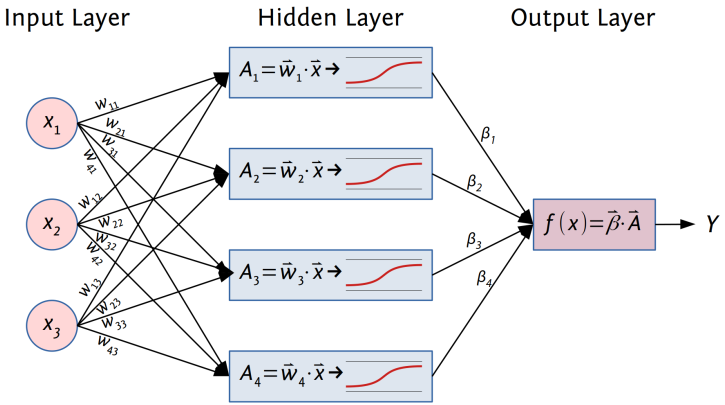
Introduction to Neural Networks

Neural Networks combine the outputs of multiple perceptron-like neurons.

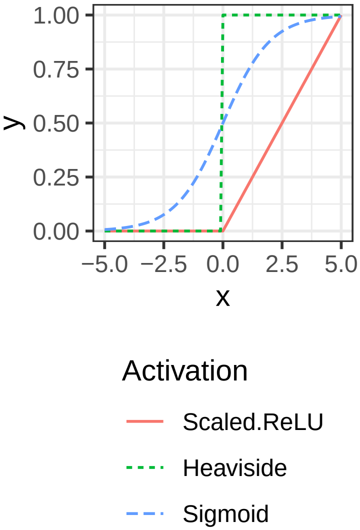


Notation: $\vec{w} \cdot \vec{x}$ is shorthand for

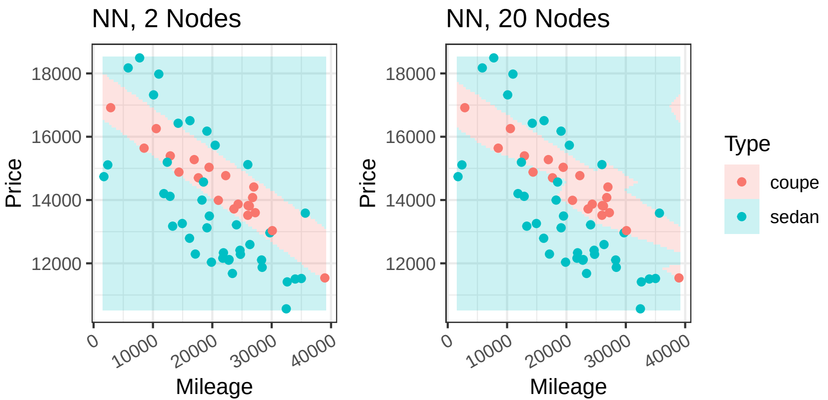
$$\langle w_0, \ldots, w_n \rangle \cdot \langle 1, x_1, \ldots, x_n \rangle = w_0 + w_1 x_1 + \cdots + w_n x_n.$$

The *activation function* might vary. Recent implementations prefer simpler activation functions.

- The *activation function* is a non-linear function that scales the weighted sum of inputs: $g(\vec{w} \cdot \vec{x})$.
- Sigmoid: $g(z) = \frac{1}{1+e^{-z}}$
- Heaviside:
  $$g(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$
- ReLU: $g(z) = \begin{cases} z & z \geq 0 \\ 0 & z < 0 \end{cases}$

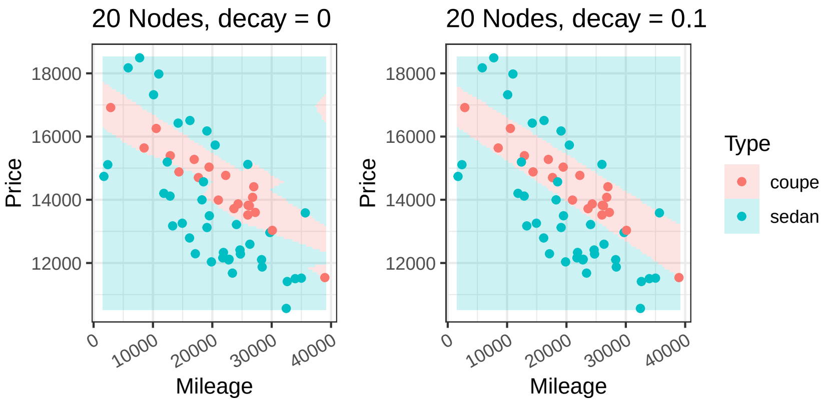


Each node in the hidden layer registers distance from a dividing line, combining to create the prediction.



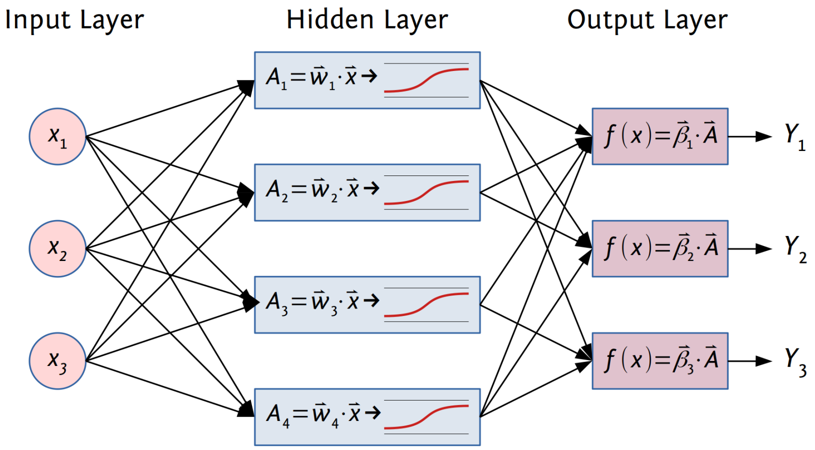
More nodes can lead to overfitting.

The decay parameter penalizes large weights as the network is trained, helping to reduce overfitting.



Simplified Idea: Rather than minimize $Loss(\vec{w})$, minimize $Loss(\vec{w})$ + $(Decay\ Const) \times |\vec{w}|^2$.

For multi-level prediction, the output layer has several nodes. Classification is one-vs-all.

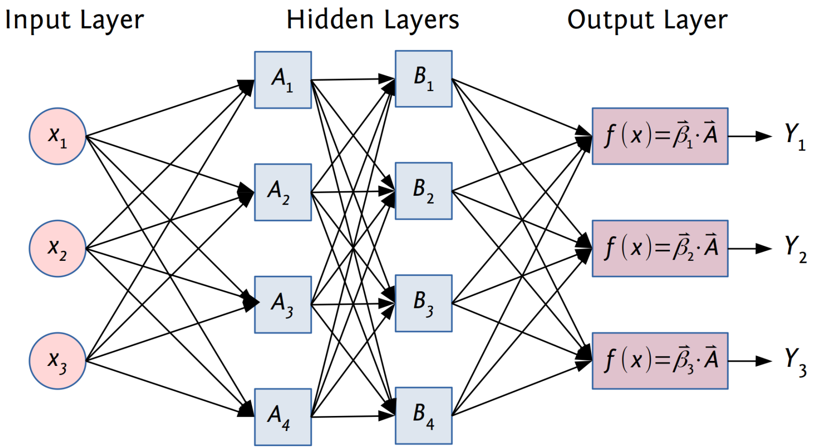


For probabilities, one might set $P(\text{Outcome } i) = \frac{e^{Y_i}}{\sum_1^K e^{Y_j}}$.

Neural networks can be used for *regression* (i.e. predicting numeric response variables).

- The algorithm attempts to find weights that minimize RMSE

Multi-layer neural networks add further connected hidden layers.



Notes on Single and Multi-Layer Networks

- The `nnet` package and command implements single-hidden-layer networks.
- The `neuralnet` package implements multiple hidden layers.
- Both are supported by `caret`, although `neuralnet` may only be supported for regression..
- Single-layer networks are versatile, but may need many nodes.
- Multi-layer networks can be more suitable to certain complex tasks (like image recognition).