NEURAL NETWORKS

E. Fersini

NEURAL NETWORK

- A **neural network** is constructed with an interconnected group of nodes, which involves the input, connected weights, hidden nodes, and output.

- To train a neural network in R you can use the package **neuralnet**, which is built to train multilayer perceptron in the context of regression analysis and contains many functions to train forward neural networks.

NEURAL NETWORK

1. First load the iris dataset and split the data into training and testing datasets:

```
> data(iris)
> ind = sample(2, nrow(iris), replace = TRUE, prob=c(0.7, 0.3))
> trainset = iris[ind == 1,]
> testset = iris[ind == 2,]
```

2. Add the columns versicolor, setosa, and virginica based on the name matched value in the Species column:

```
> trainset$setosa = trainset$Species == "setosa"
> trainset$virginica = trainset$Species == "virginica"
> trainset$versicolor = trainset$Species == "versicolor”
```

NEURAL NETWORK

3. Train the neural network with the neuralnet() function with *three hidden neurons* in each layer:

```
> library(neuralnet)
> network = neuralnet(versicolor + virginica + setosa~ Sepal.Length +
Sepal.Width + Petal.Length + Petal.Width, trainset, hidden=3)
> network
```

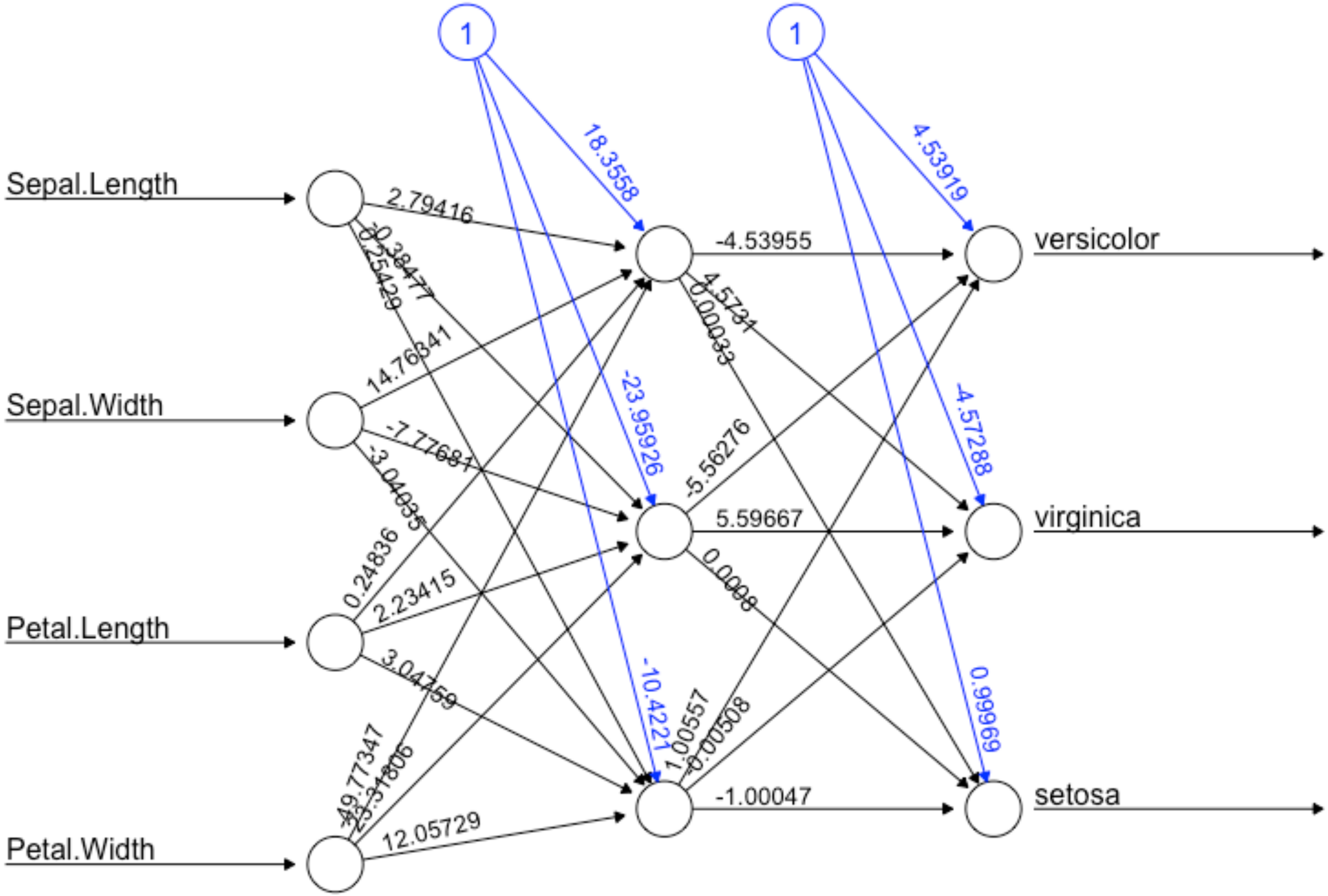
4. You can compare the initial (random) weights with the estimated weights, and visualize the overall neural network:

```
> network$startweights
```

```
> network$weights
```

```
> plot(network)
```

NEURAL NETWORK



Error: 0.337205   Steps: 57233

NEURAL NETOWORK

- Understanding the neural network:
  - If w > 0 than zero, it is in an excitation status.
  - If w <= 0, it is in an inhibition status.
  - Input neurons receive the input information:
    - the higher the input value, the greater the activation.
  - It can detect the non-linear relationships between the input features and the target variable (class)
  - the training process needed 6132 (?) steps until convergence
    - the absolute partial derivatives of the error function were lower than the default threshold 0.01

NEURAL NETWORK

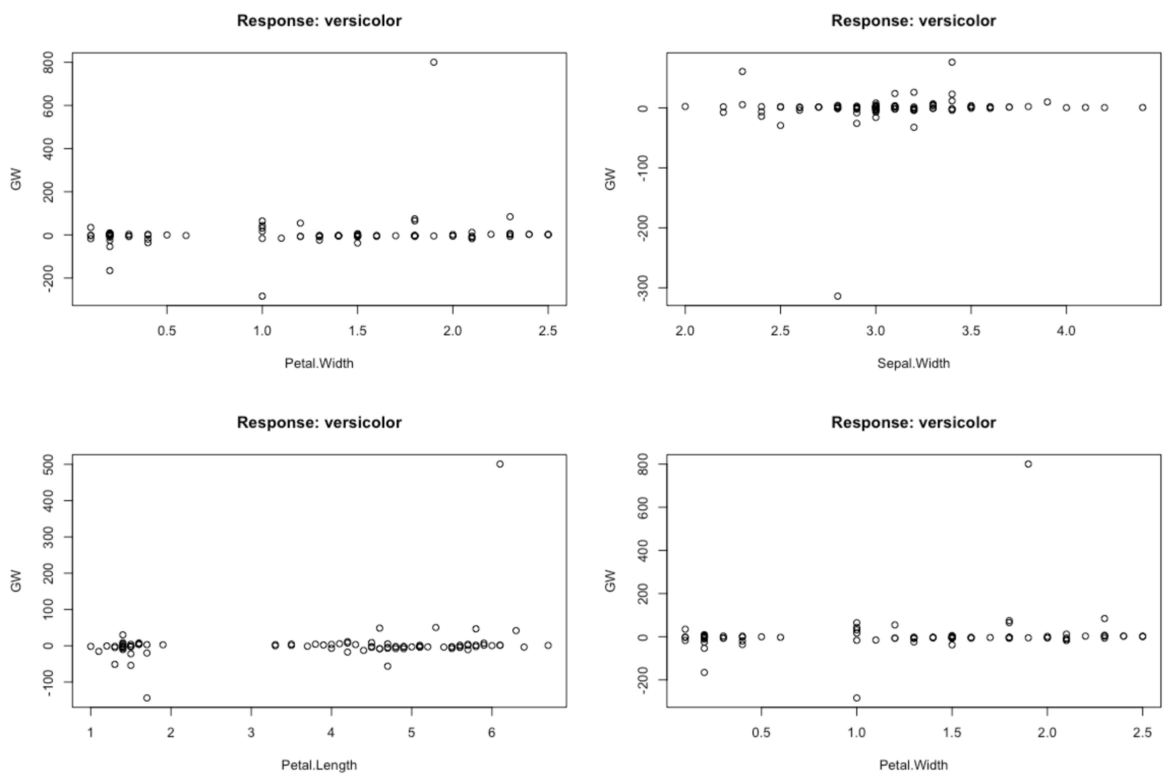
- You can also understand the **generalized weights**, which express the effect of each feature x_i
  - It has an analogous interpretation as the *i*th regression parameter in regression models.

$$\tilde{w}_i = \frac{\partial \log\left[\frac{P(y)}{1-P(y)}\right]}{\partial x_i}$$

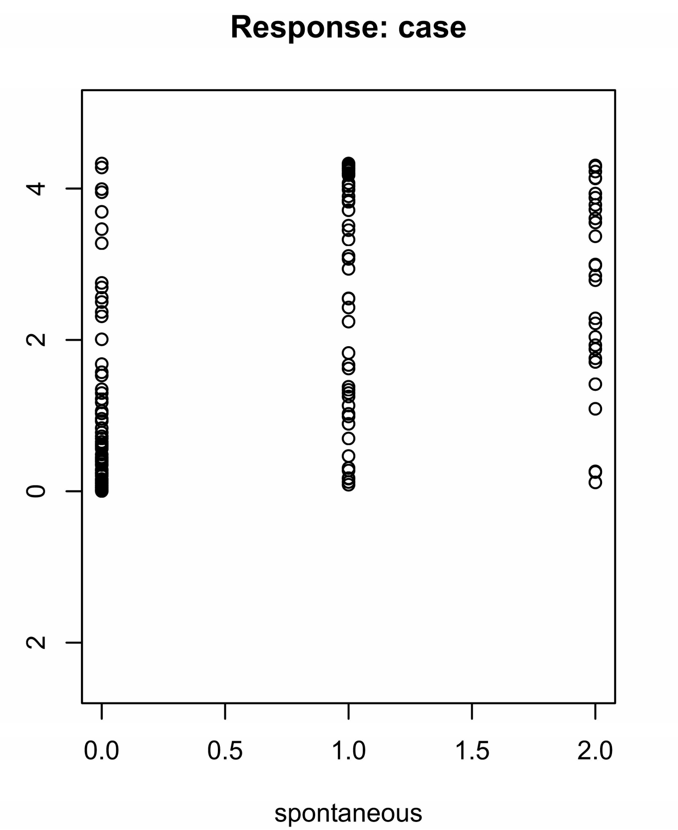
```
par(mfrow=c(2,2))
gwplot(network,selected.covariate="Petal.Width")
gwplot(network,selected.covariate="Sepal.Width")
gwplot(network,selected.covariate="Petal.Length")
gwplot(network,selected.covariate="Sepal.Length")
```

NEURAL NETOWORK

If all the generalized weights are close to zero, it means the feature has little effect.



NEURAL NETWORK



if the overall variance of generalized weights is greater than one, it means the covariate has a nonlinear effect.

PREDICTION ON NEURAL NETWORKS

- We can use the compute() function to obtain a probability distribution of labels for each test instance:

```
> net.predict = compute(network, testset[-5])$net.result
```

- To obtain the predicted class value for each test instance, take the class label with the highest probability:

```
> net.prediction = c("versicolor", "virginica", "setosa")[apply(net.predict, 1,
which.max)]
```

- Generate a classification table based on ground truth labels and predicted labels, then compute accuracy:

```
> predict.table = table(testset$Species, net.prediction)
```

REMARKS

- The default activation function is the **logistic function**:
  - It is appropriate for binary response variables since it maps the output of each neuron to the interval [0, 1]. However, it suffers of saturation during the gradient computation
    - if the activity in the network during training is close to 0 then the gradient for the logistic (or sigmoid) activation function may go to 0

ASSIGNMENT

- Does the "**tanh**" activation function achieve better performance?
- How many epocs are needed to converge?
- Does the result change with more hidden layers?
- What is the effect of a different loss function?