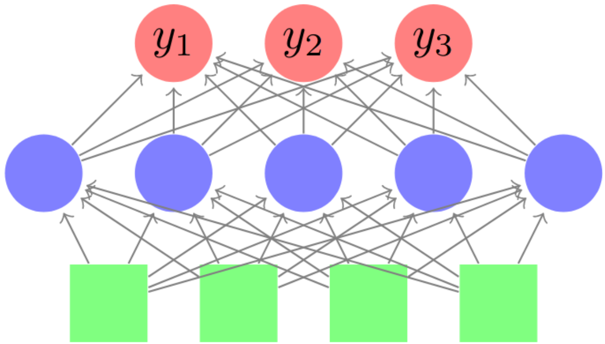

A linear[1] perceptron with two hidden layers (the one above has just a single layer):

$$\hat{\mathbf{Y}} = \mathsf{softmax}(\mathbf{b}_{(2)} + \mathbf{W}_{(2)}(\mathbf{b}_{(1)} + \mathbf{W}_{(1)}\mathbf{X})),$$

where $\mathbf{X} \in \mathrm{R}^{n \times m}$ is the matrix of inputs (n being the number of examples in a single batch, and m the number of features); $\hat{\mathbf{Y}} \in \mathrm{R}^{n \times l}$ (l being the number of output labels) is the matrix of output labels; $\mathbf{W}_{(1)} \in \mathrm{R}^{m \times k}$ is the weight matrix of the first layer together with a bias $\mathbf{b}_{(1)} \in \mathrm{R}^{k \times 1}$; $\mathbf{W}_{(2)} \in \mathrm{R}^{k \times l}$ the weight matrix of the second layer, and $\mathbf{b}_{(2)} \in \mathrm{R}^{l \times 1}$ is the bias of the second layer.

The $\mathsf{softmax}$ function calculates a normalized distribution over output labels. To be more specific, here we have for a single example $\mathbf{x} \in \mathrm{R}^{1 \times n}$, and the network's output vector being $\hat{\mathbf{y}} \in \mathrm{R}^{1 \times l}$ (where individual components are denoted by $\hat{\mathbf{y}}_j$):

$$\frac{e^{\hat{\mathbf{y}}_i}}{\sum_{j=1}^{l} e^{\hat{\mathbf{y}}_j}}.$$

This model can be readily trained with gradient descent methods, such as (mini-batch) stochastic gradient descent or Adam. As the loss function we can use the cross-entropy loss:

$$-\sum_{j=1}^{l} \mathbf{y}_j \log(\hat{\mathbf{y}}_j)$$

Where $\mathbf{y}$ is a one-hot vector, where the components are all 0 except for the index that corresponds to the true label.

[1]The intermediate activation function is just the identity in this example.