Chapter 12

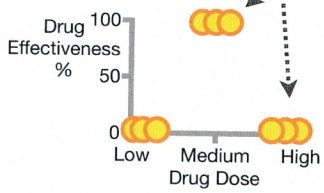
Neural Networks!!!

Neural Networks
Part One:

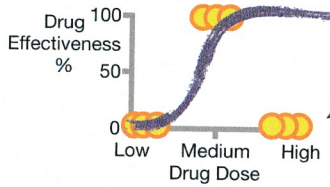
Understanding How Neural Networks Work

Neural Networks: Main Ideas

1 **The Problem:** We have data that show the Effectiveness of a drug for different Doses....



...and the **s-shaped squiggle** that **Logistic Regression** uses would not make a good fit. In this example, it would misclassify the high Doses.

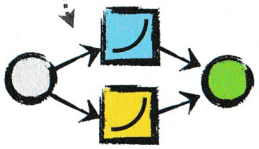


Hey! Can't we solve this problem with a **Decision Tree** or a **Support Vector Machine**?

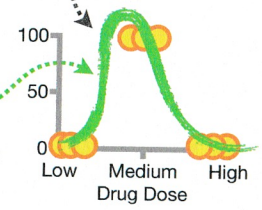
Yes!!! But they will probably not all perform the same, so it's a good idea to try each one to see which works best.

2 **A Solution:** Although they sound super intimidating, all **Neural Networks** do is fit **fancy squiggles** or **bent shapes** to data. And like **Decision Trees** and **SVMs**, **Neural Networks** do fine with any relationship among the variables.

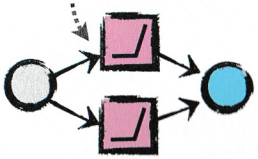
For example, we could get this **Neural Network**...



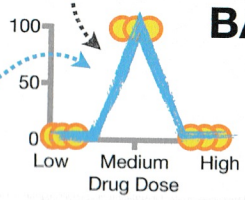
...to fit a **fancy squiggle** to the data...



...or we could get this **Neural Network**...



...to fit a **bent shape** to the data.

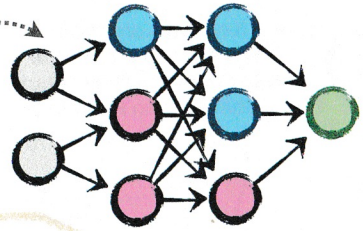


BAM!!!

Terminology Alert!!! Anatomy of a Neural Network

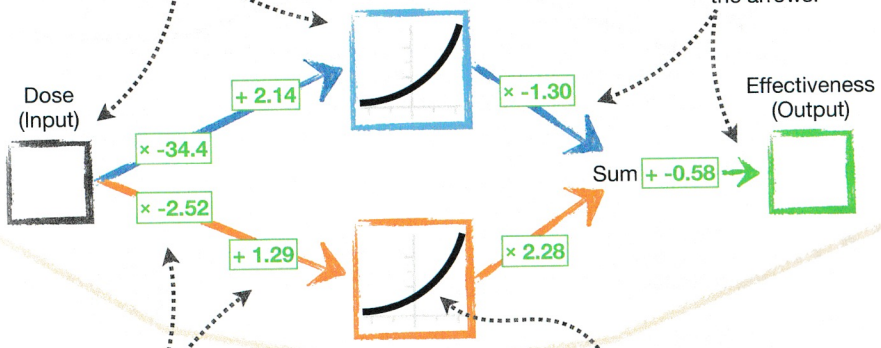
(Oh no! It's the dreaded **Terminology Alert!!!**)

1 Although **Neural Networks** usually look like super complicated groups of neurons connected by synapses, which is where the name **Neural Network** comes from, they're all made from the same simple parts.



2 **Neural Networks** consist of **Nodes**, the square boxes...

...and connections between **Nodes**, the arrows.



4 The numbers along the connections represent parameter values that were estimated when this **Neural Network** was fit to data using a process called **Backpropagation**. In this chapter, we'll see exactly what the parameters do and how they're estimated, step-by-step.

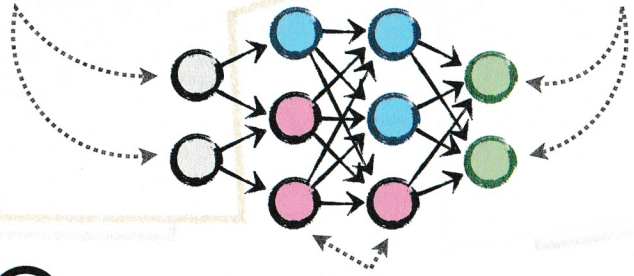
5 The bent or curved lines inside some of the **Nodes** are called **Activation Functions**, and they make **Neural Networks** flexible and able to fit just about any data.

Terminology Alert!!! Layers

(Oh no! *A Double* **Terminology Alert!!!**)

1 **Neural Networks** are organized in **Layers**. Usually, a **Neural Network** has multiple **Input Nodes** that form an **Input Layer**...

2 ...and usually there are multiple **Output Nodes** that form an **Output Layer**...

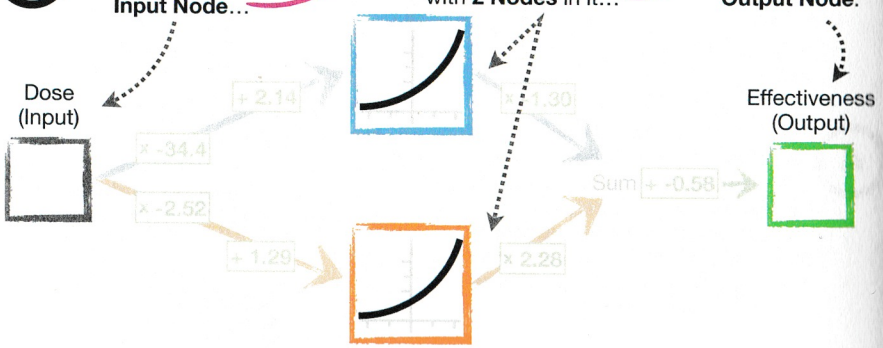


3 ...and the **Layers** of **Nodes** between the **Input** and **Output Layers** are called **Hidden Layers**. Part of the *art* of **Neural Networks** is deciding how many **Hidden Layers** to use and how many **Nodes** should be in each one. Generally speaking, the more **Layers** and **Nodes**, the more complicated the shape that can be fit to the data.

4 In the example we'll use, we have a single **Input Node**...

...a single **Hidden Layer** with **2 Nodes** in it...

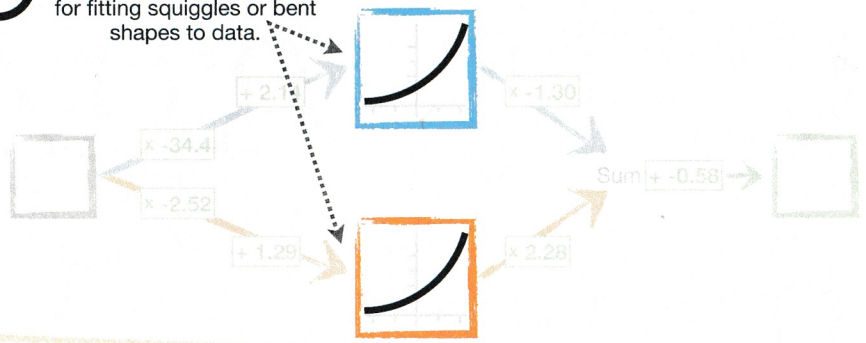
...and a single **Output Node**.



Terminology Alert!!! Activation Functions

(Oh no! *A TRIPLE* **TERMINOLOGY ALERT!!!**)

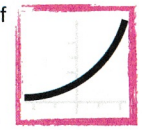
1 The **Activation Functions** are the basic building blocks for fitting squiggles or bent shapes to data.



2 There are lots of different **Activation Functions**. Here are three that are commonly used:

ReLU, which is short for **Rectified Linear Unit** and sounds like the name of a robot, is probably the most commonly used **Activation Function** with large **Neural Networks**. It's a **Bent Line**, and the bend is at $x = 0$.

SoftPlus, which sounds like a brand of toilet paper, is a modified form of the **ReLU Activation Function**. The big difference is that instead of the line being bent at **0**, we get a nice **Curve**.



3 Although they might sound fancy, **Activation Functions** are just like the mathematical functions you learned when you were a teenager: you plug in an x-axis coordinate, do the math, and the output is a y-axis coordinate.

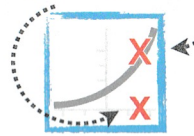
Lastly, the **Sigmoid Activation Function** is an **s-shaped squiggle** that's frequently used when people teach **Neural Networks** but is rarely used in practice.

For example, the **SoftPlus** function is:

$$\textbf{SoftPlus}(x) = \textbf{log}(1 + e^{x})$$

...where the **log()** function is the natural log, or **log base** e, and e is **Euler's Number**, which is roughly **2.72**.

So, if we plug in an x-axis value, $x = 2.14$...

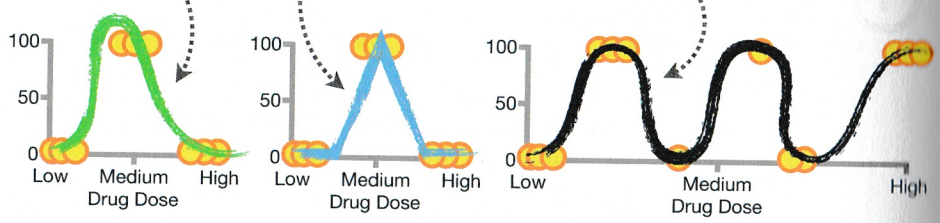


...then the **SoftPlus** will tell us the y-axis coordinate is **2.25**, because $\textbf{log}(1 + e^{2.14}) = \textbf{2.25}$.

Now let's talk about the main ideas of how **Activations Functions** work.

Activation Functions: Main Ideas

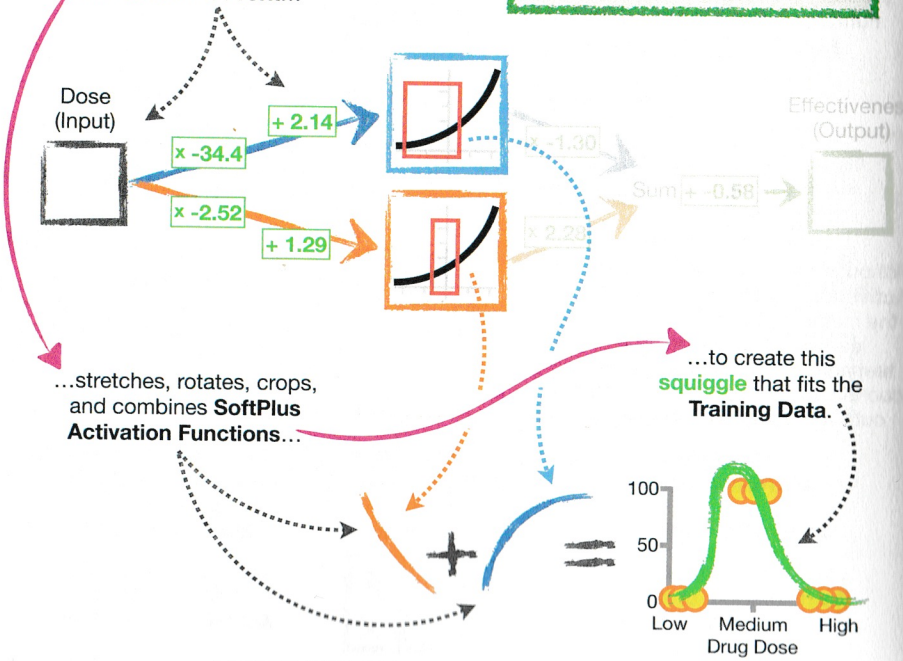
1 **The Problem:** We need to create new, exciting shapes that can fit any dataset with a **squiggle** or **bent lines**, even when the dataset is super complicated.



2 **The Solution: Neural Networks** stretch, rotate, crop, and combine **Activation Functions** to create new, exciting shapes that can fit anything!!!

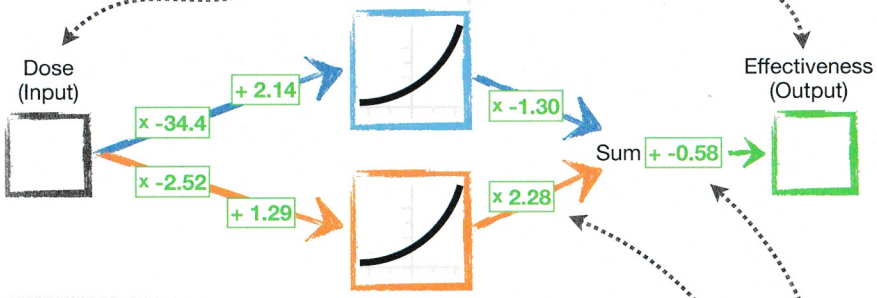
NOTE: The purpose of this illustration is simply to show you where we're headed in the next section, so rather than think too hard about it, read on!!!

In the next section, **A Neural Network in Action: Step-by-Step**, we'll show you how this **Neural Network**...

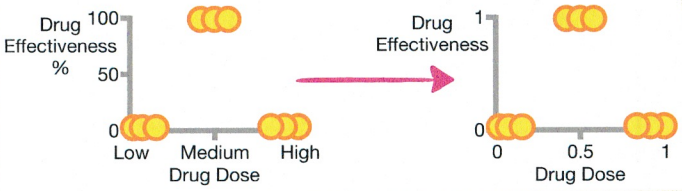


A Neural Network in Action: Step-by-Step

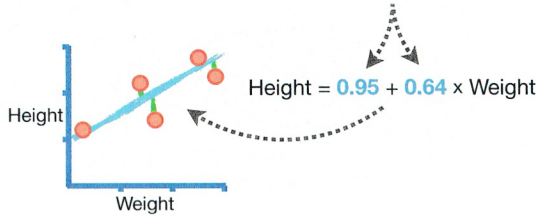
1 A lot of people say that **Neural Networks** are black boxes and that it's difficult to understand what they're doing. Unfortunately, this is true for big, fancy **Neural Networks**. However, the good news is that it's *not* true for simple ones. So, let's walk through how this simple **Neural Network** works, one step at a time, by plugging in Dose values from low to high and seeing how it converts the Dose (input) into predicted Effectiveness (output).



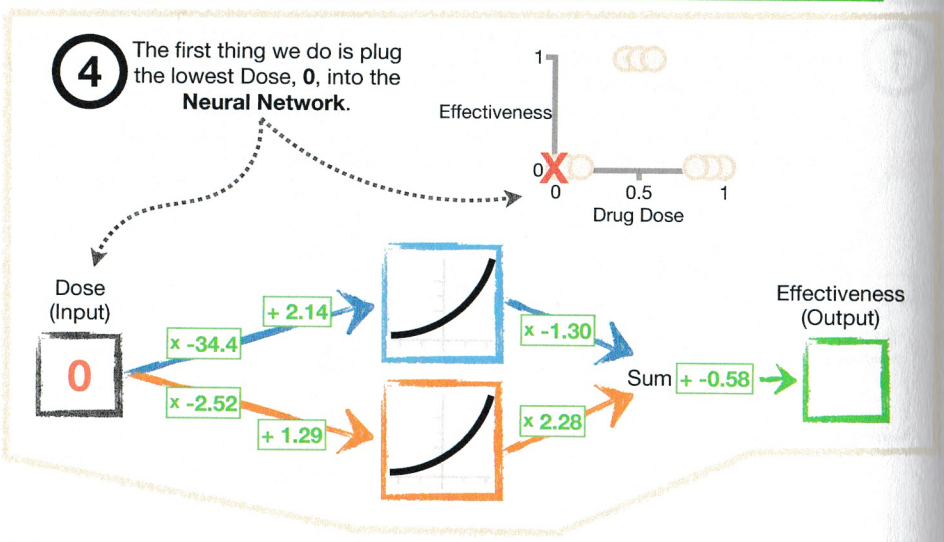
2 **NOTE:** To keep the math in this section simple, let's scale both the x- and y-axes so that they go from **0**, for *low* values, to **1**, for *high* values.

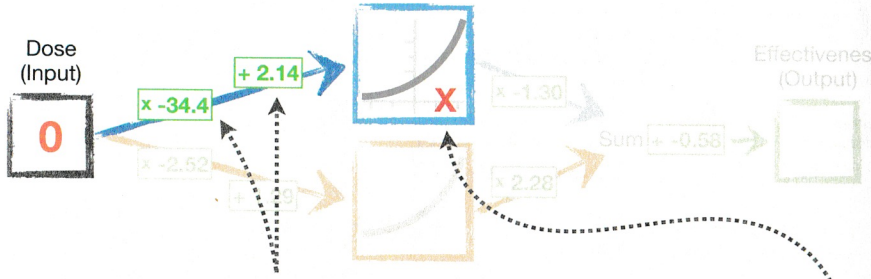


3 **ALSO NOTE:** These numbers are *parameters* that are estimated using a method called **Backpropagation**, and we'll talk about how that works, step-by-step, later. For now, just assume that these values have already been optimized, much like the **slope** and **intercept** are optimized when we fit a line to data.



A Neural Network in Action: Step-by-Step



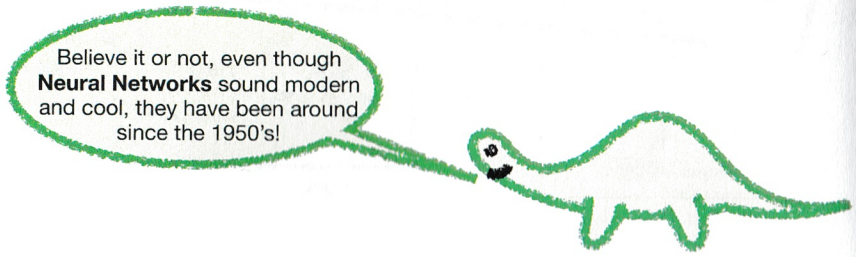


5 Now, the connection from the **Input Node** to the *top* **Node** in the **Hidden Layer** multiplies the Dose by **-34.4** and then adds **2.14**...

(**Dose** x -34.4) + 2.14 = x-axis coordinate

(**0** x -34.4) + 2.14 = x-axis coordinate = 2.14

...to get a new x-axis coordinate for the **Activation Function**, **2.14**.



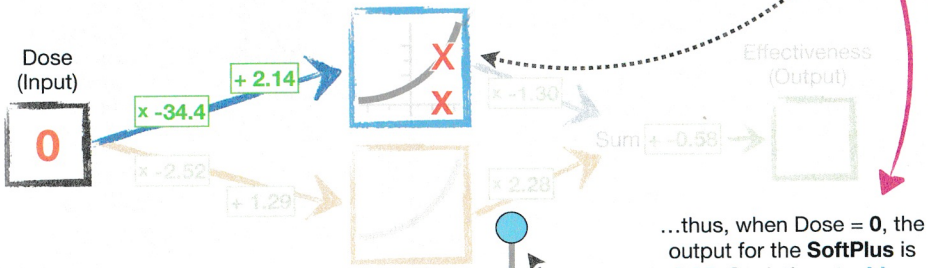
A Neural Network in Action: Step-by-Step

6 Then we plug the x-axis coordinate, **2.14** into the **SoftPlus Activation Function**...

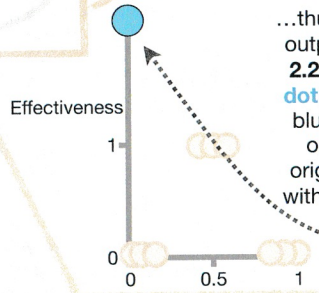
...and do the math, just like we saw earlier...

...and we get **2.25**. So, when the x-axis coordinate is **2.14**, the corresponding y-axis value is **2.25**...

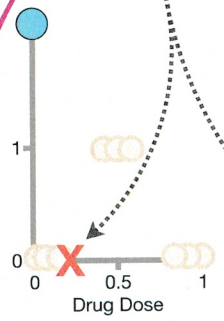
SoftPlus(x) = **SoftPlus**(2.14) = **log**($1 + e^{2.14}$) = y-axis coordinate = **2.25**



...thus, when Dose = **0**, the output for the **SoftPlus** is **2.25**. So, let's put a **blue dot**, corresponding to the blue activation function, onto the graph of the original data at Dose = **0** with a y-axis value = **2.25**.



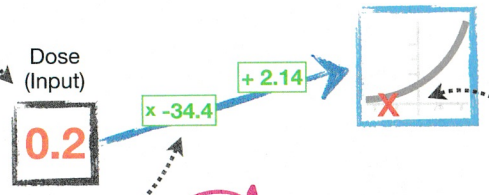
7 Now let's increase the Dose to **0.2**...



...and calculate the new x-axis coordinate by multiplying the Dose by **-34.4** and adding **2.14** to get **-4.74**...

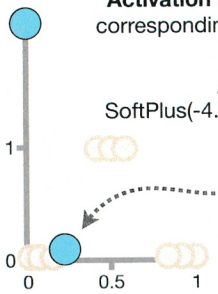
(**Dose** x -34.4) + 2.14

(**0.2** x -34.4) + 2.14 = -4.74



...and plug **-4.74** into the **SoftPlus Activation Function** to get the corresponding y-axis value, **0.01**.

SoftPlus(-4.74) = **log**($1 + e^{-4.74}$) = 0.01



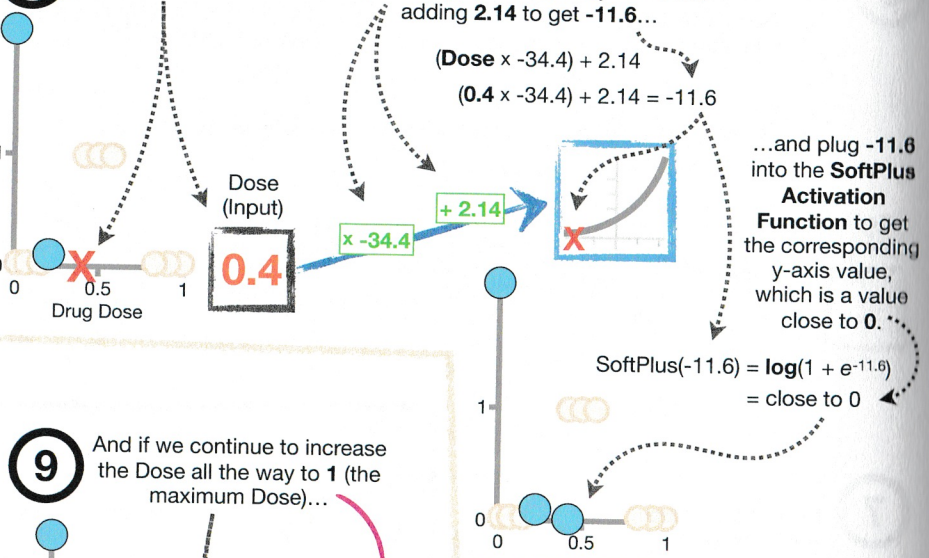
A Neural Network in Action: Step-by-Step

8 Now let's increase the Dose to **0.4**...

...and calculate the new x-axis coordinate by multiplying the Dose by **-34.4** and adding **2.14** to get **-11.6**...

(**Dose** x -34.4) + 2.14

(**0.4** x -34.4) + 2.14 = -11.6



...and plug **-11.6** into the **SoftPlus Activation Function** to get the corresponding y-axis value, which is a value close to **0**.

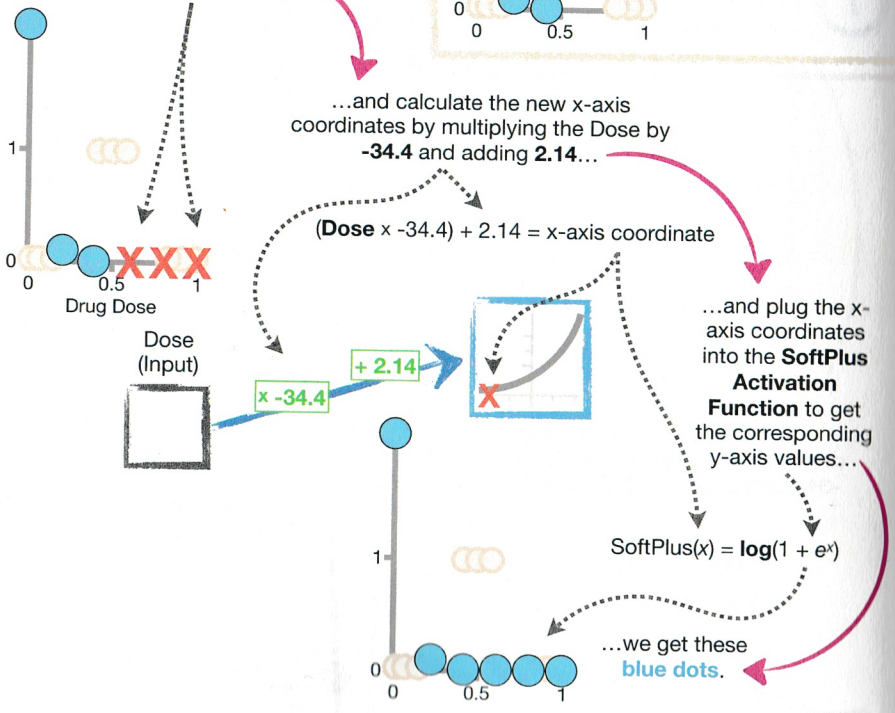
SoftPlus(-11.6) = **log**(1 + $e^{-11.6}$)

= close to 0

9 And if we continue to increase the Dose all the way to **1** (the maximum Dose)...

...and calculate the new x-axis coordinates by multiplying the Dose by **-34.4** and adding **2.14**...

(**Dose** x -34.4) + 2.14 = x-axis coordinate



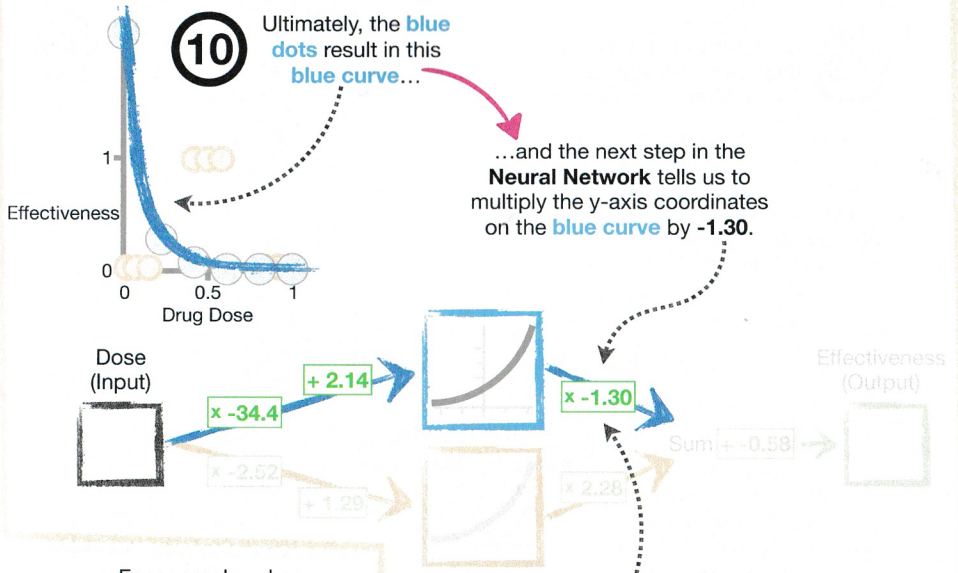
...and plug the x-axis coordinates into the **SoftPlus Activation Function** to get the corresponding y-axis values...

...we get these blue dots.

A Neural Network in Action: Step-by-Step

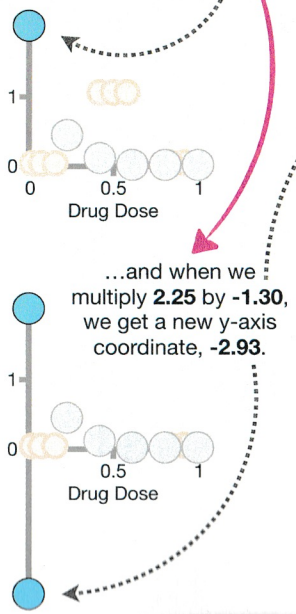
10 Ultimately, the blue dots result in this blue curve...

...and the next step in the **Neural Network** tells us to multiply the y-axis coordinates on the blue curve by **-1.30**.



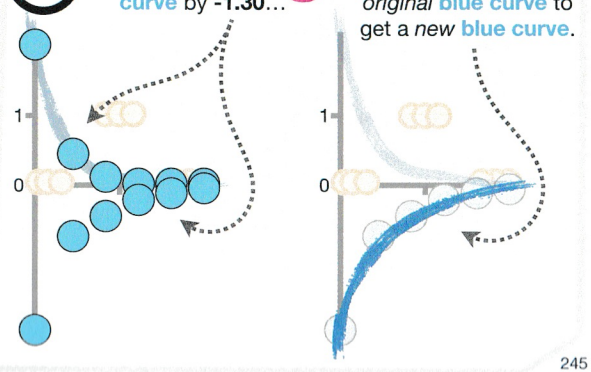
11 For example, when Dose = **0**, the current y-axis coordinate on the blue curve is **2.25**...

...and when we multiply **2.25** by **-1.30**, we get a new y-axis coordinate, **-2.93**.



12 Likewise, when we multiply all of the y-axis coordinates on the blue curve by **-1.30**...

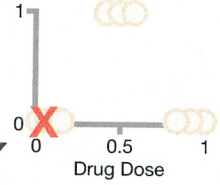
...we end up flipping and stretching the *original* blue curve to get a *new* blue curve.



A Neural Network in Action: Step-by-Step

13 Okay, we just finished a major step, so this is a good time to review what we've done so far.

First, we plugged Dose values, ranging from **0** to **1**, into the input...

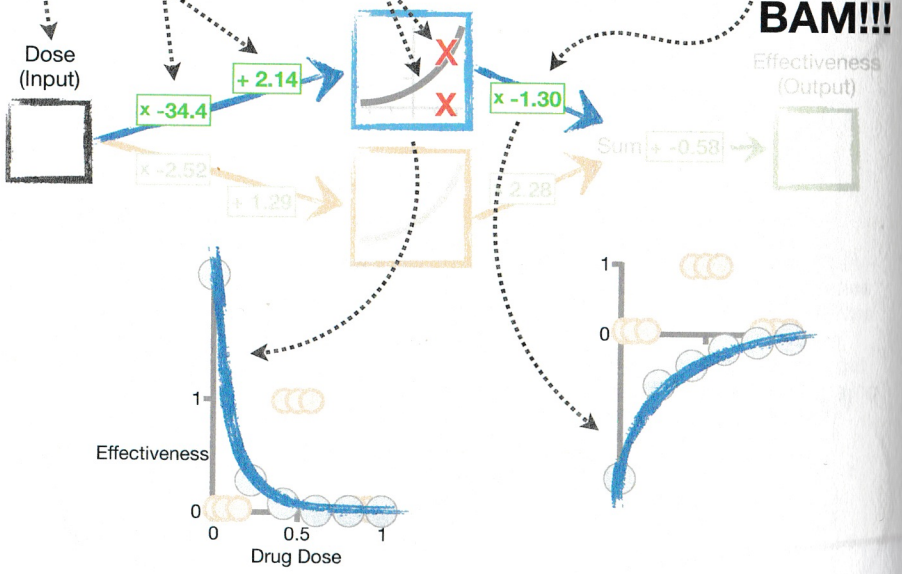


...then we transformed the Doses by multiplying them by **-34.4** and adding **2.14**...

...then the **SoftPlus Activation Function** converted the transformed Doses into y-axis coordinates...

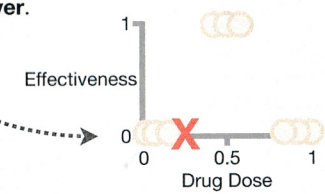
...and lastly, we transformed the y-axis coordinates by multiplying them by **-1.30**.

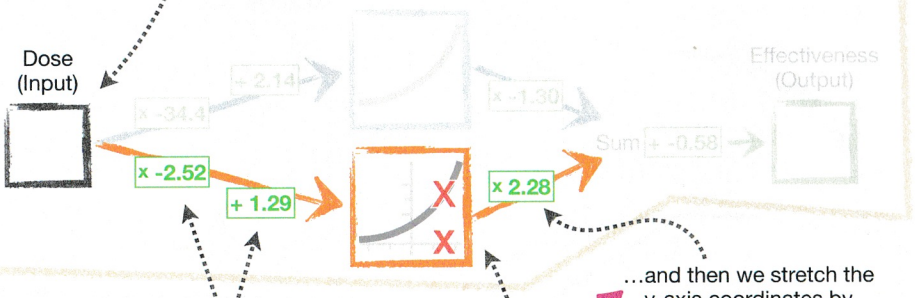
BAM!!!



A Neural Network in Action: Step-by-Step

14 Now we run Dose values through the connection to the *bottom* **Node** in the **Hidden Layer**.

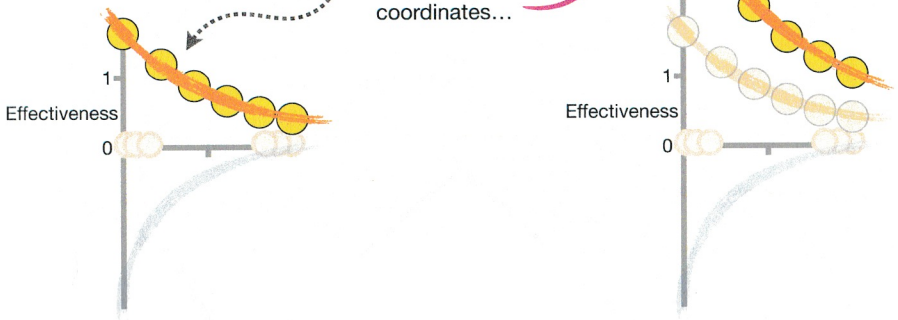




15 The good news is that the only difference between what we just did and what we're doing now is that now we multiply the Doses by **-2.52** and add **1.29**...

...before using the **SoftPlus Activation Function** to convert the transformed Doses into y-axis coordinates...

...and then we stretch the y-axis coordinates by multiplying them by **2.28** to get this **orange curve**.

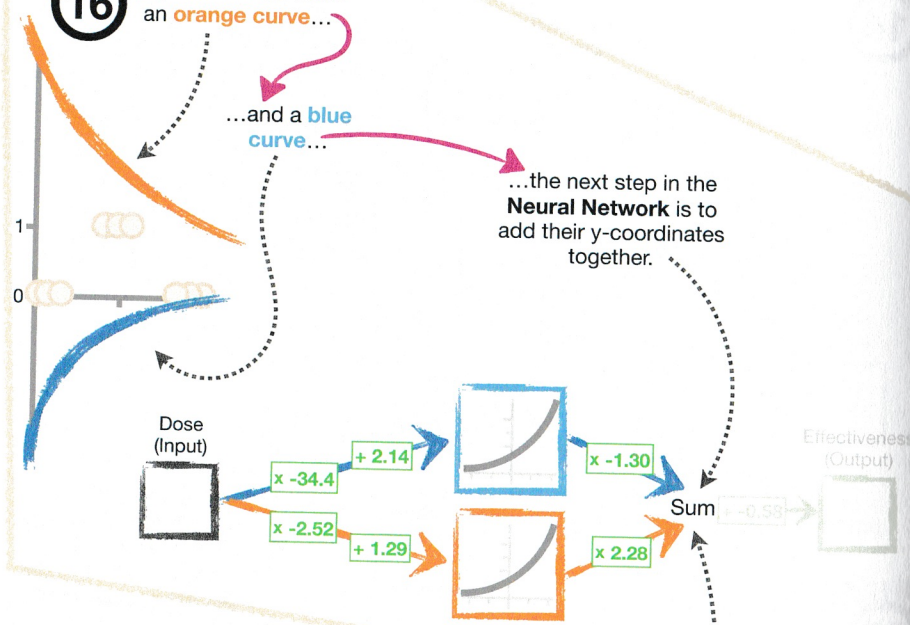


A Neural Network in Action: Step-by-Step

16 So, now that we have an **orange curve**...

...and a **blue curve**...

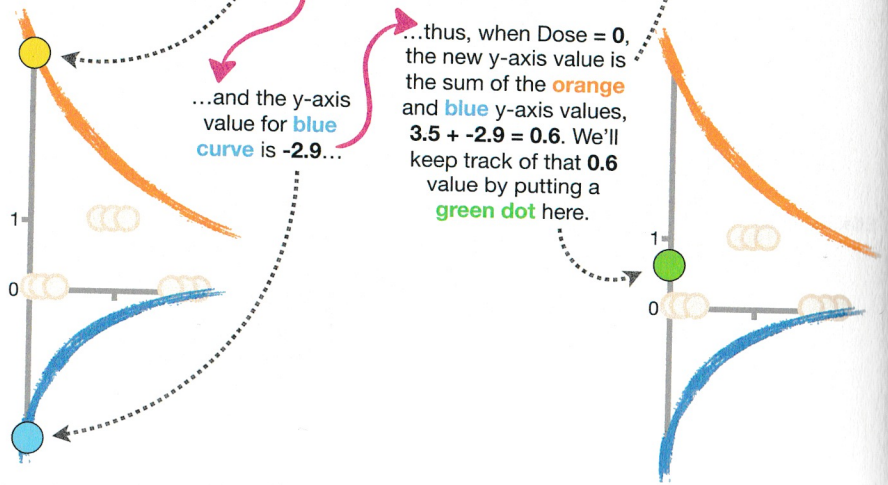
...the next step in the **Neural Network** is to add their y-coordinates together.



17 For example, when Dose = **0**, the y-axis value for the **orange curve** is **3.5**...

...and the y-axis value for **blue curve** is **-2.9**...

...thus, when Dose = **0**, the new y-axis value is the sum of the **orange** and **blue** y-axis values, **3.5 + -2.9 = 0.6**. We'll keep track of that **0.6** value by putting a **green dot** here.

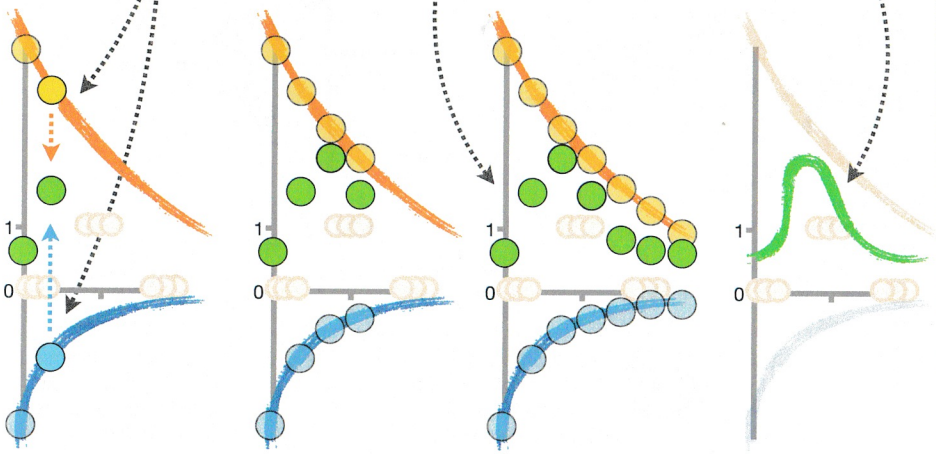


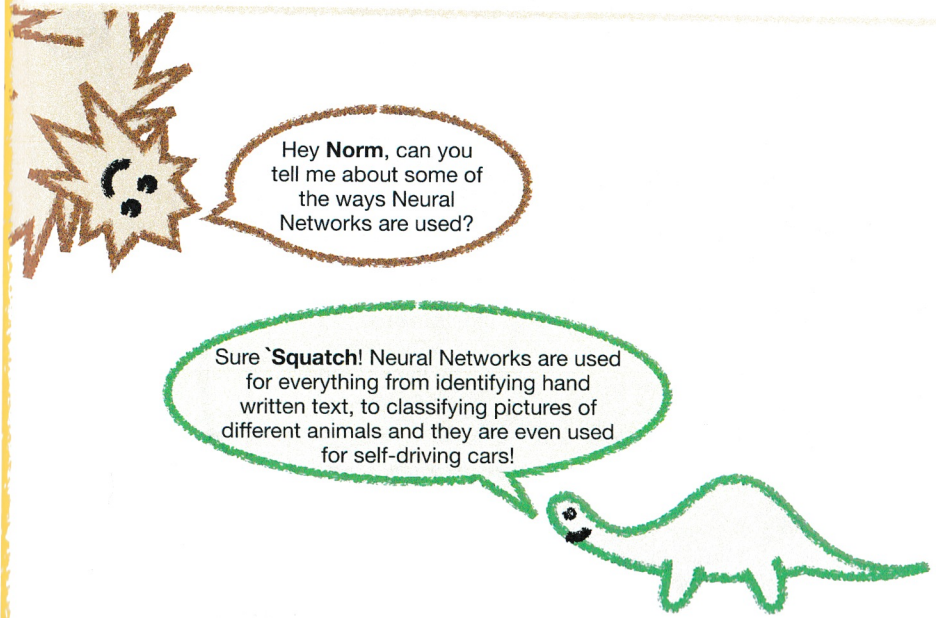
A Neural Network in Action: Step-by-Step

18 Then, for the remaining Dose values, we just add the y-axis coordinates of the **blue** and **orange** curves...

...plot the resulting **green dot** values...

...and, after connecting the dots, we ultimately end up with this **green squiggle**.

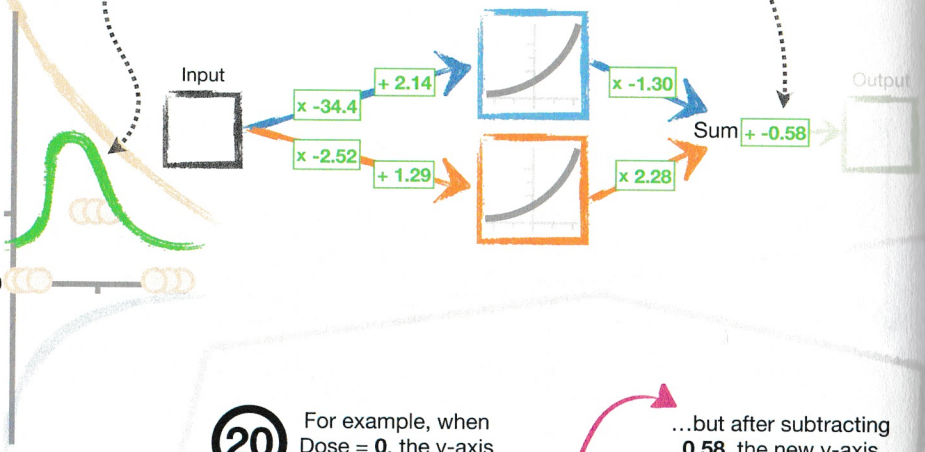




A Neural Network in Action: Step-by-Step

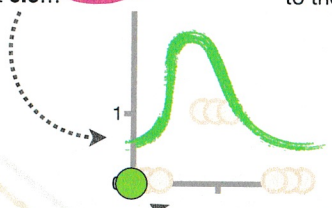
19 Now that we have this **green squiggle** from the **blue** and **orange** curves...

...we're ready for the final step, which is to add **-0.58** to each y-axis value on the **green squiggle**.

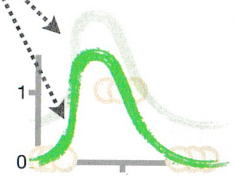


20 For example, when Dose = **0**, the y-axis coordinate on the **green squiggle** starts out at **0.6**...

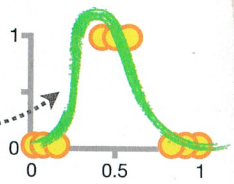
...but after subtracting **0.58**, the new y-axis coordinate for when Dose = **0** is **0.0** (rounded to the nearest **tenth**).



21 Likewise, subtracting **0.58** from all of the other y-axis coordinates on the **green squiggle** shifts it down...



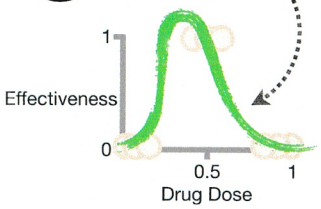
...and, ultimately, we end up with a **green squiggle** that fits the **Training Data**.



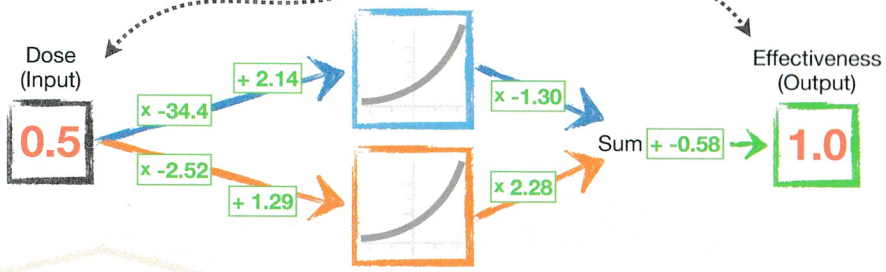
DOUBLE BAM!!!

A Neural Network in Action: Step-by-Step

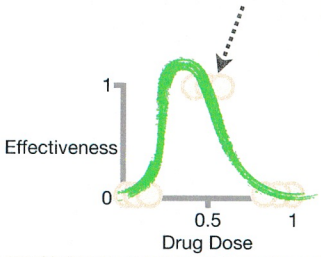
22 Hooray!!! At long last, we see the final **green squiggle**...



...that this **Neural Network** uses to predict Effectiveness from Doses.

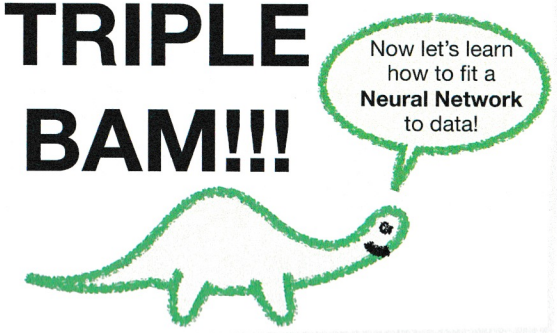


23 Now, if we're wondering if a medium Dose of **0.5** will be effective, we can look at the **green squiggle** and see that the output from the **Neural Network** will be **1**, and thus, Dose = **0.5** will be effective.



24 Alternatively, if we plug **Dose = 0.5** into the **Neural Network** and do the math, we get **1**, and thus, Dose = **0.5** will be effective.

TRIPLE BAM!!!

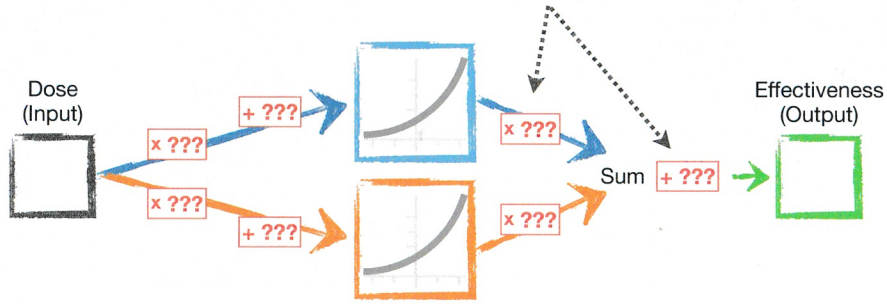


Neural Networks
Part Deux:

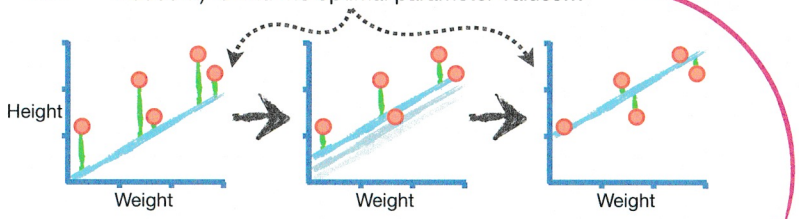
Fitting a Neural Network to Data with Backpropagation

Backpropagation: Main Ideas

1 **The Problem:** Just like for **Linear Regression**, **Neural Networks** have parameters that we need to optimize to fit a squiggle or bent shape to data. How do we find the optimal values for these parameters?

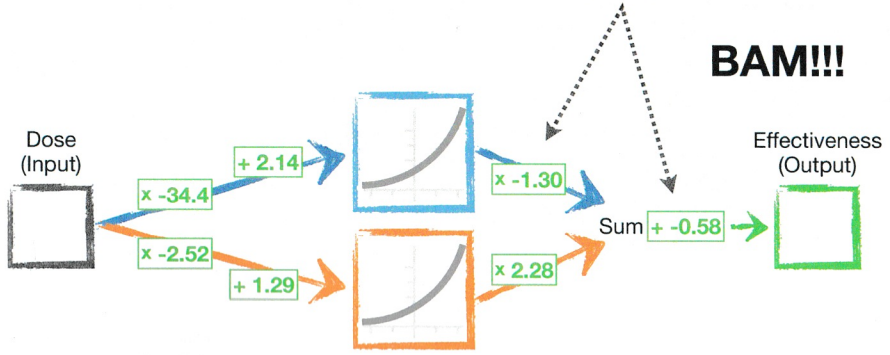


2 **A Solution:** Just like for **Linear Regression**, we can use **Gradient Descent** (or **Stochastic Gradient Descent**) to find the optimal parameter values...



...however, we don't call it **Gradient Descent**. That would be too easy. Instead, because of how the derivatives are found for each parameter in a **Neural Network** (from the back to the front), we call it **Backpropagation**.

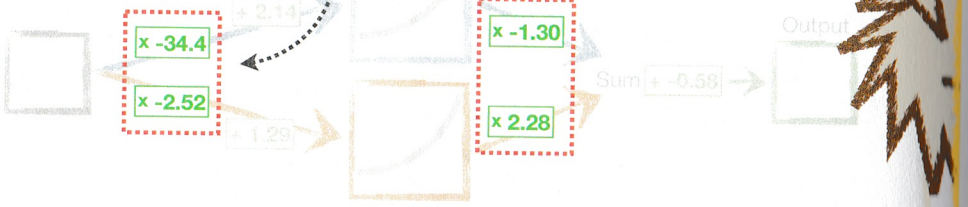
BAM!!!



Terminology Alert!!! Weights and Biases

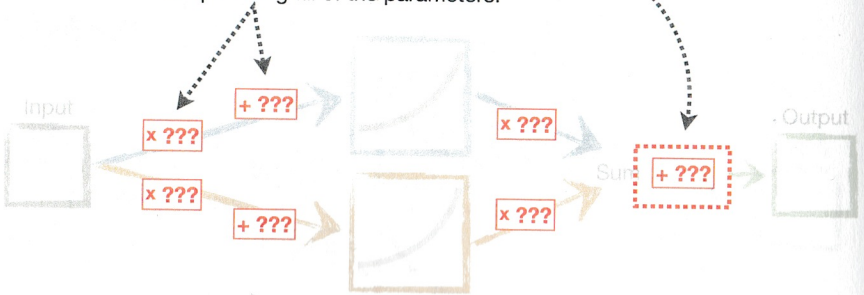
OH NO! It's the dreaded **Terminology Alert**!!!

1 In **Neural Networks**, the parameters that we multiply are called **Weights**...



2 ...and the parameters we add are called **Biases**.

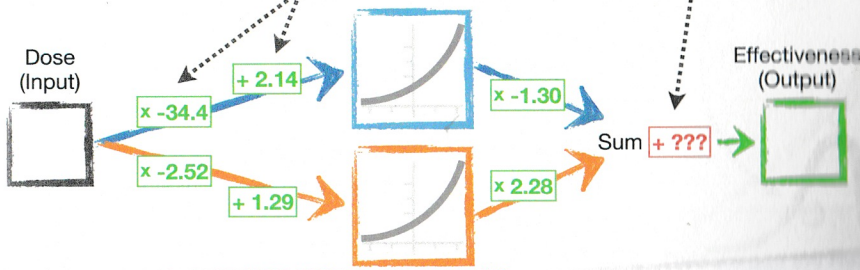
3 In the example that follows, we'll use **Backpropagation** to optimize the **Final Bias**. However, the same process and ideas apply to optimizing all of the parameters.



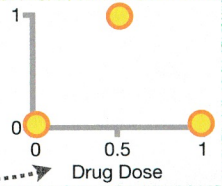
Backpropagation: Details Part 1

1 In this example, let's assume that we already have the optimal values for all of the **Weights** and **Biases**...

...except for the **Final Bias**. So the goal is to use **Backpropagation** to optimize this **Final Bias**.

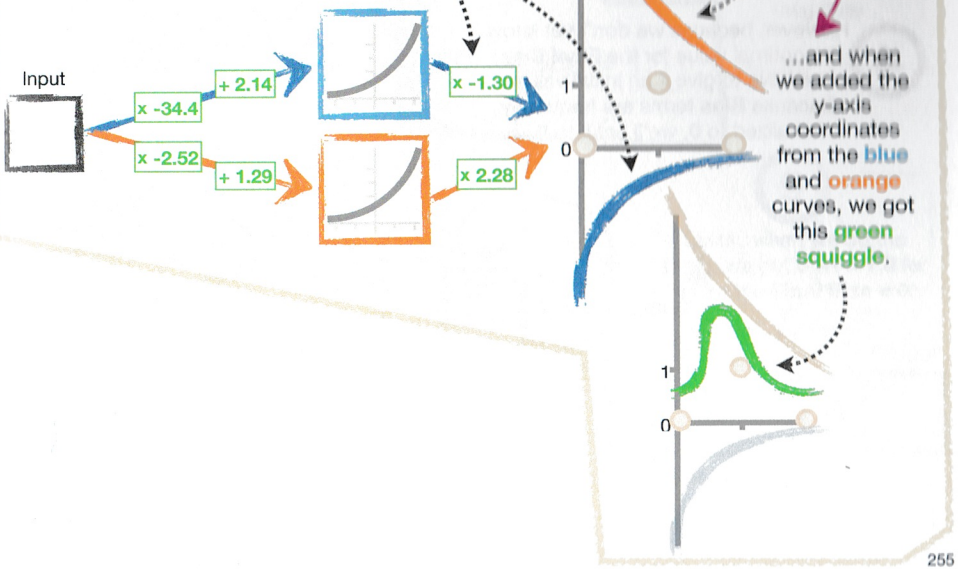


NOTE: To keep the math relatively simple, from now on, the **Training Data** will have only **3 Dose** values, **0**, **0.5**, and **1**.



2 Now let's remember that when we plugged Doses into the input, the top of the **Hidden Layer** gave us this **blue curve**...

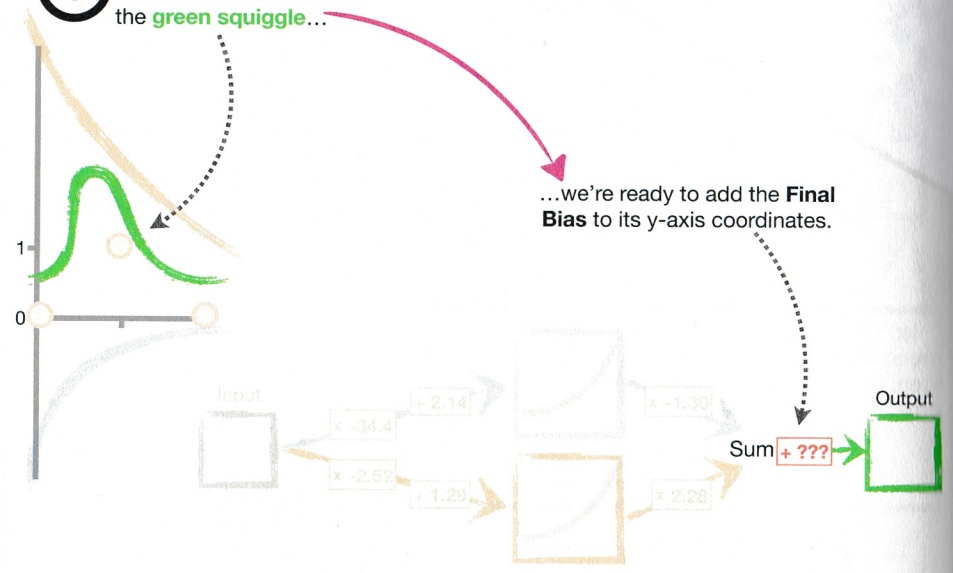
...and the bottom of the **Hidden Layer** gave us this **orange curve**...



Backpropagation: Details Part 2

3 Now that the **Neural Network** has created the green squiggle...

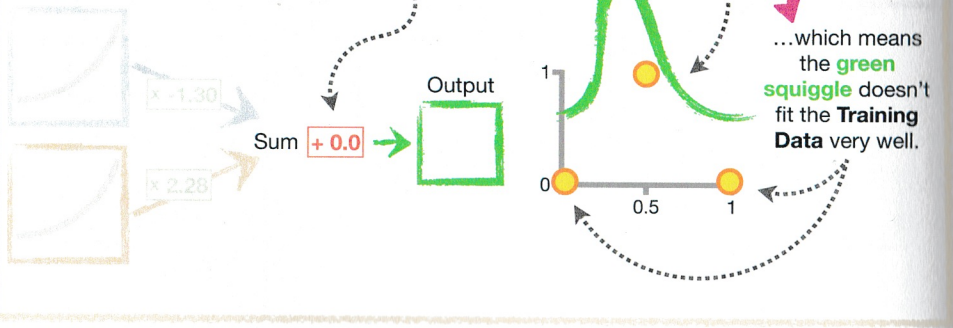
...we're ready to add the **Final Bias** to its y-axis coordinates.



4 However, because we don't yet know the optimal value for the **Final Bias**, we have to give it an initial value. Because **Bias** terms are frequently initialized to **0**, we'll set it to **0**...

...and adding **0** to all of the y-axis coordinates on the green squiggle leaves it right where it is...

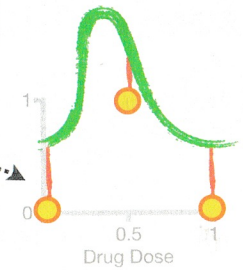
...which means the green squiggle doesn't fit the **Training Data** very well.



Backpropagation: Details Part 3

5 Now, just like we did for R^2, **Linear Regression**, and **Regression Trees**, we can quantify how well the green squiggle fits all of the **Training Data** by calculating the **Sum of the Squared Residuals** (**SSR**).

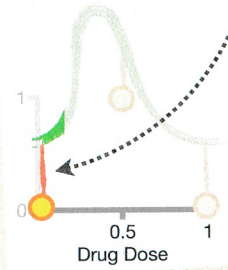
$$\textbf{SSR} = \sum_{i=1}^{n} (\text{Observed}_i - \text{Predicted}_i)^2$$

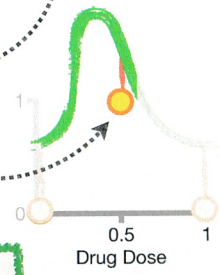


6 For example, for the first Dose, **0**, the Observed Effectiveness is **0** and the green squiggle created by the **Neural Network** predicts **0.57**, so we plug in **0** for the Observed value and **0.57** for the Predicted value into the equation for the **SSR**.

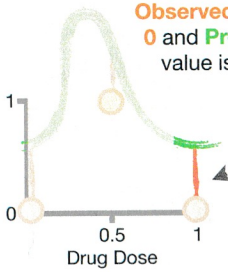
7 Then we add the **Residual** for when Dose = **0.5**. Now the Observed Effectiveness is **1**, but the green squiggle predicts **1.61**.

$$\textbf{SSR} = (0 - 0.57)^2 + (1 - 1.61)^2 + (0 - 0.58)^2 = 1.0$$

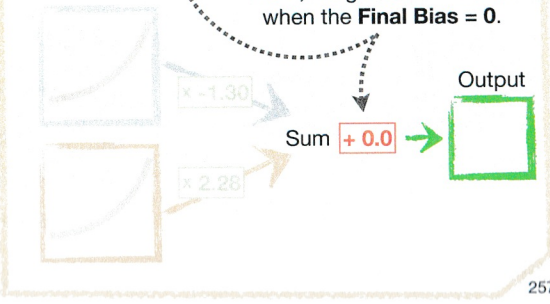




8 Then we add the **Residual** when Dose = **1**, where the Observed value is **0** and Predicted value is **0.58**.

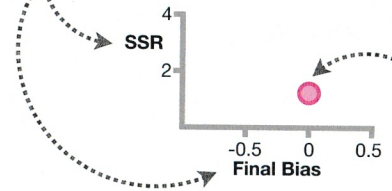


9 Lastly, when we do the math, we get **SSR = 1.0** for when the **Final Bias = 0**.

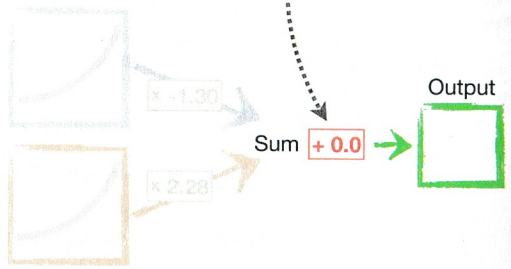


Backpropagation: Details Part 4

10 Now we can compare the **SSR** for different values for the **Final Bias** by plotting them on this graph that has values for the **Final Bias** on the x-axis and the corresponding **SSR** on the y-axis.



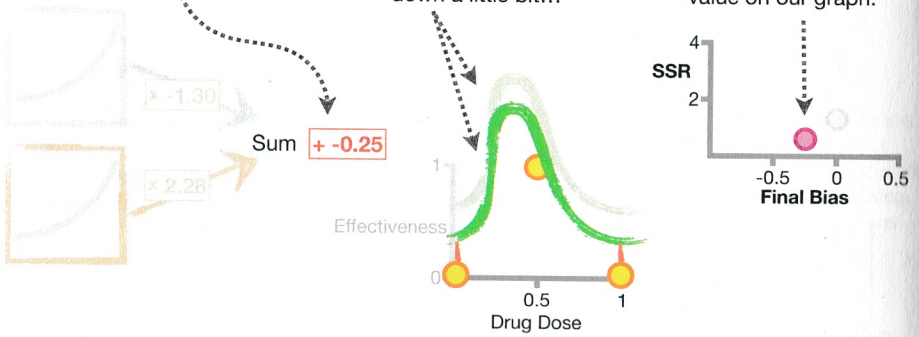
Thus, when the **Final Bias = 0**, we can draw a **pink dot** at **SSR = 1**.



11 If we set the **Final Bias** to **-0.25**...

...then we shift the **green squiggle** down a little bit...

...and we can calculate the **SSR** and plot the value on our graph.

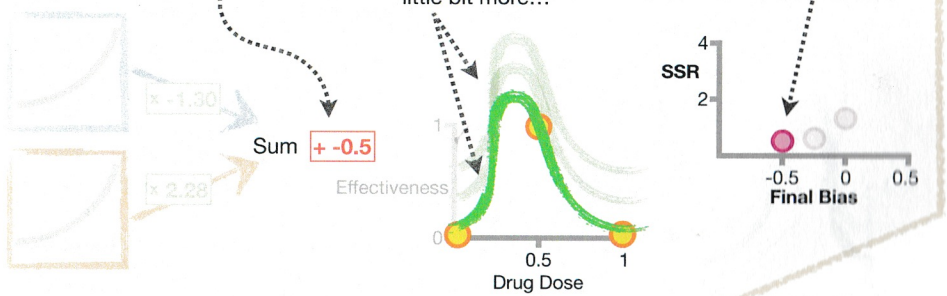


Backpropagation: Details Part 5

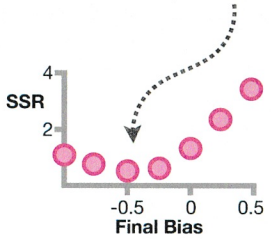
12 Setting the **Final Bias** to **-0.5**...

...shifts the **green squiggle** down a little bit more...

...and results in a slightly lower **SSR**.

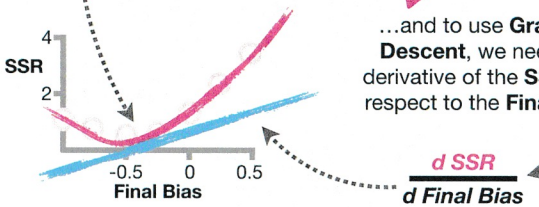


13 And if we try a bunch of different values for the **Final Bias**, we can see that the lowest **SSR** occurs when the **Final Bias** is close to **-0.5**...



14 ...however, instead of just plugging in a bunch of numbers at random, we'll use **Gradient Descent** to quickly find the lowest point in the **pink curve**, which corresponds to the **Final Bias** value that gives us the minimum **SSR**...

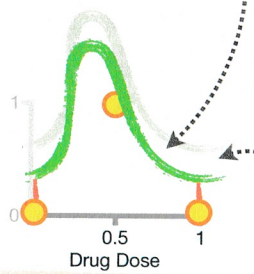
...and to use **Gradient Descent**, we need the derivative of the **SSR** with respect to the **Final Bias**.



Backpropagation: Details Part 6

15 Remember, each **Predicted** value in the **SSR** comes from the **green squiggle**...

$$SSR = \sum_{i=1}^{n} (Observed_i - Predicted_i)^2$$



Predicted = green squiggle

= blue curve + orange curve + **Final Bias**

16 ...and the **green squiggle** comes from the last part of the **Neural Network**, when we add the y-axis values from the **blue** and **orange** curves to the **Final Bias**.

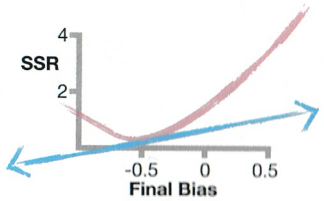
Sum + -0.25

17 Now, because the **Predicted** values link the **SSR**...

...to the **Final Bias**...

$$SSR = \sum_{i=1}^{n} (Observed_i - Predicted_i)^2$$

Predicted = green squiggle = blue curve + orange curve + **Final Bias**



$$\frac{d\ SSR}{d\ Final\ Bias}$$

...we can use **The Chain Rule** to solve for the derivative of the **SSR** with respect to the **Final Bias**.

Backpropagation: Details Part 7

18 **The Chain Rule** says that the derivative of the **SSR** with respect to the **Final Bias**...

$$\frac{d\ SSR}{d\ Final\ Bias} = \frac{d\ SSR}{d\ Predicted} \times \frac{d\ Predicted}{d\ Final\ Bias}$$

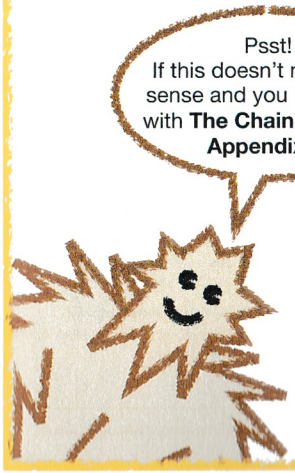
...is the derivative of the **SSR** with respect to the **Predicted** values...

$$SSR = \sum_{i=1}^{n} (Observed_i - Predicted_i)^2$$

...multiplied by the derivative of the **Predicted** values with respect to the **Final Bias**.

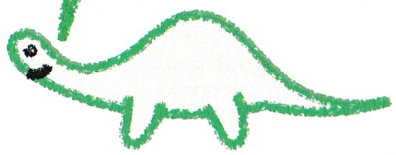
Predicted = green squiggle = blue curve + orange curve + **Final Bias**

BAM!!!



Psst! If this doesn't make any sense and you need help with **The Chain Rule**, see **Appendix F**.

The Chain Rule is worth learning about because shows up all over the place in machine learning. Specifically, anything that uses **Gradient Descent** has a good chance of involving **The Chain Rule**.



Backpropagation: Details Part 8

19 Now that we see that the derivative of the **SSR** with respect to the **Final Bias**... ...is the derivative of the **SSR** with respect to the **Predicted** values... ...multiplied by the derivative of the **Predicted** values with respect to the **Final Bias**...

$$\frac{d\,SSR}{d\,Final\,Bias} = \frac{d\,SSR}{d\,Predicted} \times \frac{d\,Predicted}{d\,Final\,Bias}$$

20 ...we can solve for the *first* part, the derivative of the **SSR** with respect to the **Predicted** values... ...which, in turn, can be solved using **The Chain Rule**...

NOTE: For more details on how to solve for this derivative, see **Chapter 5**.

$$\frac{d\,SSR}{d\,Predicted} = \frac{d}{d\,Predicted}\sum_{i=1}^{n}(Observed_i - Predicted_i)^2$$

...by moving the square to the front...

$$= \sum_{i=1}^{n} 2 \times (Observed_i - Predicted_i) \times -1$$

...and multiplying everything by the derivative of the stuff inside the parentheses, which is **-1**...

...and, lastly, we simplify by multiplying **2** by **-1**.

$$\frac{d\,SSR}{d\,Predicted} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i)$$

BAM!!!

We solved for the *first* part of the derivative. Now let's solve for the *second* part.

Backpropagation: Details Part 9

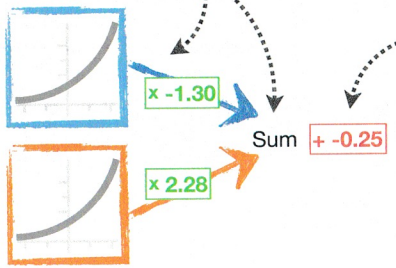
21 The *second* part, the derivative of the **Predicted** values with respect to the **Final Bias**... ...is the derivative of the **green squiggle** with respect to the **Final Bias**...

$$\frac{d\,Predicted}{d\,Final\,Bias} = \frac{d}{d\,Final\,Bias}\,\text{green squiggle}$$

...which, in turn, is the derivative of the sum of the **blue** and **orange** curves and the **Final Bias**.

$$= \frac{d}{d\,Final\,Bias}(\text{blue curve} + \text{orange curve} + \text{Final Bias})$$

22 Now, remember that the **blue** and **orange** curves... ...were created before we got to the **Final Bias**...



...thus, the derivatives of the **blue** and **orange** curves with respect to the **Final Bias** are both **0** because they do not depend on the **Final Bias**...

$$\frac{d}{d\,Final\,Bias}(\text{blue curve} + \text{orange curve} + \text{Final Bias}) = 0 + 0 + 1$$

...and the derivative of the **Final Bias** with respect to the **Final Bias** is **1**. So, when we do the math, the derivative of the Predicted values with respect to the **Final Bias** is **1**.

$$\frac{d\,Predicted}{d\,Final\,Bias} = 1$$

DOUBLE BAM!!!

We solved for the *second* part of the derivative.

Backpropagation: Details Part 10

23 Now, to get the derivative of the **SSR** with respect to the **Final Bias**, we simply plug in...

...the derivative of the **SSR** with respect to the **Predicted** values...

...and the derivative of the **Predicted** values with respect to the **Final Bias**.

$$\frac{d\,SSR}{d\,Predicted} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i)$$

$$\frac{d\,Predicted}{d\,Final\,Bias} = 1$$

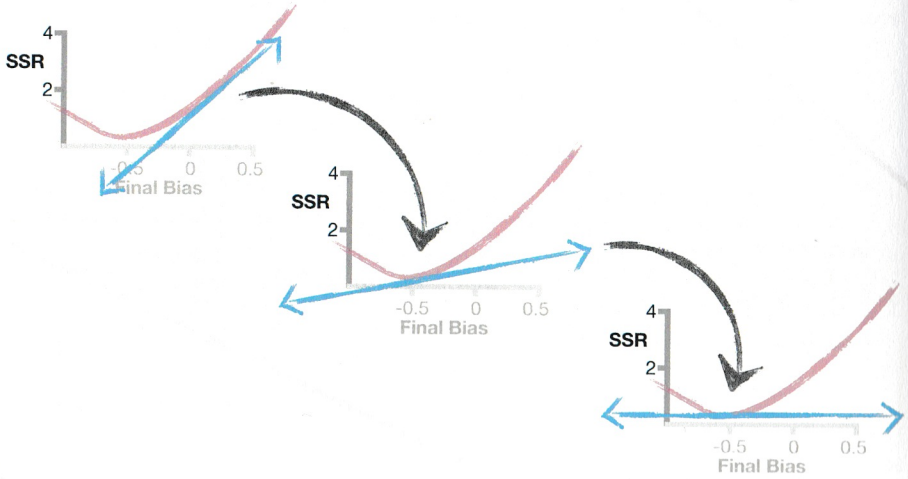
$$\frac{d\,SSR}{d\,Final\,Bias} = \frac{d\,SSR}{d\,Predicted} \times \frac{d\,Predicted}{d\,Final\,Bias}$$

$$\frac{d\,SSR}{d\,Final\,Bias} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i) \times 1$$

24 At long last, we have the derivative of the **SSR** with respect to the **Final Bias!!!**

TRIPLE BAM!!!

25 In the next section, we'll plug the derivative into **Gradient Descent** and solve for the optimal value for the **Final Bias**.

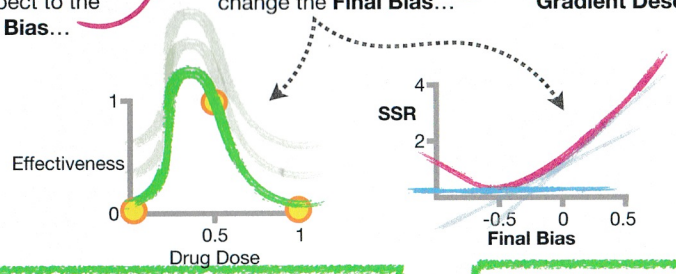


Backpropagation: Step-by-Step

1 Now that we have the derivative of the **SSR** with respect to the **Final Bias**...

...which tells us how the **SSR** changes when we change the **Final Bias**...

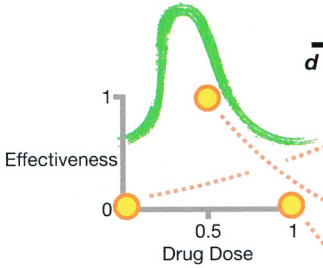
...we can optimize the **Final Bias** with **Gradient Descent**.



$$\frac{d\,SSR}{d\,Final\,Bias} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i) \times 1$$

NOTE: We're leaving the **"x 1"** term in the derivative to remind us that it comes from **The Chain Rule**. However, multiplying by **1** doesn't do anything, and you can omit it.

2 First, we plug in the **Observed** values from the **Training Dataset** into the derivative of the **SSR** with respect to the **Final Bias**.



$$\frac{d\,SSR}{d\,Final\,Bias} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i)^2 \times 1$$

$$= -2 \times (Observed_1 - Predicted_1) \times 1$$

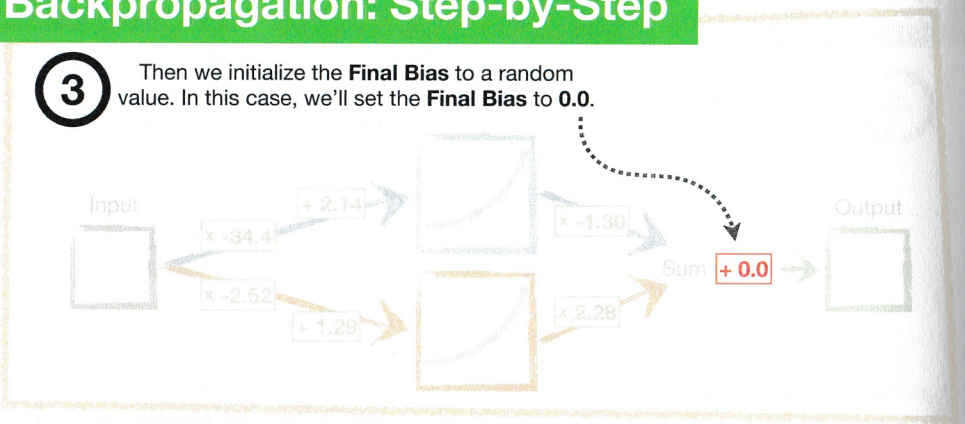
$$+ -2 \times (Observed_2 - Predicted_2) \times 1$$

$$+ -2 \times (Observed_3 - Predicted_3) \times 1$$

$$\frac{d\,SSR}{d\,Final\,Bias} = -2 \times (0 - Predicted_1) \times 1$$
$$+ -2 \times (1 - Predicted_2) \times 1$$
$$+ -2 \times (0 - Predicted_3) \times 1$$

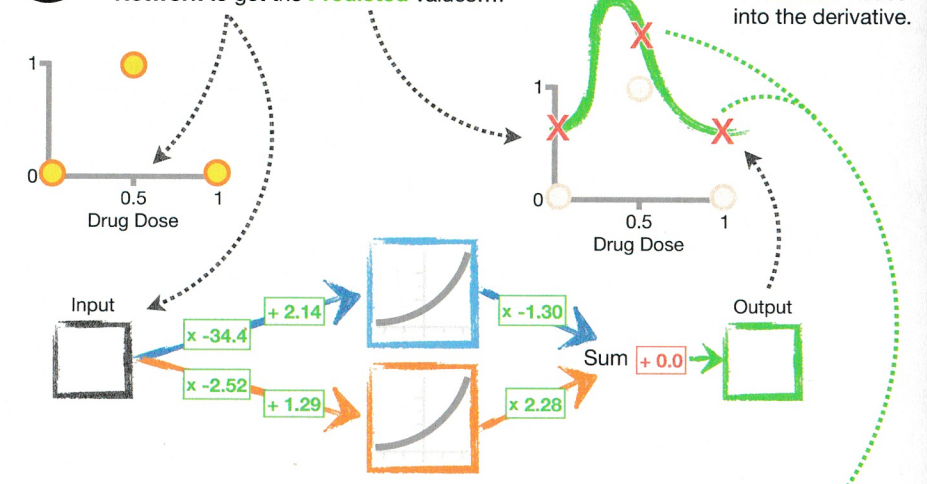
Backpropagation: Step-by-Step

3 Then we initialize the **Final Bias** to a random value. In this case, we'll set the **Final Bias** to **0.0**.



4 Then we run the **3 Doses** from the **Training Dataset**, **0**, **0.5** and **1**, through the **Neural Network** to get the **Predicted** values....

...and we plug the **Predicted** values into the derivative.



$$\frac{d\,SSR}{d\,Final\,Bias} = -2 \times (0 - Predicted) \times 1 + -2 \times (1 - Predicted) \times 1 + -2 \times (0 - Predicted) \times 1$$

$$= -2 \times (0 - 0.57) \times 1 + -2 \times (1 - 1.61) \times 1 + -2 \times (0 - 0.58) \times 1$$

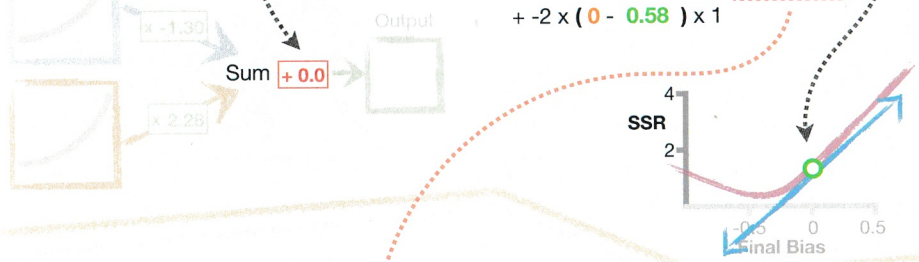
Backpropagation: Step-by-Step

5 Now we evaluate the derivative at the current value for the **Final Bias**, which, at this point, is **0.0**.

When we do the math, we get **3.5**...

...thus, when the **Final Bias = 0**, the slope of this tangent line is **3.5**.

$$\frac{d\,SSR}{d\,Final\,Bias} = -2 \times (0 - 0.57) \times 1 + -2 \times (1 - 1.61) \times 1 + -2 \times (0 - 0.58) \times 1 = 3.5$$



6 Then we calculate the **Step Size** with the standard equation for **Gradient Descent** and get **0.35**.

NOTE: In this example, we've set the **Learning Rate** to **0.1**.

Step Size = **Derivative** x **Learning Rate**

= **3.5** x **0.1**

= **0.35**

Gentle Reminder: The magnitude of the derivative is proportional to how big of a step we should take toward the minimum. The sign (+/-) tells us in what direction.

7 Lastly, we calculate a *new* value for the **Final Bias** from the *current* **Final Bias**...

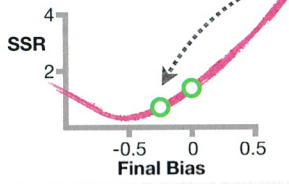
New Bias = **Current Bias** - **Step Size**

= **0.0** - **0.35**

= **-0.35**

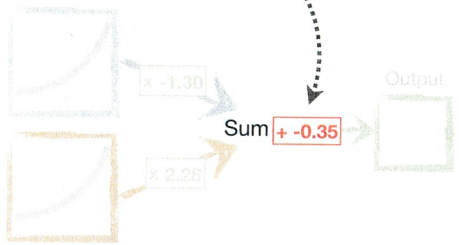
Remember, we initialized the **Final Bias** to **0.0**.

...and the *new* **Final Bias** is **-0.35**, which results in a lower **SSR**.



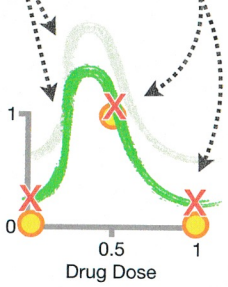
Backpropagation: Step-by-Step

8 With the new value for the **Final Bias**, **-0.35**...



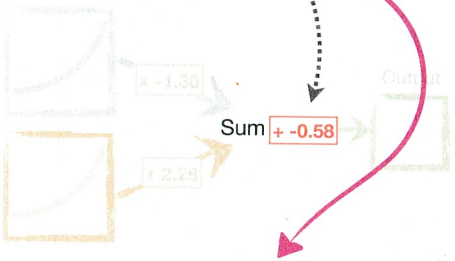
...we shift the **green squiggle** down a bit, and the **Predicted** values are closer to the **Observed** values.

BAM!!!



9 Now **Gradient Descent** iterates over the past three steps...

...and after **7** iterations, the **Final Bias** = **-0.58**...

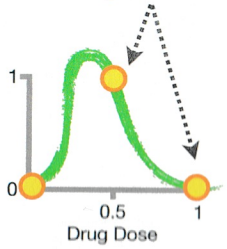


a Evaluate the derivative at the current value...

b Calculate the **Step Size**...

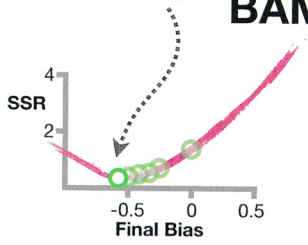
c Calculate the new value...

...and the **green squiggle** fits the **Training Data** really well...



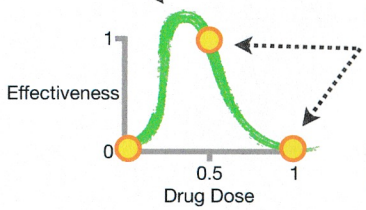
...and we've made it to the lowest **SSR**.

BAM!!!



Neural Networks: FAQ

Where the heck did this bump in the green squiggle come from?

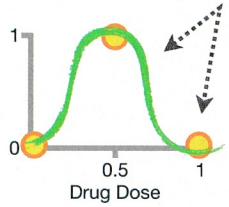


When we used **Backpropagation** to estimate the **Weights** and **Biases** in the **Neural Network**, we only calculated the **SSR** for the original **3** Doses, **0**, **0.5**, and **1**.

That means we only judge the **green squiggle** by how well it predicts Effectiveness at the original **3** Doses, **0**, **0.5**, and **1**, *and no other* Doses.

And that means the **green squiggle** can do whatever it wants in between the original **3** Doses, including making a strange bump that may or may not make good predictions. This is something I think about when people talk about using **Neural Networks** to drive cars. The **Neural Network** probably fits the **Training Data** really well, but there's no telling what it's doing between points, and that means it will be hard to predict what a self-driving car will do in new situations.

Wouldn't it be better if the green squiggle was a bell-shaped curve fit to the Training Data?



Maybe. Because we don't have any data between the **3** Doses in the **Training Data**, it's hard to say what the best fit would be.

When we have *Neural Networks*, which are cool and super flexible, why would we ever want to use *Logistic Regression*, which is a lot less flexible?

Neural Networks are cool, but deciding how many **Hidden Layers** to use and how many **Nodes** to put in each **Hidden Layer** and even picking the best **Activation Function** is a bit of an art form. In contrast, creating a model with **Logistic Regression** is a science, and there's no guesswork involved. This difference means that it can sometimes be easier to get **Logistic Regression** to make good predictions than a **Neural Network**, which might require a lot of tweaking before it performs well.

Furthermore, when we use a lot of variables to make predictions, it can be much easier to interpret a **Logistic Regression** model than a **Neural Network**. In other words, it's easy to know how **Logistic Regression** makes predictions. In contrast, it's much more difficult to understand how a **Neural Network** makes predictions.