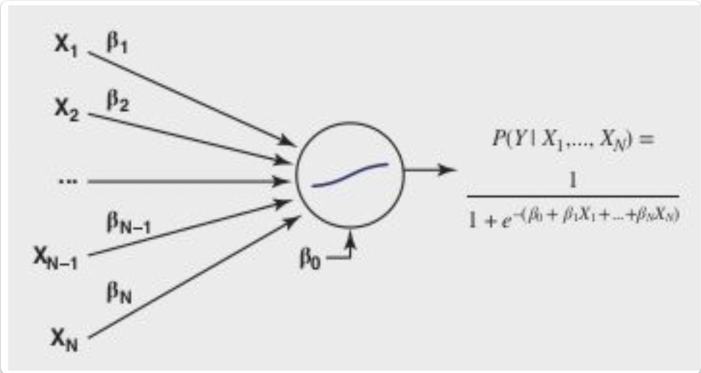
#DEF **Neural networks are mathematical representations inspired by the functioning of the human brain.**

> *Generalizations of existing statistical models.*

Neural networks can model *very complex patterns and decision boundaries in the data*.

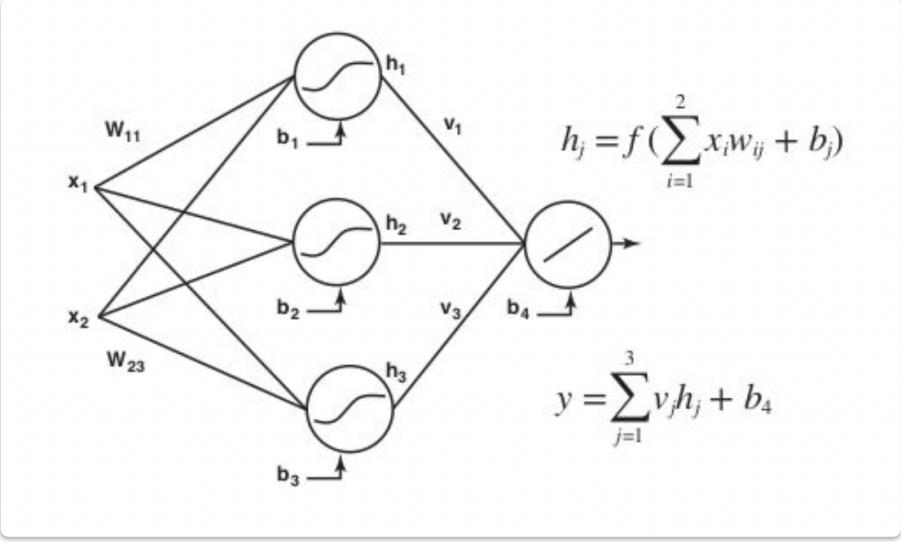
A processing element or neuron performs *two operations*:

1. It **takes the inputs and multiplies them with the weights** (including the intercept term β_0 , called the bias term)
2. Puts this into a **nonlinear transformation function** (~logistic regression).
    - Logistic (and linear) regression is a neural network with one neuron:



3.8.2 MultiLayer Perceptron Neural Network #MLP

- Input Layer
- Hidden Layer
    - Works like a *feature extractor*: combine the inputs into features that are then subsequently offered to the output layer.
    - Has a *nonlinear transformation function* f().
- Output layer
    - Makes the *prediction*.
    - Has a *linear transformation function*.



3.8.3 Transformation functions (activation functions)

The activation functions *may differ per neuron*, they are typically **fixed for each layer**:
In the output layer:

- For **classification** (e.g., fraud detection):
  - A *logistic transformation*, since the outputs can then be interpreted as *probabilities*.
- For **regression** (e.g., amount of fraud):
  - Logistic, $f(z) = \frac{1}{1+e^{-z}}$, ranging between 0 and 1
  - Hyperbolic tangent, $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, ranging between −1 and +1
  - Linear, $f(z) = z$, ranging between $-\infty$ and $+\infty$

  In the hidden layer:
- *Hyperbolic tangent* (tanH) activation function.

3.8.4 Details

In the *fraud analytics setting, complex patterns rarely occur*.

Recommended to use a *neural networks with one hidden layer*: **universal approximators**, capable of approximating any function to any desired degree of accuracy on a compact interval.

Data preprocessing:

- For *continuous variables*, standardization can be used.
- For *categorical variables (only)*, categorization can be used to reduce the number of categories.

3.8.5 Weight Learning

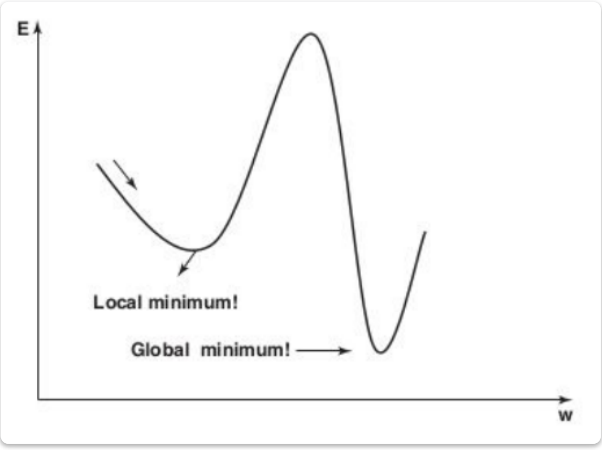
The *optimization* (for optimal parameter values) **is more complex**: Iterative algorithm that optimizes a cost-function:

- *Continuous target variable*
  - **Mean Squared Error (MSE) cost function**.
- *Binary target variable*
  - **Maximum Likelihood cost function**.

> *The procedure starts from a set of random weights, which are then iteratively adjusted to the patterns in the data using an **optimization algorithm** (e.g., Back propagation learning, Conjugate gradient).*

🔑 **Key issue**: the curvature of the objective function is not convex and may be multimodal.

The **error function** can thus have *multiple local minima* but typically *only one global minimum*.



If the starting weights are chosen in a suboptimal way, *one may get stuck in a local minimum*.

3.8.5.1 Preliminary Training

- Try out *different* starting weights.
- Start the *optimization procedure* for a few steps.
- Continue with the *best intermediate solution*.

3.8.5.2 Stopping Criterion

The optimization procedure then continues until:

- The *error function shows no further progress*.
- The *weights stop changing substantially*.
- After a *fixed number of optimization steps* (Epochs).

3.8.5.3 Hidden Neurons (Weight and) Number

Number of hidden neurons -> nonlinearity in data.

- *More complex, nonlinear patterns* -> more hidden neurons.

They should be carefully tuned:

1. Split the data into a *training, validation, and test set*.
2. Vary the *number of hidden neurons*.
3. *Train* a neural network on the training set.
4. *Measure the performance* on the validation set.
5. *Choose* the number of hidden neurons with optimal validation set performance.
6. *Measure the performance* on the independent test set.

3.8.5.4 Overfitting Problem

Neural networks can model *very complex patterns* and decision boundaries in the data -> They can even model the noise in the training data -> **Overfitting**

Option 1

- *Training set* -> estimate the weights.
- *Validation set* -> independent data set used to decide when to stop training **and avoid this overfitting**.

Option 2
Weight regularization: keep *weights small in absolute sense* to avoid fitting the noise in the data.

- *Add a weight size term* to the objective function of the neural network.

3.8.5.5 The Neural Network Black Box

#DEF ***Black-box*: relate inputs to outputs in a mathematically complex, nontransparent, and opaque way.**

- Applied as high-performance analytical tools in settings where interpretability is not a key concern.
- In application areas where insight into the fraud behavior is important, one needs to be careful with NNs.

Opening the **Neural Network black box**:

1. Variable selection.
2. Rule extraction.
3. Two-stage models.

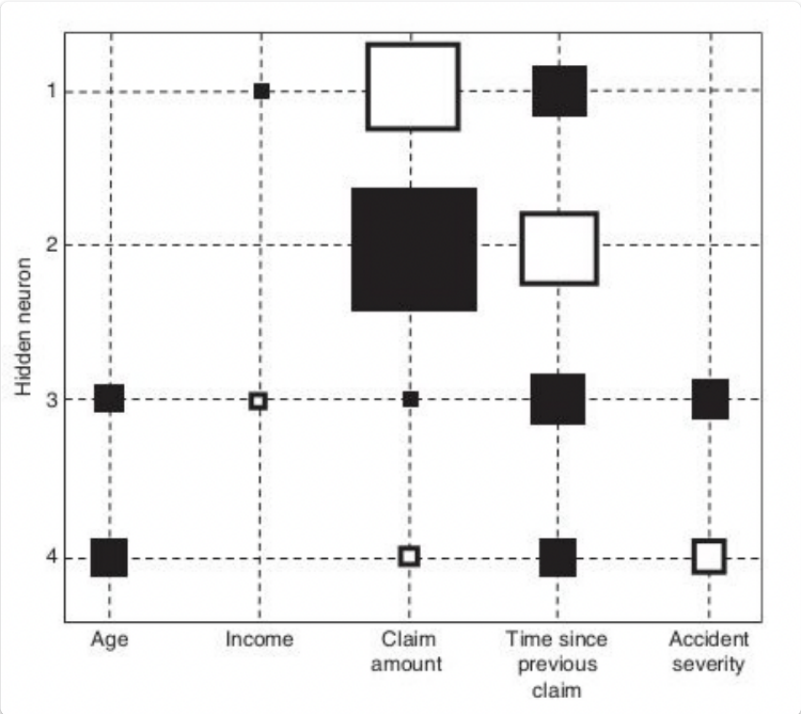
3.8.5.6 Variable Selection

#DEF **Select variables that actively contribute to the NN output.**
In linear and logistic regression -> *inspecting the p-values*.
In neural networks -> *no p-values*.

3.8.5.6.1 HINTON DIAGRAM

Visualizes the weights between the inputs and the hidden neurons as squares:

- *Size of the square* is proportional to the **size** of the weight.
- *Color of the square* represents the **sign** of the weight (e.g., black=negative weight and white=positive weight).



Steps:

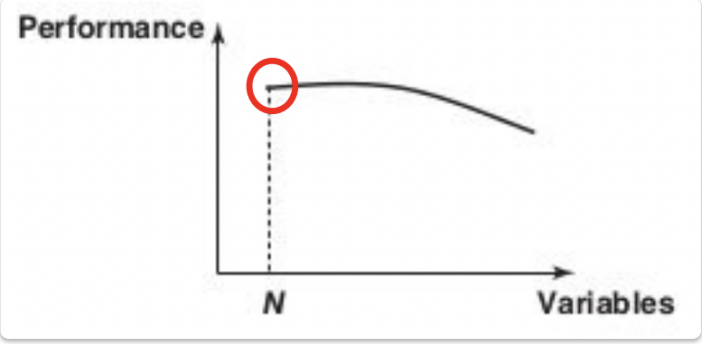
1. Inspect the **Hinton diagram** and *remove the variable whose weights are closest to zero*.
2. *Reestimate the neural network with the variable removed*. To speed up the convergence, it could be beneficial to start from the previous weights.
3. *Continue with step 1 until a stopping criterion is met*. The stopping criterion could be a decrease of predictive performance or a fixed number of steps.

3.8.5.6.2 BACKWARD VARIABLE SELECTION

Backward Variable Selection is a performance driven selection.

Steps:

1. Build a neural network *with all N variables*.
2. *Remove each variable in turn and reestimate the network*. This will give N networks each having N – 1 variables.
3. *Remove the variable whose absence gives the best performing network* (e.g., in terms of misclassification error, mean squared error).
4. *Repeat* this procedure *until the performance decreases significantly*.



3.8.5.6.3 VARIABLE SELECTION - DISCUSSION

Variable selection:

- Allows users to *see which variables are important* and which ones are not.
- It *does not offer a clear insight into its internal workings*. The relationship between the inputs and the output remains nonlinear and complex.

> *Sampling can be used to make the procedure less resource intensive and more efficient.*

3.8.5.7 Rule Extraction Procedure

#DEF Extract if-then classification rules, mimicking the behavior of the neural network:

3.8.5.7.1 DECOMPOSITIONAL TECHNIQUE

#DEF Decompose the network's internal workings by *inspecting weights and/or activation values*.

Example:

Customer	Age	Income	Known Customer	...	Fraud
Emma	28	1000	Y		No
Wil	44	1500	N		Yes
Dan	30	1200	N		No
Bob	58	2400	Y		Yes

Step 1: Start from original data

Customer	Age	Income	Known Customer	h1	h2	h3	h1	h2	h3	Fraud
Emma	28	1000	Y	−1,20	2,34	0,66	1	3	2	No
Wil	44	1500	N	0,78	1,22	0,82	2	3	2	Yes
Dan	30	1200	N	2,1	−0,18	0,16	3	1	2	No
Bob	58	2400	Y	−0,1	0,8	−2,34	1	2	1	Yes

Step 2: Build a neural network (e.g., 3 hidden neurons)

Step 3: Categorize hidden unit activations

If h1 = 1 **and** h2 = 3 **Then** Fraud = No
If h2 = 2 **Then** Fraud = Yes

Step 4: Extract rules relating network outputs to categorized hidden units

If Age < 28 **and** Income < 1000 **Then** h1 = 1
If Known Customer = Y **Then** h2 = 3
If Age > 34 **and** Income > 1500 **Then** h2 = 2

Step 5: Extract rules relating categorized hidden units to inputs

If Age < 28 **and** Income < 1000 **and** Known Customer = Y **Then** Fraud = No
If Age > 34 **and** Income > 1500 **Then** Fraud = Yes

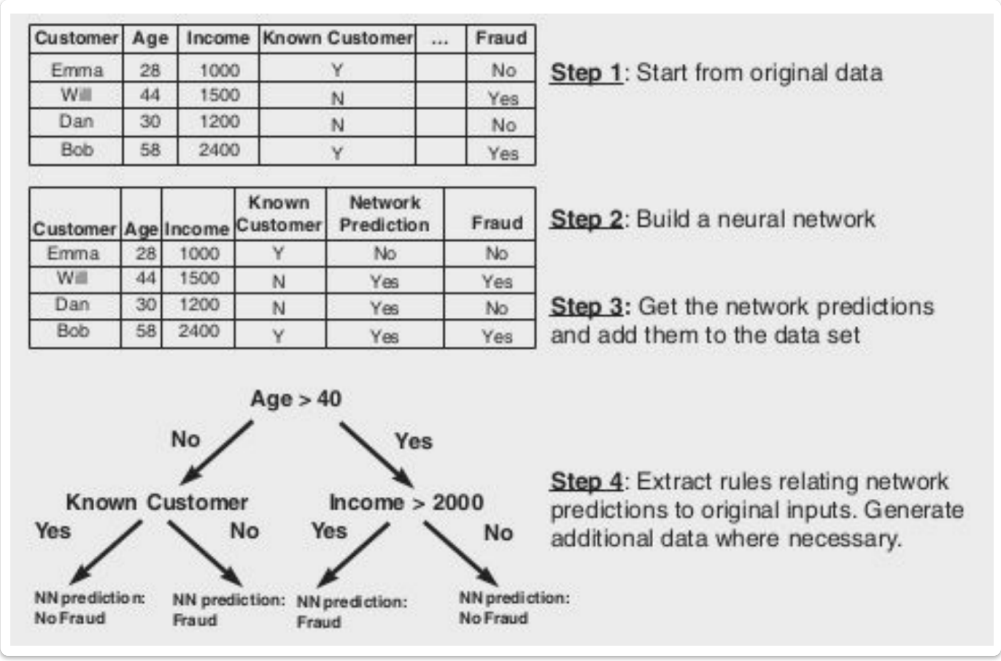
Step 6: Merge both rule sets

3.8.5.7.2 PEDAGOGICAL TECHNIQUE

#DEF **Consider the neural network as a black box and use the neural *network predictions as input* to a white-box analytical technique such as decision trees.**

- Can essentially be used with any underlying algorithm.

Example:



3.8.5.8 Two-stage Model Setup

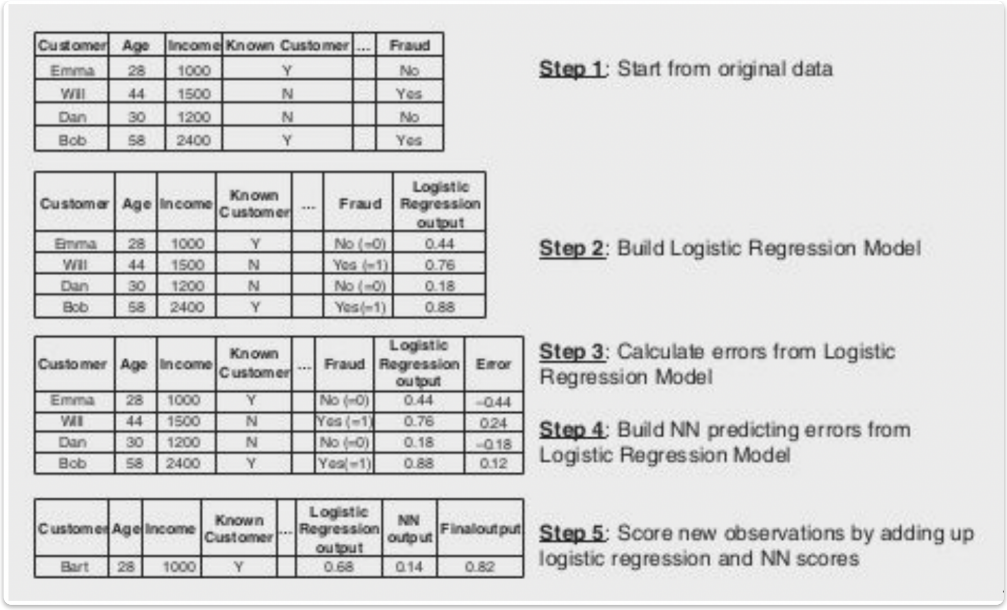
1. Estimate an *easy-to-understand model first* (e.g., linear regression, logistic regression).
    - Estimate an easy-to-understand model first (e.g., linear regression, logistic regression).
2. Use a *neural network to predict the errors* made by the simple model using the same set of predictors.
    - Performance benefit of using a nonlinear model.

Both models are then combined in an additive way, for example as follows:

- $\text{Target} = \text{Linear regression}\ (X_1, X_2, \ldots X_N) + \text{Neural network}\ (X_1, X_2, \ldots X_N)$
- $\text{Score} = \text{Logistic regression}\ (X_1, X_2, \ldots X_N) + \text{Neural network}\ (X_1, X_2, \ldots X_N)$

This provides an **ideal balance** *between model interpretability* (which comes from the first part) *and model performance* (which comes from the second part).

Example:



Next chapter: Support Vector Machine