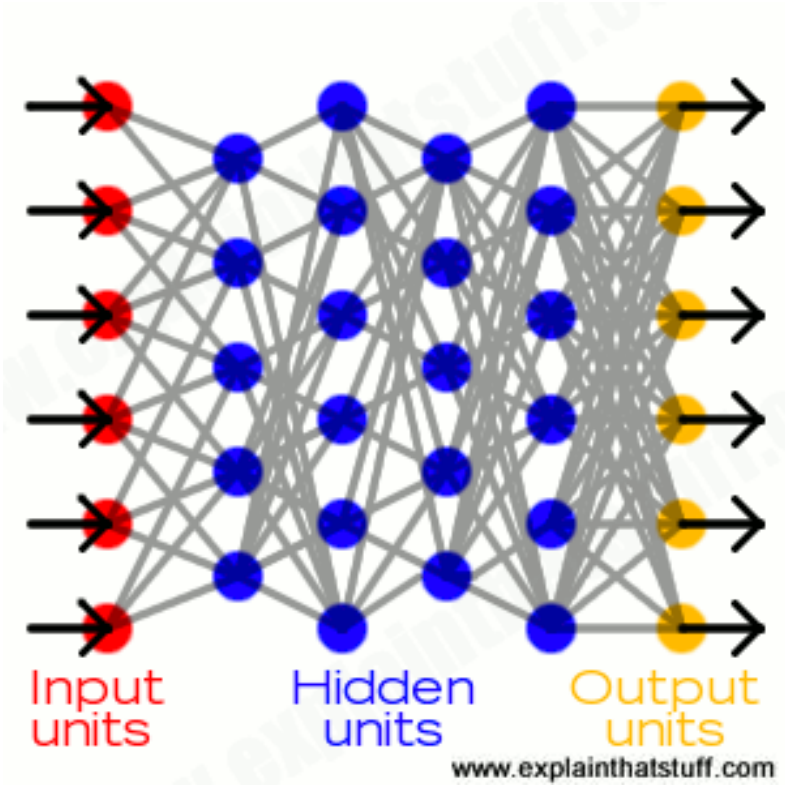
Neural Network

Yang Wang

College of Charleston

Neural network

- Goal: predict Y for given X
- Supervised method
- May be used for both classification and regression



picture: https://www.explainthatstuff.com/introduction-to-neural-networks.html

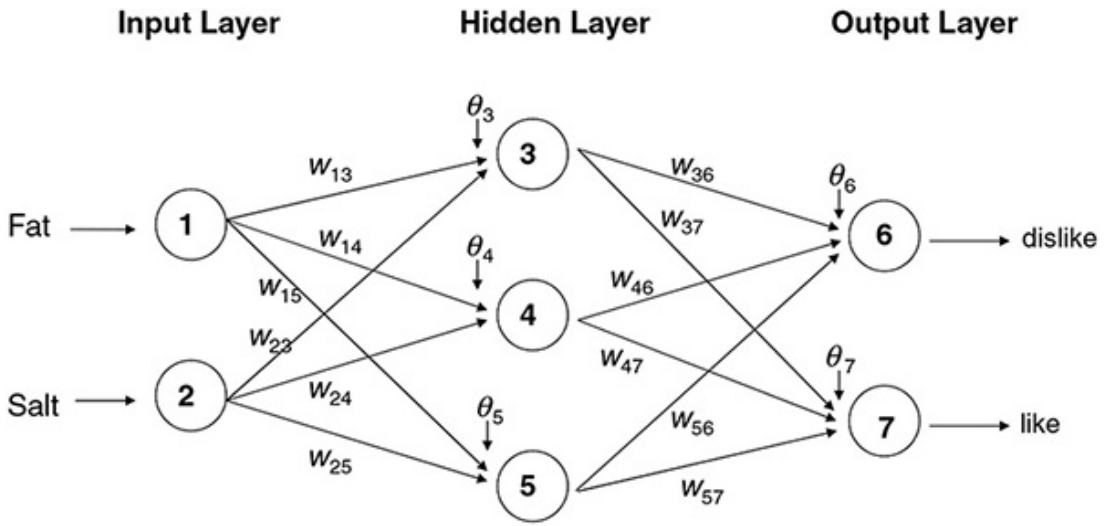
Example on tiny dataset

- Consider the following dataset with 6 observations and 3 variables which includes information on a tasting score for a certain processed cheese.
- The predictor variables are scores for fat and salt, indicating the relative presence of fat and salt in that particular sample. (min 0, max 1)
- The response variable is the consumer taste preference and has two levels ”like” and ”dislike”.

Obs	Fat Score	Salt Score	Acceptance
1	0.2	0.9	like
2	0.1	0.1	dislike
3	0.2	0.4	dislike
4	0.2	0.5	dislike
5	0.4	0.5	like
6	0.3	0.8	like

Example on tiny dataset

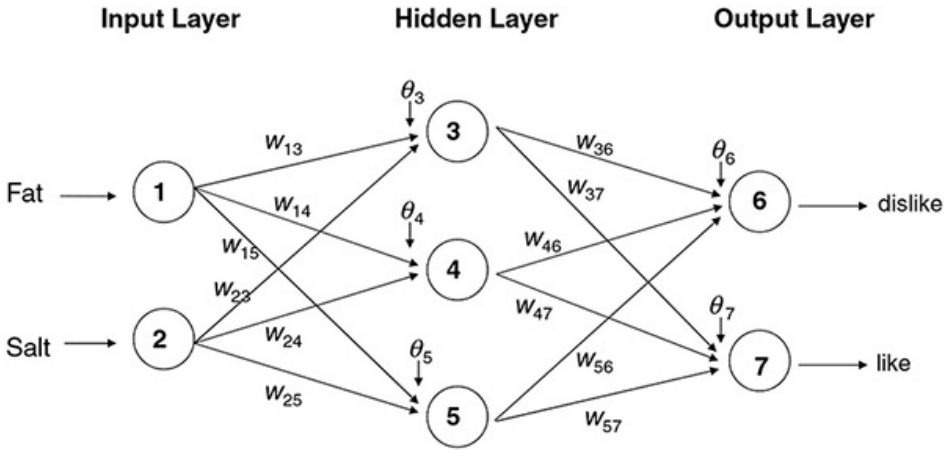
- Nodes 1 and 2 belong to the input layer (predictor variables ”Fat score” and ”Salt Score”).
- Nodes 3 to 5 belong to the hidden layer.
- Nodes 6 and 7 belong to the output layer (binary response variable with labels ”like” and ”dislike”).
- The values on the connecting arrows are called *weights*. The weight on the arrow from node *i* to node *j* is denoted by w_{ij}.
- The additional *bias* nodes, denoted by θ_j, serve as an intercept for the output from node *j*.



Example on tiny dataset: Input layer

- If we have p predictors, the input layer will usually include p nodes.
- Neural network works best when the numerical variables (including response) is rescaled to [0, 1] interval.
- For the first record, the input into the input layer is $Fat = 0.2$ and $Salt = 0.9$.
- The output from the input layer is the same as the input.
- The output of the layer is $x_1 = 0.2$ and $x_2 = 0.9$ (Let's denote Fat as x_1 and Salt as x_2).

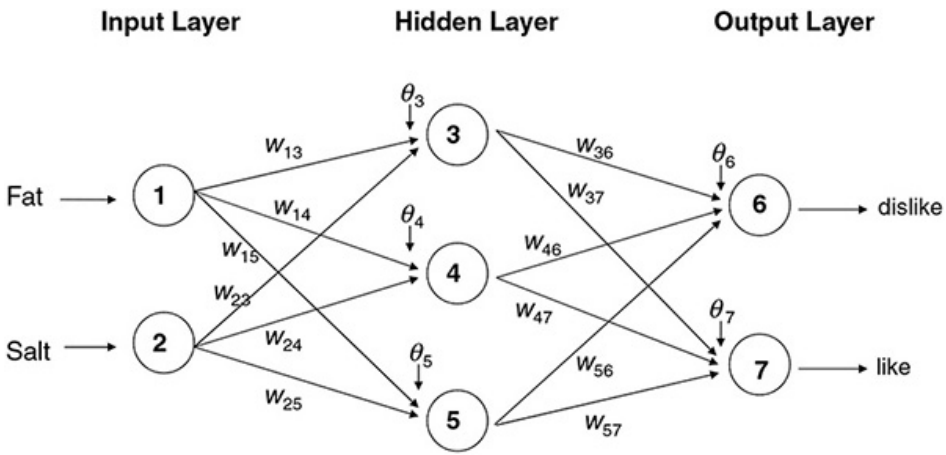
Obs	Fat Score	Salt Score	Acceptance
1	0.2	0.9	like
2	0.1	0.1	dislike
3	0.2	0.4	dislike
4	0.2	0.5	dislike
5	0.4	0.5	like
6	0.3	0.8	like



Example on tiny dataset: Hidden layer

- Hidden layer nodes take as input the output values from the input layer *i.e.* $x_1 = 0.2$ and $x_2 = 0.9$.
- Nodes 3 to 5 are hidden layer nodes. Each receive input from all the input nodes.

Obs	Fat Score	Salt Score	Acceptance
1	0.2	0.9	like
2	0.1	0.1	dislike
3	0.2	0.4	dislike
4	0.2	0.5	dislike
5	0.4	0.5	like
6	0.3	0.8	like

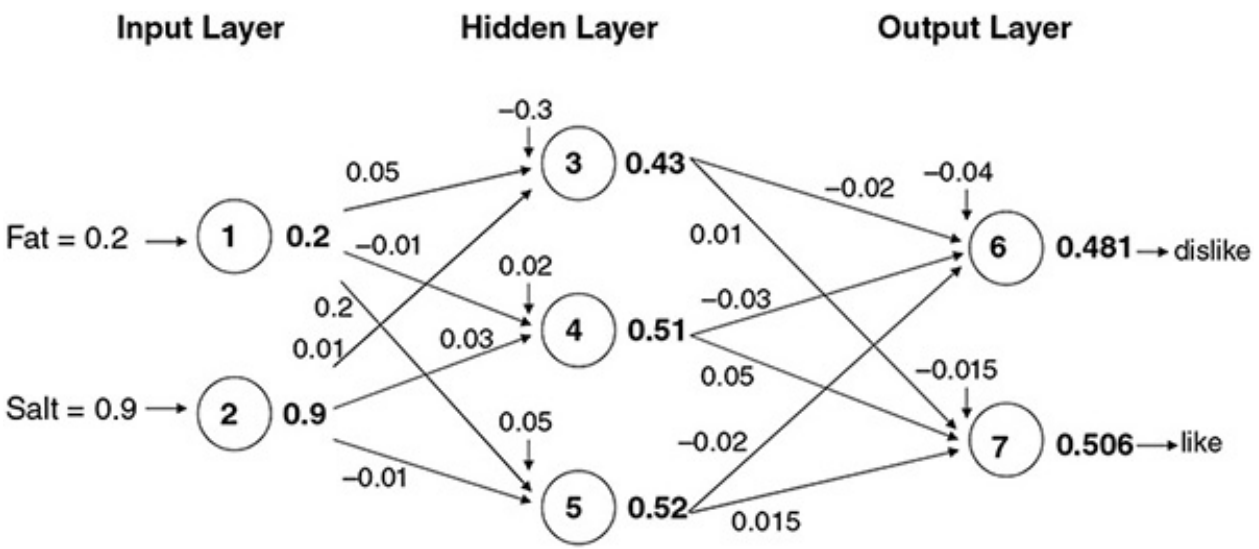


Example on tiny dataset: Hidden layer

- The output of a hidden layer node is computed by taking a weighted sum of the inputs and applying a certain function to it. Output of node j is given by $g(\theta_j + \sum_{i=1}^{p} w_{ij} x_i)$.
  - The weights w_{ij}'s are initially set randomly and then modified as the network "learns".
  - The bias θ_j is a constant that controls the level of contribution of node j. This is also set randomly at the beginning.
  - p is number of nodes of the previous layer.
  - The function g is some monotone function and it is known as a transfer function or activation function.
  - Example of g: linear function ($g(s) = bs$), exponential function ($g(s) = e^{bs}$), logistic/sigmoidal function ($g(s) = 1/(1 + e^{-s})$).
  - Output for a logistic activation function: $g(\theta_j + \sum_{i=1}^{p} w_{ij} x_i) = \frac{1}{1+e^{-(\theta_j + \sum_{i=1}^{p} w_{ij} x_i)}}$.

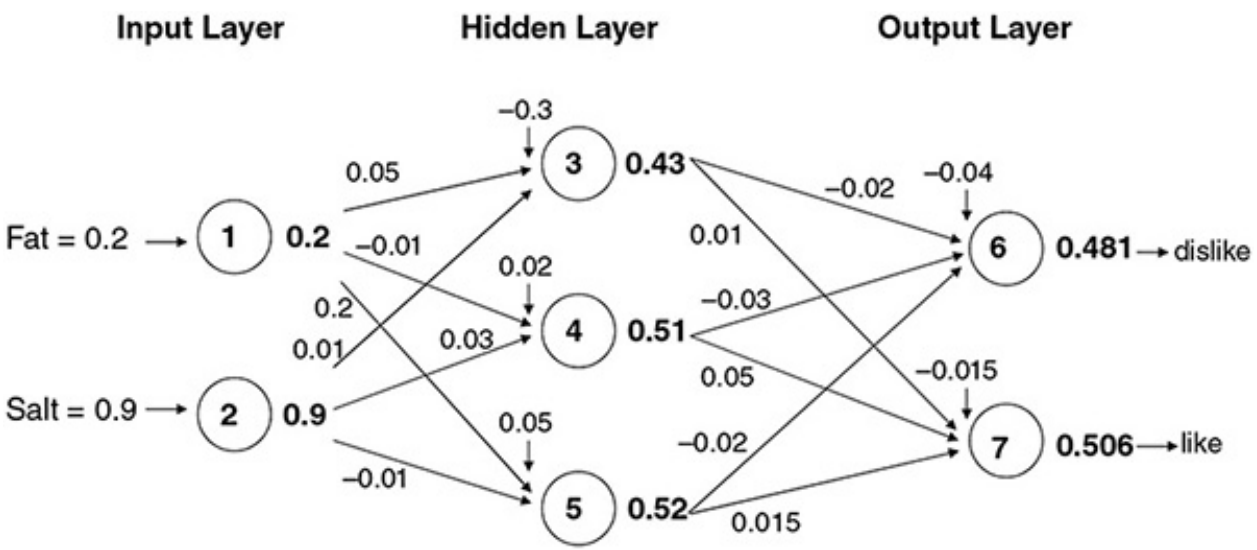
Example on tiny dataset: Hidden layer

- The values of θ_j and w_{ij} are initialized to small, usually random, numbers (typically 0.00 ± 0.05).
- Such small values represent a state of no knowledge by the network, similar to a model with no predictors.
- Suppose for node 3, the initial weights are $\theta_3 = -0.3$, $w_{13} = 0.05$, $w_{23} = 0.01$.
- With a logistic activation function $Output_j = \frac{1}{1+e^{-(\theta_j + \sum_{i=1}^{p} w_{ij} x_i)}}$.
- Thus, $Output_3 = 1/(1 + e^{-(-0.3+(0.05)(0.2)+(0.01)(0.9))}) = 0.43$.



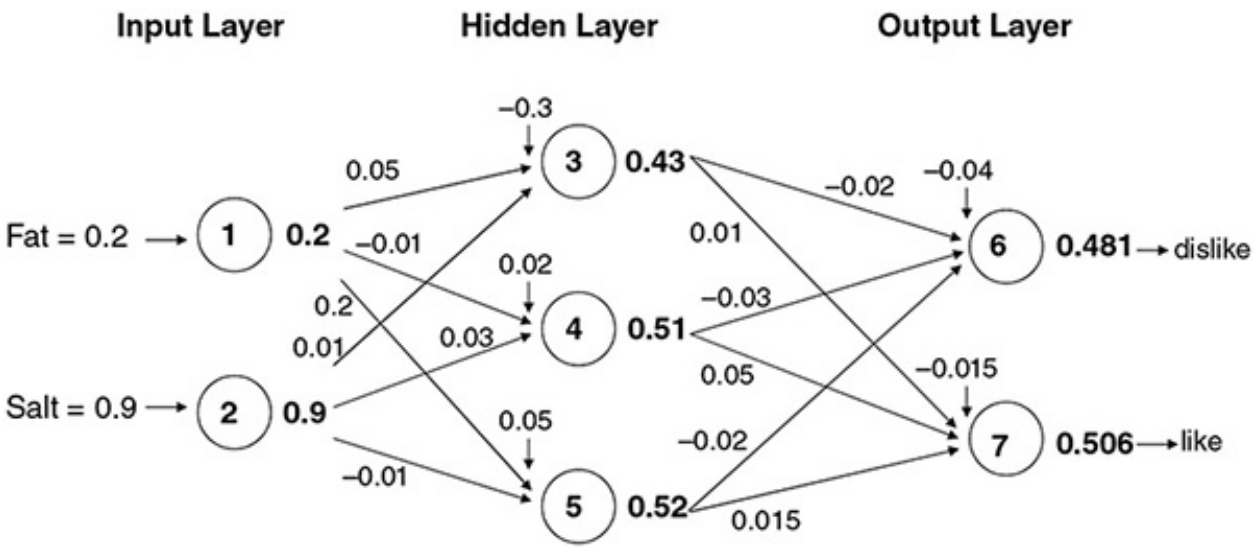
Your turn #1

- Find the output for hidden layer node 5.
- Recall: with a logistic activation function $Output_j = \frac{1}{1+e^{-(\theta_j + \sum_{i=1}^{p} w_{ij} x_i)}}$



Your turn #1

- Find the output for hidden layer node 5.
- Recall: with a logistic activation function $Output_j = \frac{1}{1+e^{-(\theta_j + \sum_{i=1}^{p} w_{ij} x_i)}}$
- $x_1 = 0.2, x_2 = 0.9$
- For node 5, the initial weights are $\theta_5 = 0.05$, $w_{15} = 0.2$, $w_{25} = -0.01$
- Thus, $Output_5 = 1/(1 + e^{-(0.05+(0.2)(0.2)+(-0.01)(0.9))}) = 0.52$

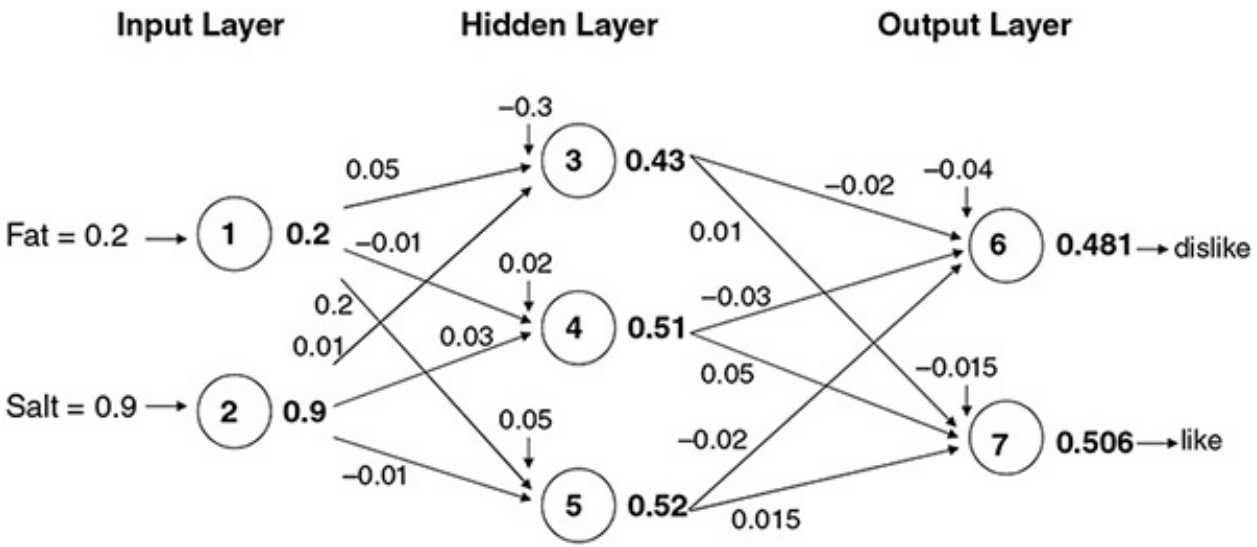


Example on tiny dataset: Hidden layer

- If there is more than one hidden layer, the same calculation applies, except that the input values for the second, third, and so on, hidden layers would be the output of the preceding hidden layer *i.e.* the number of input values into a certain node is equal to the number of nodes in the preceding layer.
- The use of the logistic activation function is driven by the fact that it has a squashing effect on very small or very large values but is almost linear in the range where the value of the function is between 0.1 and 0.9.

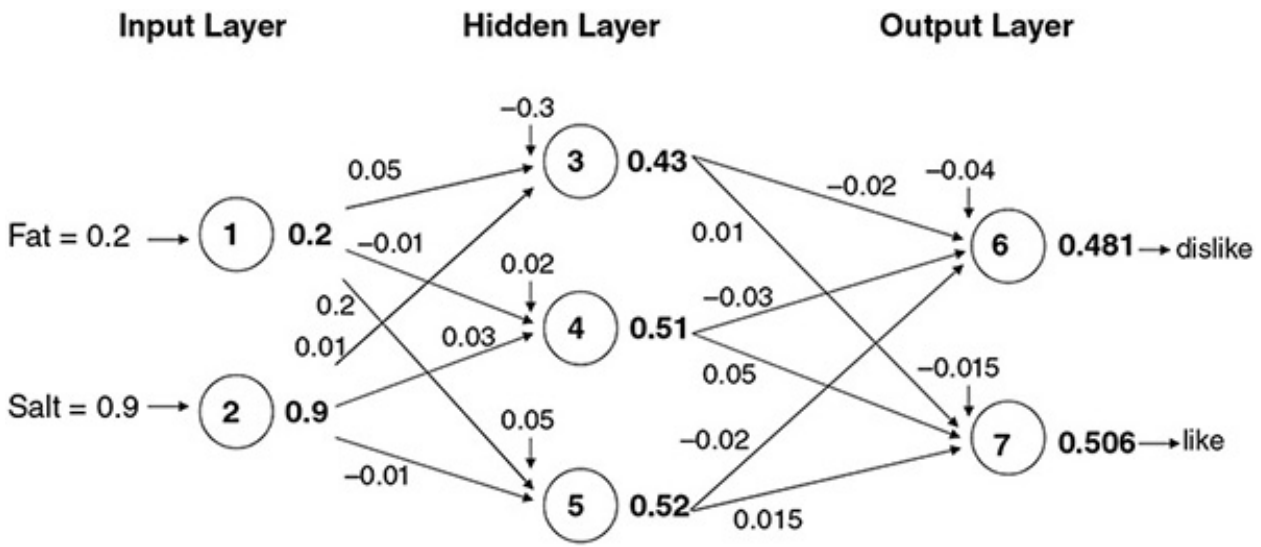
Example on tiny dataset: Output layer

- The output layer obtains input values from the (last) hidden layer.
- It uses the same function described before (weighted sum of its input values and then applies the function g) to create the output i.e. $Output_j = \frac{1}{1+e^{-(\theta_j+\sum_{i=1}^{p} w_{ij}x_i)}}$.
- $Output_6 = 1/(1+e^{-(-0.04+(-0.02)(0.43)+(-0.03)(0.51)+(-0.02)(0.52))}) = 0.481.$
- $Output_7 = 1/(1+e^{-(-0.015+(0.01)(0.43)+(0.05)(0.51)+(0.015)(0.52))}) = 0.506.$



Example on tiny dataset: Output layer

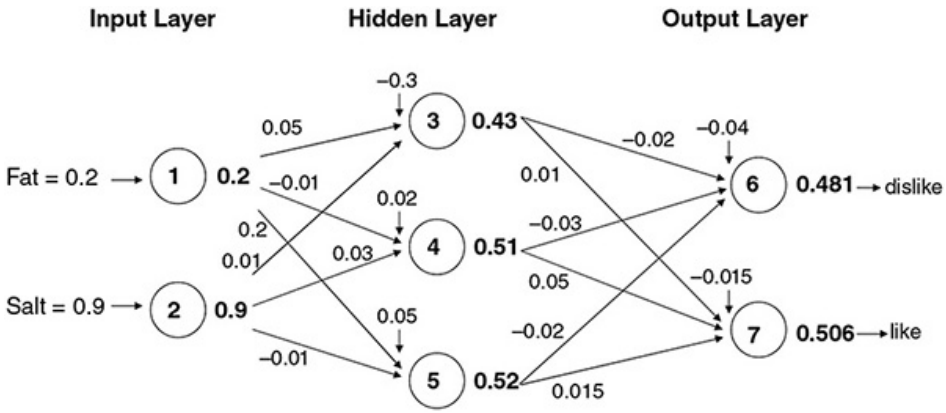
- The last step involves normalizing these two values so that they add up to 1.
- $P(Y = \textit{dislike}) = \textit{Output}_6/(\textit{Output}_6 + \textit{Output}_7) = 0.481/(0.481 + 0.506) = 0.49$.
- $P(Y = \textit{like}) = \textit{Output}_7/(\textit{Output}_6 + \textit{Output}_7) = 0.506/(0.481 + 0.506) = 0.51$.
- For a binary classification problem, use a cutoff value (*e.g.* 0.5) to assign labels.
- For applications with m classes ($m > 2$), there are m output nodes and label of the output node with the largest value gets assigned.
- Note: for a regression problem, there is only one output node.



How to modify weights? Example on tiny dataset: compute error

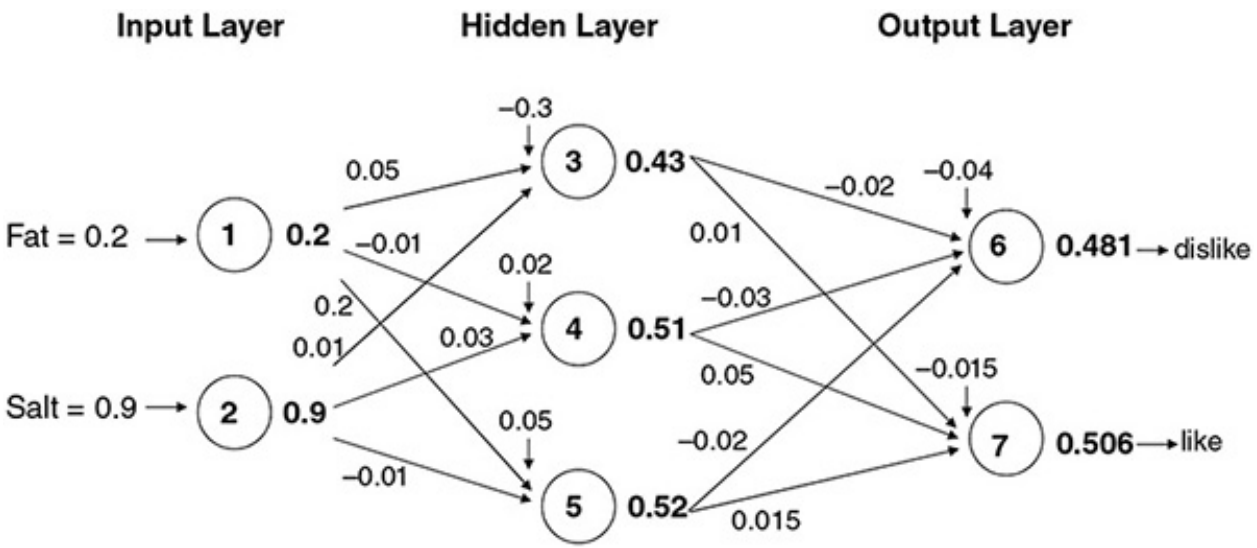
- Repeat the process for all records in the training set.
- For each record, compare the model prediction to the actual outcome value.
- Let $\hat{y}_k$ be the output from node k (*e.g.* $\hat{y}_6 = 0.481$ and $\hat{y}_7 = 0.506$).
- Set y_k as 1 or 0 depending in whether the actual class of the observation coincides with the label of output node k or not.
- For observation 1, response variable label is "like". Hence, $y_6 = 0$ and $y_7 = 1$.

Obs	Fat Score	Salt Score	Acceptance
1	0.2	0.9	like
2	0.1	0.1	dislike
3	0.2	0.4	dislike
4	0.2	0.5	dislike
5	0.4	0.5	like
6	0.3	0.8	like



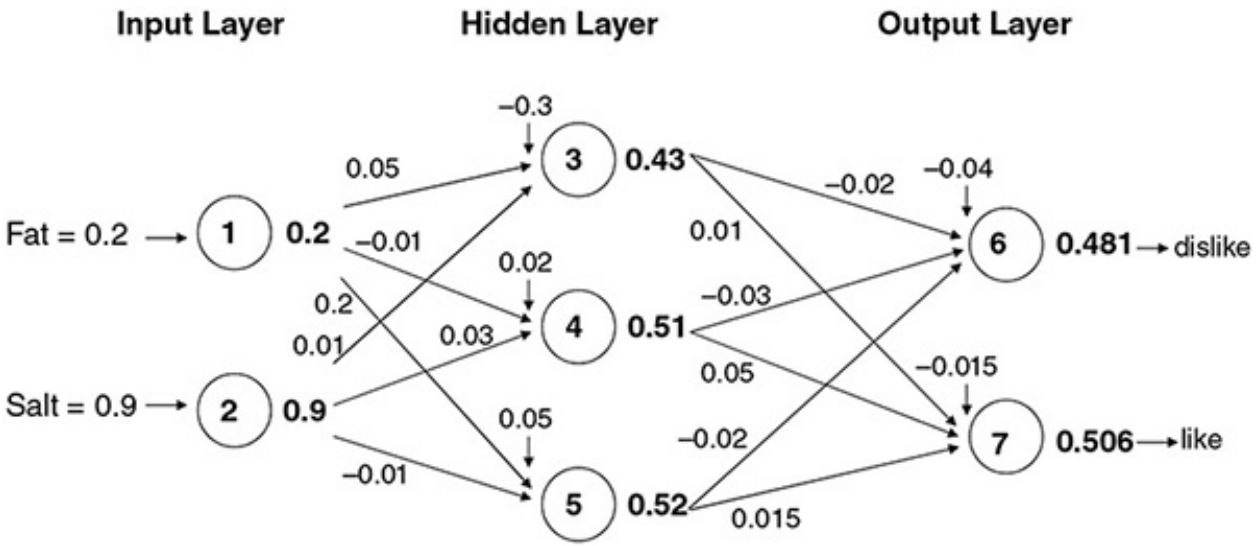
How to modify weights? Example on tiny dataset: compute error for output node

- Idea: back propagation: errors are computed from the last layer (the output layer) back to the hidden layers.
- The error associated to output node k is given by $err_k = \hat{y}_k(1 - \hat{y}_k)(y_k - \hat{y}_k)$.
- For obs 1, we have $\hat{y}_6 = 0.481, \hat{y}_7 = 0.506$ and $y_6 = 0, y_7 = 1$.
- $err_6 = 0.481(1 - 0.481)(0 - 0.481) = -0.120$.
- $err_7 = 0.506(1 - 0.506)(1 - 0.506) = 0.123$.



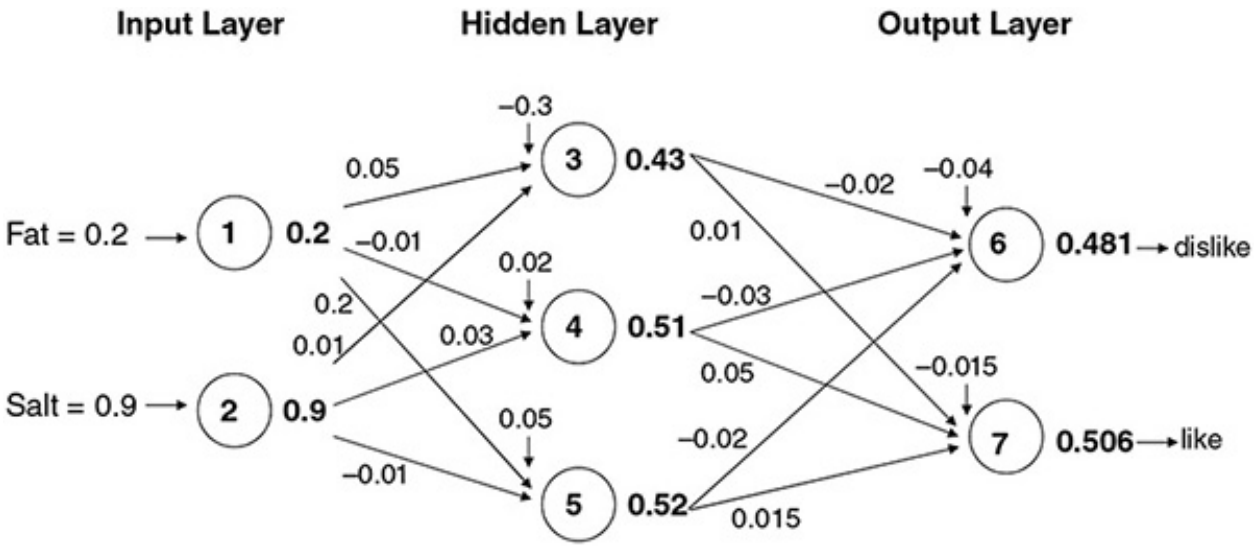
How to modify weights? Example on tiny dataset: compute error for hidden node

- The error associated to hidden node h is given by
  $err_h = \hat{y}_h(1-\hat{y}_h) \sum_{k \in downstream(h)} w_{hk} err_k$
- For obs 1, we have $\hat{y}_3 = 0.43, \hat{y}_4 = 0.51, \hat{y}_5 = 0.52$, $err_6 = -0.120, err_7 = 0.123$
- $err_3 = 0.43(1-0.43)(w_{36} err_6 + w_{37} err_7) =$
  $0.43(1-0.43)((-0.02)(-0.120) + (0.01)(0.123)) = 0.008$



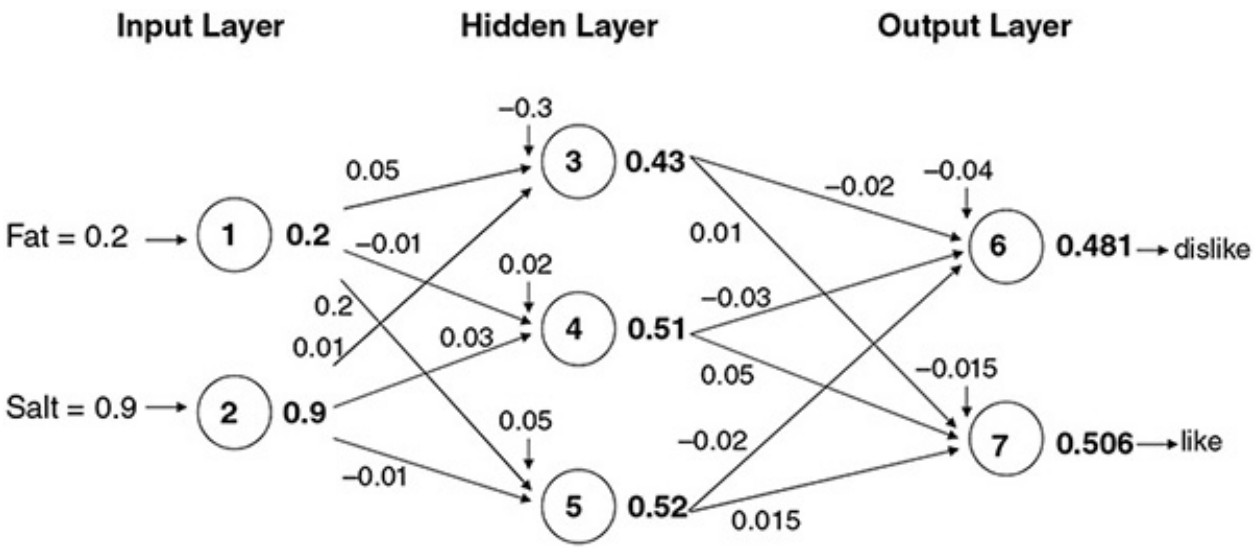
Your turn #2

- The error associated to hidden node h is given by $err_h = \hat{y}_h(1 - \hat{y}_h) \sum_{k \in downstream(h)} w_{hk} err_k$.
- For obs 1, we have $\hat{y}_3 = 0.43, \hat{y}_4 = 0.51, \hat{y}_5 = 0.52$, $err_6 = -0.120, err_7 = 0.123$.
- Compute err_5



Your turn #2

- The error associated to hidden node h is given by
  $err_h = \hat{y}_h(1 - \hat{y}_h) \sum_{k \in downstream(h)} w_{hk} err_k$
- For obs 1, we have $\hat{y}_3 = 0.43, \hat{y}_4 = 0.51, \hat{y}_5 = 0.52, err_6 = -0.120, err_7 = 0.123$
- $err_5 = 0.52(1 - 0.52)(w_{56} err_6 + w_{57} err_7) =$
  $0.52(1 - 0.52)((-0.02)(-0.120) + (0.015)(0.123)) = 0.001$

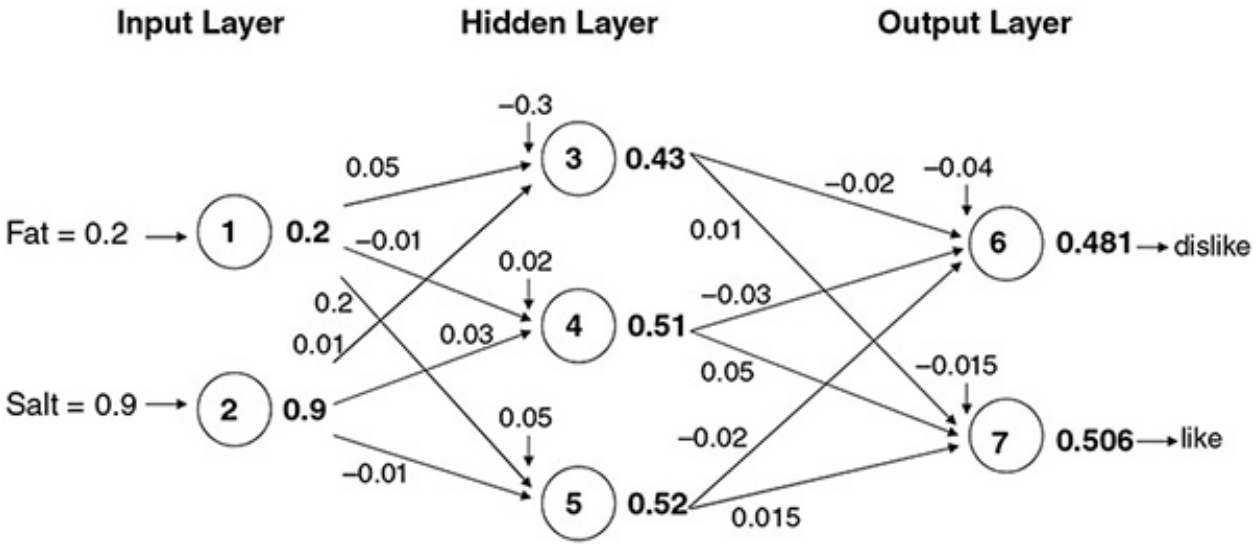


Training the model

- Idea: Iteratively estimate weights θ_j and w_{ij} that lead to the best predictive result.
- Update weights as the following:
  - $\theta_j^{new} = \theta_j^{old} + l \times err_j$
  - $w_{ij}^{new} = w_{ij}^{old} + l \times err_j$
  - l is a learning rate or weight decay parameter, a constant ranging typically between 0 and 1, which controls the change in weights from one iteration to the next.
- Two methods for updating the weights:
  - Case updating: weights are updated after each record is run through the network.
  - Batch updating: the entire training set is run through the network before each updating of weights take place. In this case the errors err_k in the updating equation is the sum of the errors from all records.
  - Case updating tends to yeild more accurate results than batch updating, but requires a longer run time.

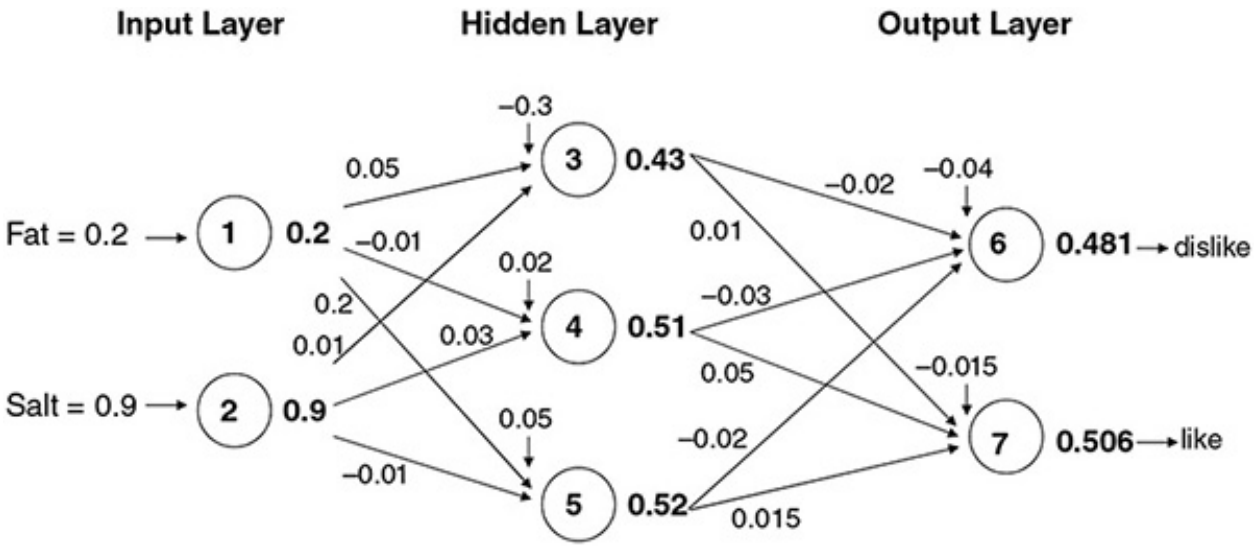
Example on tiny dataset: update weight

- $\theta_j^{new} = \theta_j^{old} + l \times err_j$
- $w_{ij}^{new} = w_{ij}^{old} + l \times err_j$
- Set $l = 0.5$
- $err_6 = -0.120, err_7 = 0.123$
- $\theta_7^{new} = -0.015 + (0.5)(0.123) = 0.047$
- $w_{37}^{new} = 0.01 + (0.5)(0.123) = 0.072$
- $w_{47}^{new} = 0.05 + (0.5)(0.123) = 0.112$
- $w_{57}^{new} = 0.015 + (0.5)(0.123) = 0.077$



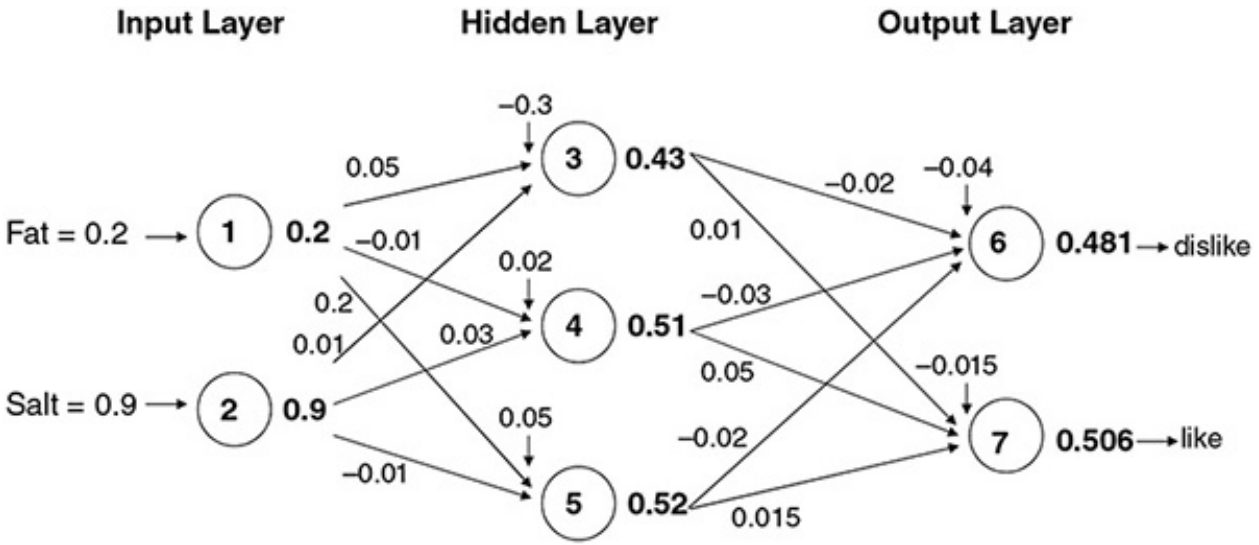
Your turn #3

- $\theta_j^{new} = \theta_j^{old} + l \times err_j$
- $w_{ij}^{new} = w_{ij}^{old} + l \times err_j$
- Set $l = 0.5$
- $err_6 = -0.120, err_7 = 0.123$
- Update the weights related to node 6.



Your turn #3

- $\theta_j^{new} = \theta_j^{old} + l \times err_j$
- $w_{ij}^{new} = w_{ij}^{old} + l \times err_j$
- Set $l = 0.5$
- $err_6 = -0.120, err_7 = 0.123$
- $\theta_6^{new} = -0.04 + (0.5)(-0.120)$
- $w_{36}^{new} = -0.02 + (0.5)(-0.120)$
- $w_{46}^{new} = -0.03 + (0.5)(-0.120)$
- $w_{56}^{new} = -0.02 + (0.5)(-0.120)$



Training the model: When does the updating stop?

- Stop updating when
  - the new weights are only slightly different from those of the preceding iteration.
  - the misclassification rate reaches a required threshold.
  - the limit on the number of runs is reached.

Preprocessing the data

- When using a logistic activation function, neural networks perform best when the predictors and outcome variable are on a scale of [0,1]
- Numerical variable:
  - Suppose X takes values in the range $[a, b]$
  - Normalization: $X_{norm} = \frac{X-a}{b-a}$
- Ordinal variable
  - Suppose X has m categories
  - Normalization: X_{norm} should take values $[0, 1/(m-1), 2/(m-2), \ldots, 1]$
  - Example: If X has 5 categories, it can be mapped into $[0, 0.25, 0.5, 0.75, 1]$
- Nominal variable
  - Suppose X has m categories
  - Normalization: transform into $m-1$ dummy variables

Relationship with linear regression

- Consider a neural network with a single output node and no hidden layers.
- For a data set with p predictors, the output is $g(\theta + \sum_{i=1}^{p} w_i x_i)$.
- Consider g as the identity function *i.e.* $g(s) = s$.
- Output $\hat{Y} = \theta + \sum_{i=1}^{p} w_i x_i$.
- This form is equivalent to the formulation of multiple linear regression.
- A neural network with no hidden layers, a single output node, and an identity function g searches only for linear relationships between the outcome and the predictors.

Relationship with logistic regression

- Consider a neural network with a single output node and no hidden layers.
- For a data set with p predictors, the output is $g(\theta + \sum_{i=1}^{p} w_i x_i)$.
- Consider a binary output variable Y.
- Consider g as the logistic function.
- Output $\hat{P}(Y = 1) = \frac{1}{1+e^{-(\theta+\sum_{i=1}^{p} w_i x_i)}}$.
- This form is equivalent to the formulation of logistic regression.

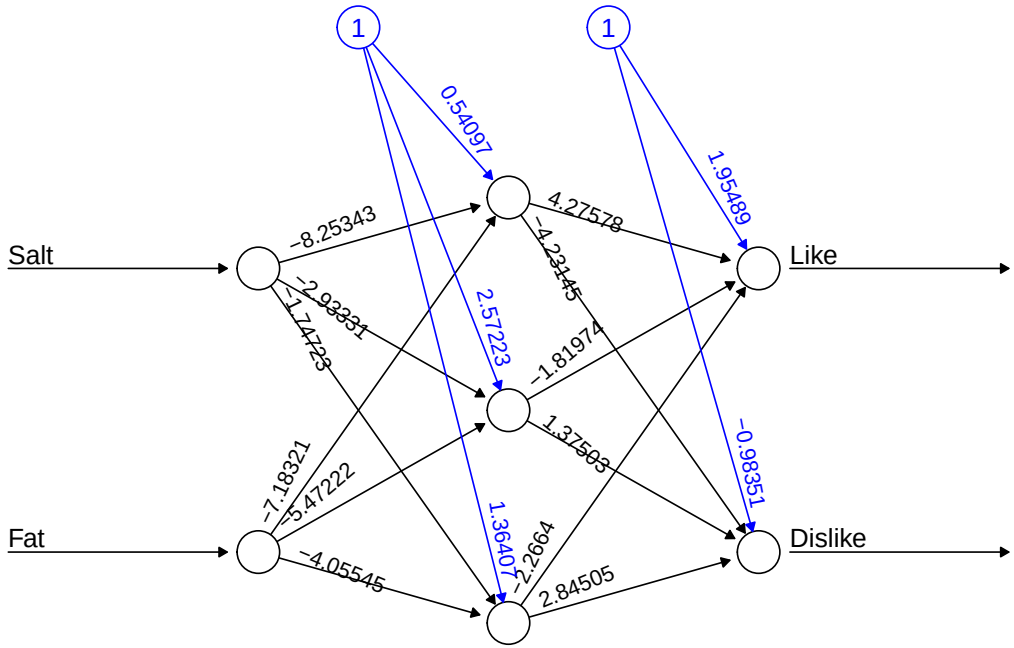
Example on tiny dataset

Obs	Fat Score	Salt Score	Acceptance
1	0.2	0.9	like
2	0.1	0.1	dislike
3	0.2	0.4	dislike
4	0.2	0.5	dislike
5	0.4	0.5	like
6	0.3	0.8	like

```
### Create tiny.data dataframe
Fat <- c(0.2, 0.1, 0.2, 0.2, 0.4, 0.3)
Salt <- c(0.9, 0.1, 0.4, 0.5, 0.5, 0.8)
Y <- c("like", "dislike", "dislike", "dislike", "like","like")
tiny.data <- data.frame(Fat, Salt, Y)

### create dummies for response variable
table(tiny.data$Y)
tiny.data$Like <- ifelse(tiny.data$Y == "like", 1, 0)
tiny.data$Dislike <- ifelse(tiny.data$Y == "dislike", 1, 0)

### get rid of the original response variable (only keep dummies)
tiny.data <- tiny.data[, -3]

library(neuralnet)
### Run neural net with 3 hidden nodes
nn <- neuralnet(Like + Dislike ~ Salt + Fat,
                data = tiny.data, hidden = 3)
```

Example on tiny dataset

```
### display weights
> nn$weights
[[1]]
[[1]][[1]]
           [,1]       [,2]       [,3]
[1,]  0.5409746  2.572231  1.364074
[2,] -8.2534333 -2.933313 -1.747227
[3,] -7.1832099 -5.472220 -4.055454
[[1]][[2]]
          [,1]       [,2]
[1,]  1.954890 -0.983507
[2,]  4.275781 -4.231448
[3,] -1.819743  1.375027
[4,] -2.266403  2.845047
### plot network
plot(nn, rep="best")
```

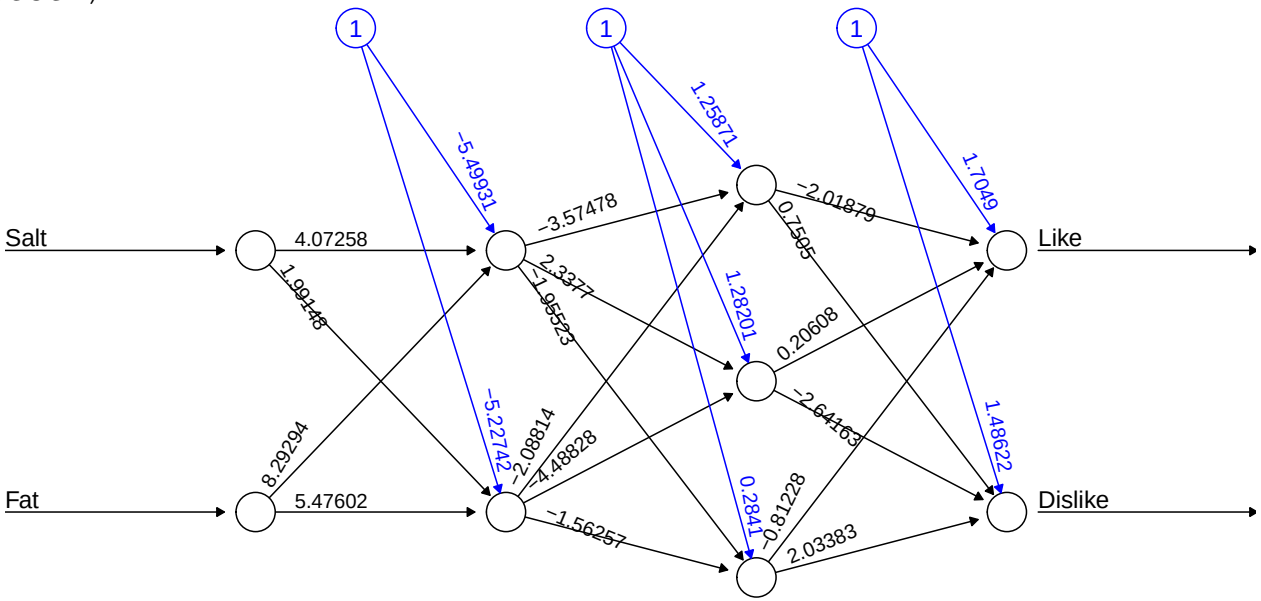


Example on tiny dataset

```
### display predictions
> nn$net.result
[[1]]
              [,1]         [,2]
[1,]  0.9214689965  0.09737271
[2,]  0.0005765927  1.00210090
[3,] -0.0797538001  1.06409623
[4,]  0.1148697788  0.87653963
[5,]  0.9487529132  0.05133784
[6,]  1.0960987848 -0.08271711

> nn$response
  Like Dislike
1    1       0
2    0       1
3    0       1
4    0       1
5    1       0
6    1       0
```

Example on tiny dataset: more than one hidden layer

```
### run neural network with 2 layers
### layer 1 has 2 hidden nodes, layer 2 has 3 hidden nodes
nn2 <- neuralnet(Like + Dislike ~ Salt + Fat,
                 data = tiny.data, hidden = c(2,3))
### plot network
plot(nn2, rep="best")
```



Error: 0.075926 Steps: 404

Example of neural network for classification problem

Use diabetes data

```
diabetes <- read.csv("E:/Data mining/datasets/diabetes.csv")
diabetes$Outcome <- as.factor(diabetes$Outcome)
levels(diabetes$Outcome) <- c("no", "yes")

## Include the functions required for data partitioning
source("E:/Data mining/Lecture Notes/myfunctions.R")

RNGkind (sample.kind = "Rounding")
set.seed(0)
### call the function for creating 70:30 partition
p2 <- partition.2(diabetes, 0.7)
training.data <- p2$data.train
test.data <- p2$data.test
```

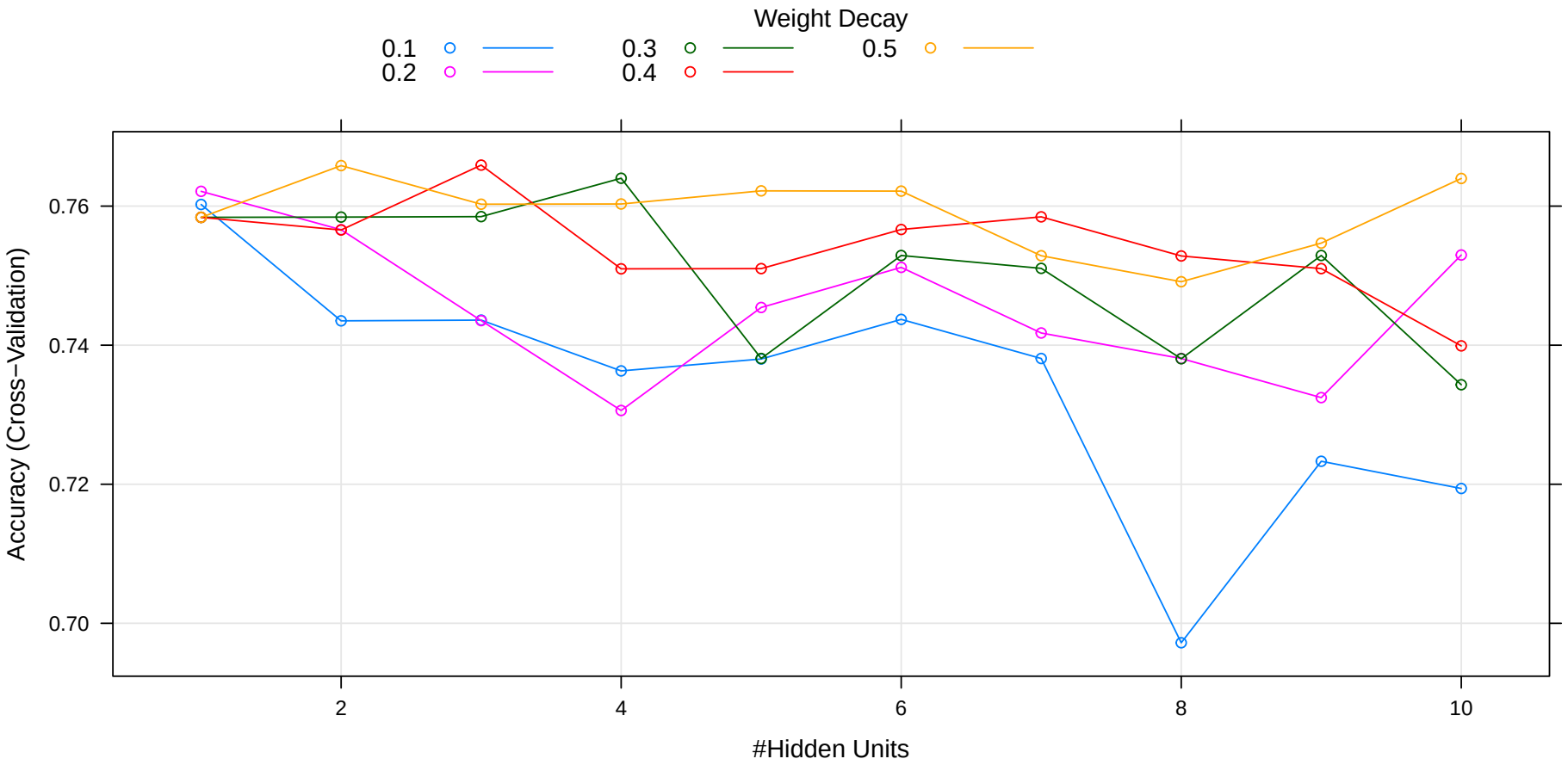
Example of neural network for classification problem

Neural net model on training data using cross validation

```
library(caret)
library(nnet)
train_control <- trainControl(method="cv", number=10)
tune.grid <- expand.grid(size = seq(from = 1, to = 10, by = 1),
                         decay = seq(from = 0.1, to = 0.5, by = 0.1))
cv.nn <- train(Outcome ~ . , data = training.data, method = "nnet",
               preProc = c("center", "scale"),
               trControl = train_control, tuneGrid = tune.grid)
# cv.nn <- train(Outcome ~ . , data = training.data, method = "nnet",
#                preProc = c("center", "scale"), metric = "Kappa",
#                trControl = train_control, tuneLength = 10)
print(cv.nn)
plot(cv.nn)

# Best size and decay
> cv.nn$bestTune
   size decay
14    3   0.4
```

Example of neural network for classification problem

Neural net model on training data using cross validation Optimal size (number of hidden nodes) = 3, optimal decay (learning rate) = 0.4



Example of neural network for classification problem

Prediction on test data

```
pred.prob <- predict(cv.nn, test.data, type = "prob")
pred.y.nn <- ifelse(pred.prob[,2] > 0.5, "yes", "no") # using cutoff = 0.5
> confusionMatrix(as.factor(pred.y.nn), as.factor(test.data$Outcome), positive = "yes")
Confusion Matrix and Statistics

          Reference
Prediction  no yes
       no  126  23
       yes  24  57

               Accuracy : 0.7957
                 95% CI : (0.7377, 0.8458)
    No Information Rate : 0.6522
    P-Value [Acc > NIR] : 1.383e-06
                  Kappa : 0.5509
 Mcnemar's Test P-Value : 1
            Sensitivity : 0.7125
            Specificity : 0.8400
         Pos Pred Value : 0.7037
         Neg Pred Value : 0.8456
             Prevalence : 0.3478
         Detection Rate : 0.2478
   Detection Prevalence : 0.3522
      Balanced Accuracy : 0.7762

       'Positive' Class : yes
```

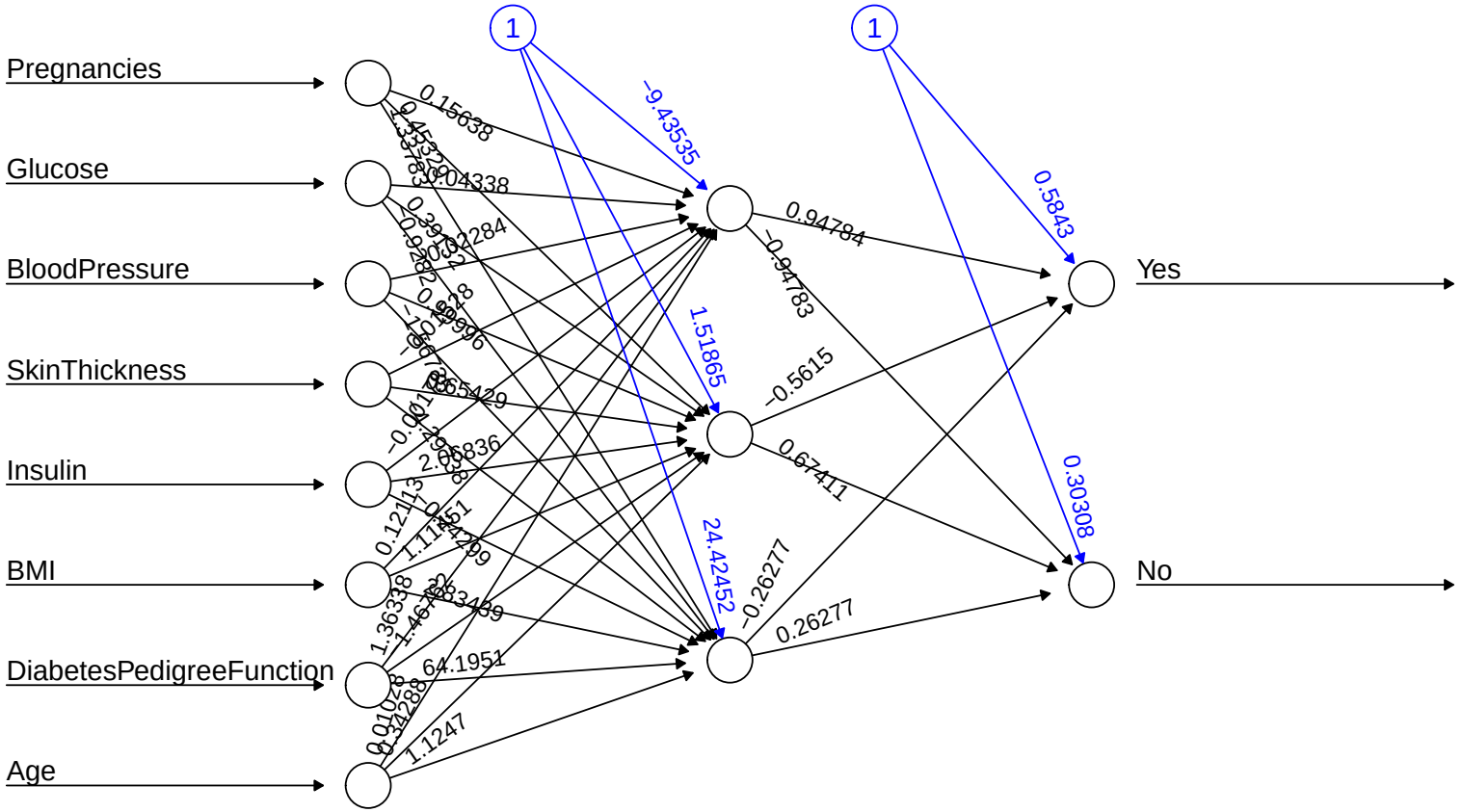
Example of neural network for classification problem

Plotting the neural network using neuralnet package

```
training.data.nn <- training.data
### create dummies for variables with multiple categories
training.data.nn$Yes <- ifelse(training.data.nn$Outcome == "yes", 1, 0)
training.data.nn$No <- ifelse(training.data.nn$Outcome == "no", 1, 0)

### get rid of the mutiple category variables (only keep dummies)
training.data.nn <- training.data.nn[, -9]

### fit neural net model with the bestTune parameters
nn <- neuralnet(Yes + No ~ ., data = training.data.nn,
                hidden = cv.nn$bestTune$size, learningrate = cv.nn$bestTune$decay)
### plot network
plot(nn, rep="best")
```

Example of neural network for classification problem

Neural net model on training data using cross validation Optimal size (number of hidden nodes) = 3, optimal decay (learning rate) = 0.4



Error: 80.97999 Steps: 13029

How to avoid overfitting?

- Neural net is prone to overfitting.
- Detect overfitting by examining the performance on validation sample.
- Overfitting can also be controlled using a simpler model (smaller number of layers and hidden nodes).
- Strategy:
  - The most popular choice for the number of hidden layer is one.
  - For the number of hidden nodes, a rule of thumb is to start with p (number of predictors) nodes and gradually decrease or increase while checking for overfitting.
  - In case more than one layer is required, the *caret* package *train*() function with $method = "mlpML"$ may be used.

Advantages and shortcomings of Neural Network

- Advantage:
  - Ability to capture highly complicated relationship between response and predictors.
  - High tolerance to noisy data.
- Shortcoming:
  - Difficult to interpret.
  - No built-in variable selection.
  - Requires a large number of records to obtain good results.
  - Computationally expensive.
  - Weights often lead to local optima rather than global optima i.e. weights converge to values that do not provide the best fit to training data.