Introduction to Neural Networks

Neural Networks

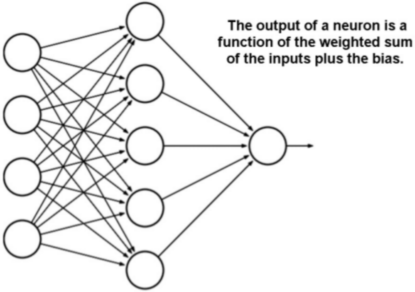
Cathy Poliak, Ph.D.
cpoliak@central.uh.edu

Department of Mathematics
University of Houston

How do neural netWORKs WORK?

Similar to the biological neuron structure, ANNs define the neuron as a:

"Central processing unit that performs a mathematical operation to generate output from a set of inputs."



Essentially, ANN is a set of mathematical function approximations.

Another Example in R

- Data collected at a restaurant through customer interviews. The data set is named RestuarntTips
- The customers were asked to give a score to the following aspects: Service, Ambience, and Food.
- Also were asked whether they would leave the tip on the basis of these scores, CustomerWillTip (Tip = 1 and No-tip = 0)

1. What type of problem is this?
   a) Classification problem
   b) Regression problem

R Code

Type and run the following in R

```
library(nnet)
#Import dataset ResturantTips
attach(RestaurantTips)
names(RestaurantTips)

#Train the model based on output from imput

model = nnet(CustomerWillTip ~ Service + Ambience + Food,
              data = RestaurantTips,
              size = 5,
              rang = 0.1,
              decay = 5e-2,
              maxit = 5000)
print(model)
```

This output processes the forward and backpropagation until convergence.

The Parameters in the `nnet()` Function

The parameters used in the `nnet()` function can be tuned to improve performance.

- Size : Number of units in the hidden layer
- Decay : Weight decay specifies regularization in the neural network. As a rule of thumb, the more training examples you have, the weaker this term should be. The more parameters you have the higher this term should be.
- Maxit : Maximum iterations, the default is 100.

rang = Initial random weights defualt is 0.7

Lab Questions

2. How many weights are there?

   a) 5
   b) 3
   c) 15
   d) 26

   $(3 \times 5 + 5) + 6$

3. Type and run the following

```
pred_nnet<-predict(model,RestaurantTips,type = "class")
(mtab<-table(RestaurantTips$CustomerWillTip,pred_nnet))
```

   What is the training error rate?

   a) 16.67%
   b) 50%
   c) 86.67%
   d) 4%

```
  pred_nnet
    0  1
0  12  3
1   2 13
```

$\frac{5}{30}$

Plots of the Model

4. Type and run the following `plotnet(model)`. How many nodes are in the hidden layer?

   a) 3
   b) 5
   c) 1
   d) 26

5. Type are run the following `garson(model)`, this is using the `NeuralNetTools`. Which input parameter has the greatest influence to give a tip?

   a) Ambience
   b) Food
   c) Service
   d) All of them are equal.

Step-by-step ANN training.

Let us analyze in detail, step by step, all the operations to be done for network training:

1. Initialize the weights w^0 and biases b^0 with random values (one time).
2. Repeat the steps 3 to 5 for each training pattern (presented in random order), until the error is minimized or a stopping criteria is reached.
3. Conduct forward propagation for ANN with weights w^0 and biases b^0:

   $$\text{input layer} \rightarrow \text{hidden layer(s)} \rightarrow \text{output layer}$$

4. Conduct backpropagation:

   $$\text{compute error} \rightarrow \text{take derivative} \rightarrow \text{adjust weights/biases}$$

5. Set w^0 and b^0 equal to adjusted weights/biases, back to step 3.

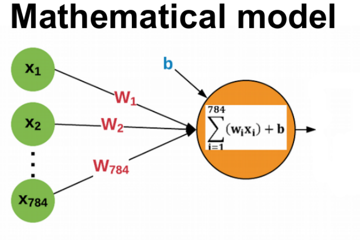
The complete pass back and forth is called a training cycle or epoch. The updated weights and biases are used in the next cycle. We keep recursively training until the error is very minimal.

Training ANN: Minimizing the Error

We desire to adjust the weight and bias parameters such that to **minimize** the error $\hat{y} - y =$ predicted output $-$ true output.

Example **Single Neuron Model:**

- p nodes $x_1, \ldots, x_p$ in the input layer,
- one output node y, calculated as $\hat{y} = b + \sum_{j=1}^{p} w_j x_j$



Presume we are supplied with:

- n data samples $\mathbf{x}_i = (x_{1,i}, \ldots, x_{p,i}), i = 1, \ldots, n$
- n corresponding true responses (or labels) $y_i, i = 1, \ldots, n$

Training ANN: Minimizing the Error

Feed the whole data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ to our ANN at once. Then the total error formula is

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}[y_i - (b + \sum_{j=1}^{p} w_j x_{j,i})]^2 \equiv f_{err}(b, w_1, \ldots, w_p)$$

and we need to minimize it with respect to $b, w_1, \ldots, w_p$

Training ANN: Minimizing the Error

Feed the whole data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ to our ANN at once. Then the total error formula is

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}[y_i - (b + \sum_{j=1}^{p} w_j x_{j,i})]^2 \equiv f_{err}(b, w_1, \ldots, w_p)$$

and we need to minimize it with respect to $b, w_1, \ldots, w_p$

$$\min_{b,w_1,\ldots,w_p} \sum_{i=1}^{n}[y_i - (b + \sum_{j=1}^{p} w_j x_{j,i})]^2 = \min_{b,w_1,\ldots,w_p} f_{err}(b, w_1, \ldots, w_p) \quad (1)$$

Similar to the **least squares regression**:

- $\hat{y}_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_{j,i}$
- $\min_{\beta_0,\ldots,\beta_p} \sum_{i=1}^{n}[y_i - (\beta_0 + \sum_{j=1}^{p} \beta_j x_{j,i})]^2 = \min_{\beta_0,\ldots,\beta_p} f_{err}(\beta_0, \beta_1, \ldots, \beta_p)$

which we could solve **analytically** via $\Delta_{\boldsymbol{\beta}=(\beta_0,\ldots,\beta_p)} f(\boldsymbol{\beta}) \equiv 0$.

Training ANN: Minimizing the Error

Likewise for the Linear Neuron Model, in order to obtain optimal values $b, w_1, \ldots, w_p$ that minimize (1), we **could** analytically solve:

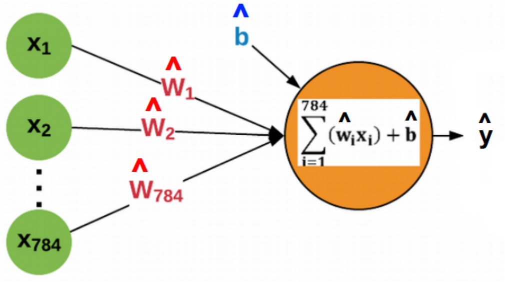
$$\Delta_{\boldsymbol{w}=(b,w_1,\ldots,w_p)} f(\boldsymbol{w}) \equiv 0$$

Training ANN: Minimizing the Error

Likewise for the Linear Neuron Model, in order to obtain optimal values $b, w_1, \ldots, w_p$ that minimize (1), we **could** analytically solve:

$$\Delta_{\boldsymbol{w}=(b,w_1,\ldots,w_p)} f(\boldsymbol{w}) \equiv 0$$

$$\Longrightarrow \ \hat{\boldsymbol{w}} = (\hat{b}, \hat{w}_1, \ldots, \hat{w}_p) = \underset{b,w_1,\ldots,w_p}{argmin} \sum_{i=1}^{n} [y_i - (b + \sum_{j=1}^{p} w_j x_{j,i})]^2$$



Training ANN: Gradient Descent

While such simple case as Linear Neuron Model could be solved analytically (read "one could calculate an explicit formula for weight and bias estimates"), there are multiple aspects to note:

1. We'd like to avoid solving large systems of equations analytically.
2. We want a method that real neurons are likely using. Hint: they're probably not analytically solving any equations.

Training ANN: Gradient Descent

While such simple case as Linear Neuron Model could be solved analytically (read "one could calculate an explicit formula for weight and bias estimates"), there are multiple aspects to note:

1. We'd like to avoid solving large systems of equations analytically.
2. We want a method that real neurons are likely using. Hint: they're probably not analytically solving any equations.
3. We want a method that can be generalized to **multi**-layer, **non**-linear neural networks, for most of which the closed-form analytical solutions (read "explicit formulas") simply won't be available.

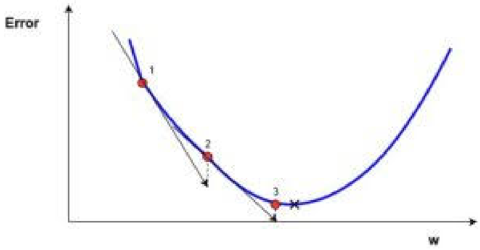
That's where such numerical optimization technique as **gradient descent** comes in extremely handy.

Gradient Descent.

Gradient descent is an optimization approach, which entails using:

- Partial derivatives (gradients) of a function, and
- a step size parameter,

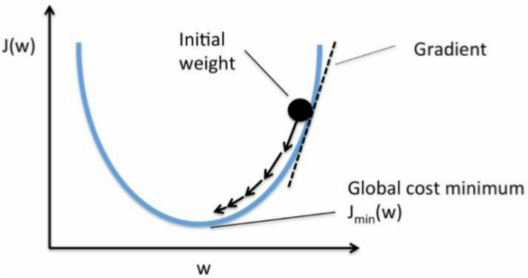
in order to calculate the minimum (or maximum) of **that** function.



We move by the direction of steepest descent.

Gradient Descent for ANN

For neural networks, the gradient descent approach is used when iterating the **updates** of weights and biases.



Global cost function in our case is the squared prediction error,

$$\frac{1}{2}(\hat{y} - y)^2, \quad \text{or} \quad \frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

which we are trying to minimize.

Gradient Descent: online- and batch-learning

Questions about weight updates:

- Which function to optimize, $\frac{1}{2}(\hat{y} - y)^2$? Or is it $\frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$?
- How often do we need to update the parameter estimates?

Gradient Descent: online- and batch-learning

Questions about weight updates:

- Which function to optimize, $\frac{1}{2}(\hat{y} - y)^2$? Or is it $\frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$?
- How often do we need to update the parameter estimates?

Given data $(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^n, y^n)$, where $\mathbf{x}^i = (x_1^i, \ldots, x_p^i)$, gradient descent can be performed for

- **online**-learning - ANN is fed one training sample $(\mathbf{x}^i, y^i)$ at a time) $\implies$ weights are updated each time by minimizing $\frac{1}{2}(\hat{y}^i - y^i)^2$.

Gradient Descent: online- and batch-learning

Questions about weight updates:

- Which function to optimize, $\frac{1}{2}(\hat{y} - y)^2$? Or is it $\frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$?
- How often do we need to update the parameter estimates?

Given data $(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^n, y^n)$, where $\mathbf{x}^i = (x_1^i, \ldots, x_p^i)$, gradient descent can be performed for

- **online**-learning - ANN is fed one training sample $(\mathbf{x}^i, y^i)$ at a time) $\Longrightarrow$ weights are updated each time by minimizing $\frac{1}{2}(\hat{y}^i - y^i)^2$.
- **full-batch** - ANN is fed full training set all at once $\Longrightarrow$ weights are updated once all $\hat{y}^i$ got calculated, minimize $\frac{1}{2}\sum_{i=1}^{n}(\hat{y}^i - y^i)^2$.

Gradient Descent: online- and batch-learning

Questions about weight updates:

- Which function to optimize, $\frac{1}{2}(\hat{y} - y)^2$? Or is it $\frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$?
- How often do we need to update the parameter estimates?

Given data $(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^n, y^n)$, where $\mathbf{x}^i = (x_1^i, \ldots, x_p^i)$, gradient descent can be performed for

- **online**-learning - ANN is fed one training sample $(\mathbf{x}^i, y^i)$ at a time) $\Longrightarrow$ weights are updated each time by minimizing $\frac{1}{2}(\hat{y}^i - y^i)^2$.
- **full-batch** - ANN is fed full training set all at once $\Longrightarrow$ weights are updated once all $\hat{y}^i$ got calculated, minimize $\frac{1}{2}\sum_{i=1}^{n}(\hat{y}^i - y^i)^2$.
- **mini-batches** - ANN is iteratively fed subsets of training data $\Longrightarrow$ weights are updated once all $\hat{y}^i \in$ subset got calculated, minimize $\frac{1}{2}\sum_{i \in subset}(\hat{y}^i - y^i)^2$

Gradient Descent: online- and batch-learning

For either of three approaches (online, full- or mini-batch),

1. The weights & biases are randomly initialized at first.

Gradient Descent: online- and batch-learning

For either of three approaches (online, full- or mini-batch),

1. The weights & biases are randomly initialized at first.
2. They get updated each time after:
   a) Data is fed to ANN (be it single sample, full- or mini-batch),
   b) Forward propagation is performed to get $\hat{y}_i$'s,
   c) Error function is calculated (be it single $\frac{1}{2}(\hat{y} - y)$ or $\frac{1}{2}\sum_i(\hat{y}_i - y_i)^2$)
   d) Weights are updated via gradient descent.

Gradient Descent: online- and batch-learning

For either of three approaches (online, full- or mini-batch),

1. The weights & biases are randomly initialized at first.
2. They get updated each time after:
   a) Data is fed to ANN (be it single sample, full- or mini-batch),
   b) Forward propagation is performed to get $\hat{y}_i$'s,
   c) Error function is calculated (be it single $\frac{1}{2}(\hat{y} - y)$ or $\frac{1}{2}\sum_i(\hat{y}_i - y_i)^2$)
   d) Weights are updated via gradient descent.
3. The step 2 is iterated until all training observations were used, with updated weights being used for next iteration.

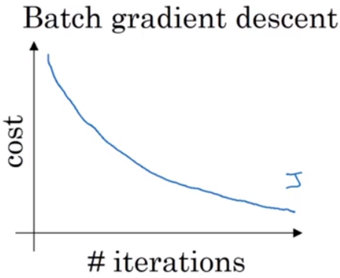
Gradient Descent: online- and batch-learning

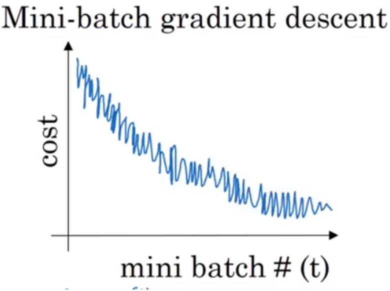
For either of three approaches (online, full- or mini-batch),

1. The weights & biases are randomly initialized at first.
2. They get updated each time after:
   a) Data is fed to ANN (be it single sample, full- or mini-batch),
   b) Forward propagation is performed to get $\hat{y}_i$'s,
   c) Error function is calculated (be it single $\frac{1}{2}(\hat{y} - y)$ or $\frac{1}{2}\sum_i(\hat{y}_i - y_i)^2$)
   d) Weights are updated via gradient descent.
3. The step 2 is iterated until all training observations were used, with updated weights being used for next iteration.
4. Steps 2 & 3 formulate an **epoch** - a single pass through the whole training set. ANNs are trained over many epochs, with each epoch potentially containing multiple iterations (bar the full-batch approach).

Gradient Descent: full- vs and mini-batch example

Example. Below is an exemplary progression for a full-batch gradient descent (left) as opposed to mini-batch (right). Cost here is the error function, $\frac{1}{2}\sum_i(\hat{y}_i - y_i)$.





Changes of error function are

- fewer and smoother for full-batch,
- more frequent and bumpier for mini-batch.

Iterative Approach: Example

An example to illustrate the iterative method of online gradient descent:

- Each day you get lunch at the cafeteria.
  - Your diet consists of fish, chips, and ketchup.
  - You get several portions (inputs x) of each $\Rightarrow x_{fish}, x_{chips}, x_{ketchup}$.

Iterative Approach: Example

An example to illustrate the iterative method of online gradient descent:

- Each day you get lunch at the cafeteria.
  - Your diet consists of fish, chips, and ketchup.
  - You get several portions (inputs x) of each $\Rightarrow x_{fish}, x_{chips}, x_{ketchup}$.
- Cashier only tells you the total price of the meal (response y).
  - After several days, you should be able to figure out the price (weight w) of each portion $\Rightarrow w_{fish}, w_{chips}, w_{ketchup}$

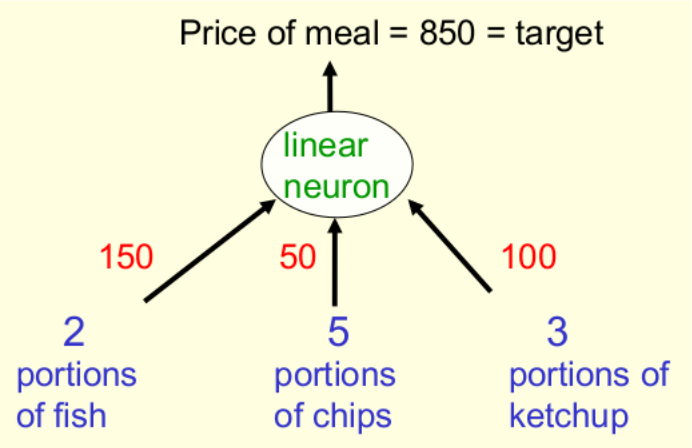
Iterative Approach: Example

An example to illustrate the iterative method of online gradient descent:

- Each day you get lunch at the cafeteria.
  - Your diet consists of fish, chips, and ketchup.
  - You get several portions (inputs x) of each $\Rightarrow x_{fish}, x_{chips}, x_{ketchup}$.
- Cashier only tells you the total price of the meal (response y).
  - After several days, you should be able to figure out the price (weight w) of each portion $\Rightarrow w_{fish}, w_{chips}, w_{ketchup}$
- The iterative approach:
  1. Start with random guesses for the prices ($\Leftrightarrow$ weights), and then
  2. Adjust them to get a better fit to the observed prices of whole meals (y_i's).

Iterative Approach: Example

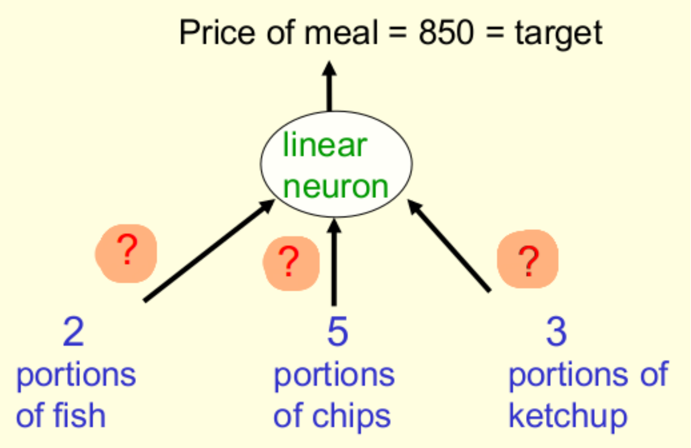
The true weights used by the cashier



$2(150) + 5(50) + 3(100) = 850$

Iterative Approach: Example

In reality? True weights are unknown.



Iterative Approach: Example

- The portion prices are weights of our linear neuron.

$$\mathbf{w} = (w_{fish}, w_{chips}, w_{ketchup})$$

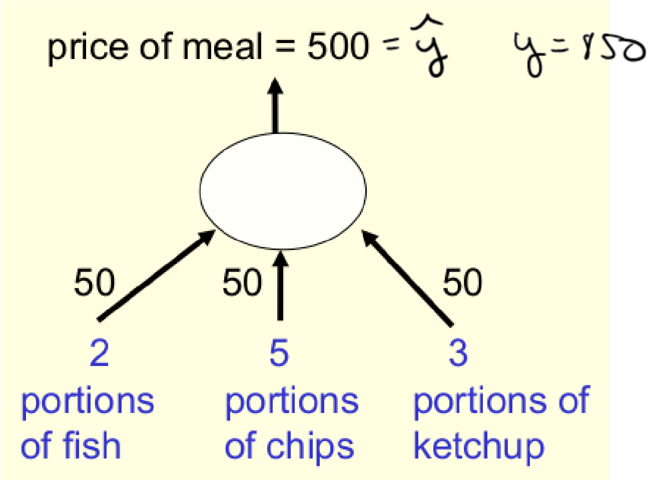
- Each meal price is our response y, and it is calculated via linear combination of portion prices (weights w) multiplied by # of portions (inputs x):

$$price = x_{fish}w_{fish} + x_{chips}w_{chips} + x_{ketchup}w_{ketchup}$$

- We start with random guesses for the weights, and then adjust the guesses slightly to give a better fit to the prices given by the cashier.

Iterative Approach: Example

Initializing the weights with random values, e.g $50, 50, 50$**.**



Iterative Approach: Example

- True value $y = 850$; predicted value $\hat{y} = 500 \implies$

$$\text{Residual error} = 850 - 500 = 350$$

Iterative Approach: Example

- True value $y = 850$; predicted value $\hat{y} = 500 \implies$

$$\text{Residual error} = 850 - 500 = 350$$

- Below we provide a **delta-rule** update for gradient descent algorithm given a learning rate α:

$$\Delta w_i = \alpha \times \frac{\partial}{\partial w_i}\left[\frac{1}{2}(\hat{y} - y)^2\right], \; i = 1, 2, 3$$

$$\frac{\partial}{\partial w_1}\left[\frac{1}{2}\left(y - (x_1 w_1 + x_2 w_2 + x_3 w_3)\right)^2\right]$$

$$= -x_1\left(y - (x_1 w_1 + x_2 w_2 + x_3 w_3)\right)$$

$$= -x_1(y - \hat{y})$$

Iterative Approach: Example

- True value $y = 850$; predicted value $\hat{y} = 500 \implies$

$$\text{Residual error} = 850 - 500 = 350$$

- Below we provide a **delta-rule** update for gradient descent algorithm given a learning rate α:

$$\Delta w_i = \alpha \times \frac{\partial}{\partial w_i}\left[\frac{1}{2}(\hat{y} - y)^2\right],\ i = 1, 2, 3$$

where

$$\frac{\partial}{\partial w_i}\left[\frac{1}{2}(y - \hat{y})^2\right] = \frac{\partial}{\partial w_i}\left[\frac{1}{2}(y - \sum_i w_i x_i)^2\right] = \cdots = -x_i(y - \hat{y})$$

Iterative Approach: Example

- True value $y = 850$; predicted value $\hat{y} = 500 \implies$

$$\text{Residual error} = 850 - 500 = 350$$

- Below we provide a **delta-rule** update for gradient descent algorithm given a learning rate α:

$$\Delta w_i = \alpha \times \frac{\partial}{\partial w_i}\left[\frac{1}{2}(\hat{y} - y)^2\right],\ i = 1, 2, 3$$

where

$$\frac{\partial}{\partial w_i}\left[\frac{1}{2}(y - \hat{y})^2\right] = \frac{\partial}{\partial w_i}\left[\frac{1}{2}(y - \sum_i w_i x_i)^2\right] = \cdots = -x_i(y - \hat{y})$$

- Hence, the eventual weight updates are:

$$w_i^{upd} = w_i - \Delta w_i = w_i + \alpha \times x_i(y - \hat{y}),\quad i = 1, 2, 3$$

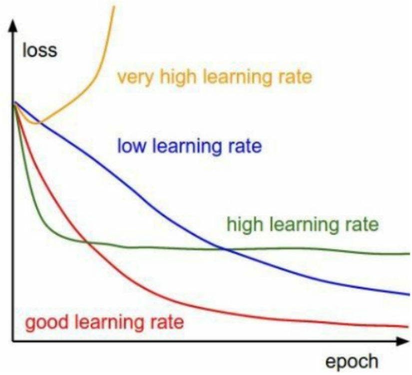
Learning Rate α.

Learning rate α is a scalar parameter used to set the rate of adjustments/updates in order to reduce the training errors faster.

Picking a learning rate value is an art, which can lead to your model:

- training & learning fast,
- training & learning slow,
- not training & learning at all.

Unfortunately, we won't be mastering that art in this course.



Iterative Approach: Example

- If we select learning rate $\alpha = \frac{1}{35}$:
  - $w_1^{upd} = 50 + \frac{1}{35} 2(350) = 70,$
  - $w_2^{upd} = 50 + \frac{1}{35} 5(350) = 100,$
  - $w_3^{upd} = 50 + \frac{1}{35} 3(350) = 80$

$w_i^{upd} = w_i + \alpha x_i (y - \hat{y})$

Iterative Approach: Example

- If we select learning rate $\alpha = \frac{1}{35}$:
  - $w_1^{upd} = 50 + \frac{1}{35}2(350) = 70$,
  - $w_2^{upd} = 50 + \frac{1}{35}5(350) = 100$,
  - $w_3^{upd} = 50 + \frac{1}{35}3(350) = 80$
- The updated predicted value $\hat{y}$:

$$\hat{y} = \sum_i w_i^{upd} x_i = 880,$$

which is much closer to the true value $y = 850$.

$2(70) + 5(100) + 3(80) = 880$

residual $= 880 - 850 = -30$

Iterative Approach: Example

- If we select learning rate $\alpha = \frac{1}{35}$:
  - $w_1^{upd} = 50 + \frac{1}{35}2(350) = 70,$
  - $w_2^{upd} = 50 + \frac{1}{35}5(350) = 100,$
  - $w_3^{upd} = 50 + \frac{1}{35}3(350) = 80$
- The updated predicted value $\hat{y}$:

$$\hat{y} = \sum_i w_i^{upd} x_i = 880,$$

  which is much closer to the true value $y = 850$.
- Notice that, while the weight for fish and ketchup got **better** (70 is closer to 150, 80 is closer to 100), the weight for chips got **worse** (100 instead of correct value 50)!
- We aren't guaranteed a weight improvement at each update, but over time, given sufficient data, weights typically converge to the correct values.

$$w_1^{upd^2} = 70 + \frac{1}{35} 2(-30) = 68.28$$

$$w_2^{upd^2} = 100 + \frac{1}{35}(5)(-30) = 95.714$$

$$w_3^{upd^3} = 80 + \frac{1}{35}(3)(-30) = 77.43$$

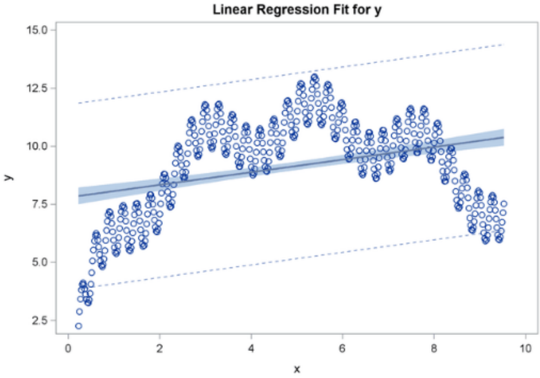
Linear Neuron Model: Limitations

Question. If we stick with a Linear Neuron Model, where each neuron simply outputs a weighted linear combination of its outputs, what type of function of original inputs $x_1, \ldots, x_n$ would we inevitably get?

Linear Neuron Model: Limitations

Question. If we stick with a Linear Neuron Model, where each neuron simply outputs a weighted linear combination of its outputs, what type of function of original inputs $x_1, \ldots, x_n$ would we inevitably get?

Answer. Linear function.



Linear functions can only do that much when dealing with non-linearity.