CSE 375 Machine Learning and Pattern Recognition

11. Artificial Neural Networks

Slides Credit: N. Rich Nguyen

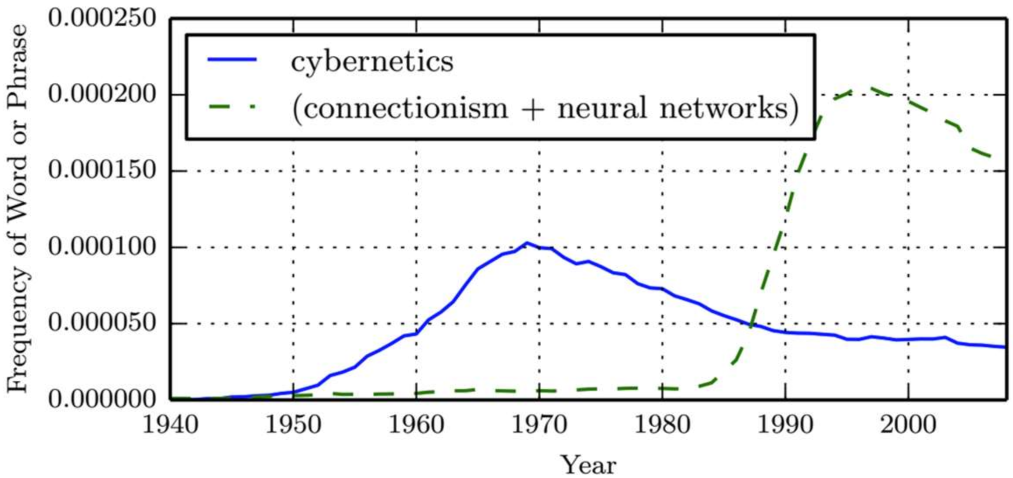
The neural perspective of deep learning

- Deep Learning is also known as **artificial neural network** as its early models are engineered systems *inspired* by the **brain** (biological neural network)
- However, artificial neural networks used for machine learning are generally **not** designed to be realistic models of biological function
- The modern term of "**deep learning**" is not necessarily neurally inspired, but appeals more to the principle of *learning at multiple levels of representation*
- Deep Learning has drawn heavily on our knowledge not only of the human brain architecture, but also in statistics and applied math

Historical Timeline* of Deep Learning

As a field, Deep Learning has been around for a long time and known under many names. Broadly speaking, there have been three waves of development:

- First wave: known as Cybernetics (1940s-1960s)
- Second wave: rebranded as Connectionism (1980s-1990s)
- Third wave: rebranded as Deep Learning (2006-present)



*These dates are for perspective only and not as definitive historical record of invention

First wave: Cybernetics (1940s-1960s)

- 1943: Inspired by the brain architecture, **neurons** and theories of biological learning proposed by McCulloch and Pitts
- 1958: **Perceptrons** created by Rosenblatt could learn the weights that defined the categories given inputs from each category.
- 1960: **Adaptive Linear Element** (ADALINE) by Widrow and Hoff could predict numeric values from data. Early success led to failed belief of its capability.
- 1969: **Backlash** against biologically inspired learning by Minsky and Papert mainly because its linear models could not learn the XOR function.
- 1970s: **First AI winter** as many promises would go unfulfilled

*These dates are for perspective only and not as definitive historical record of invention

Second wave: Connectionism (1980s-1990s)

- 1986: Revival of interest in research with **Backpropagation** by Rumelhart
- 1994: Advanced **modeling sequences** with recurrent neural nets by Bengio
- 1998: Early **convolutional neural nets** by LeCun (LeNet-5) performed well on MNIST dataset of handwritten digits
- 1990s-2000s: **Second AI winter** as deep networks were generally believed to be *very difficult to train*. Kernel methods (SVM) and graphical models were preferred by AI researchers and practitioners.
- 2004: The Canadian Institute for Advanced Research (**CIFAR**) helped *keep the research alive* with its Neural Computation and Adaptive Perception (**NCAP**) research initiative led by Hinton (U of Toronto), Bengio (U of Montreal) and LeCun (NYU).

*These dates are for perspective only and not as definitive historical record of invention

Third wave: Deep Learning (2006-present)

- 2006: A research breakthrough by Hinton, **deep belief network,** could be trained *efficiently* by a strategy called *greedy layer-wise pretraining*.
- 2007: Same training strategy adapted by other CIFAR-affiliated groups.
- 2012: First Deep Learning method, **AlexNet**, won the ImageNet Competition.
- 2014: Development of generative adversarial network (**GAN**)
- 2016: Achievement in deep reinforcement learning with **AlphaGo, AlphaZero**
- Present: Has seen tremendous growth in **popularity** and **usefulness**

Would this new wave of interest last?

*These dates are for perspective only and not as definitive historical record of invention

5

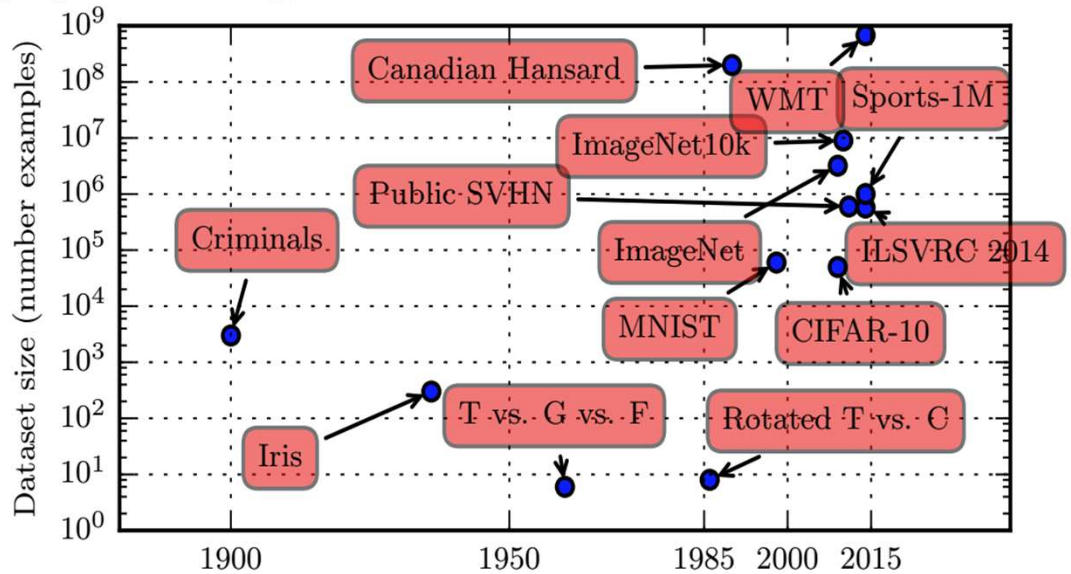
A few reasons why this time will have impact

1. **Data:** Huge *quantity of data* available to train
2. **Hardware:** Tremendous increase in *computing power*
3. **Algorithms:** Improved training procedures
4. **Investment:** A virtuous circle of *funding and progress*

Reason 1: Data

Game changer has been the rise of the Internet and the age of "Big Data" making it feasible to collect very large datasets for machine learning:

- User-generated tags on Flickr images and Youtube videos
- ImageNet dataset and annual competition
- Wikipedia for natural language processing



Reason 2: Hardware

Thanks to Moore's Law, we have the computational resources to run much larger models today:

- Availability of faster CPUs (5000X between 1990 and 2010)
- Advent of general purpose GPUs for parallel computing (neural network consisting of many small matrix multiplications are highly parallelizable).
- Beyond GPU: Google revealed its tensor processing unit (TPU), a new chip design developed from ground up to run deep neural networks (10X faster)
- Faster network connectivity

This trend is generally expected to continue well into the future

Reason 3: Algorithms

Previously, back-propagation through deep stack of layers had fade away feedback signal. Advent of several algorithmic improvements that allowed better gradient propagation:

- Better **activation function** for neural layers
- Improved **weight initialization** schemes
- Advanced **optimization schemes** (RMSProp and Adam)
- Many recent **algorithmic innovations** (batch normalization, residual connection, depth-wise separable convolutions)

Better software infrastructure and **libraries** (Theano, PyLearn, Torch, Caffee, MXNet, and TensorFlow)

Reason 4: Investment

As deep learning become the new state-of-the-art for computer vision and eventually all perceptual tasks, industry leaders take note.

Total venture capital investment in AI jumped from $19M (2011) to $384M (2014).

Many tech giants invested in R&D AI departments.

- Google acquired deep-learning startup DeepMind for $500M in 2013
- Baidu started a Deep Learning research center in Silicon Valley investing $300M in 2014
- Intel bought deep-learning hardware Nervana Systems for $400M in 2016

Research progress has reached a frenetic pace.

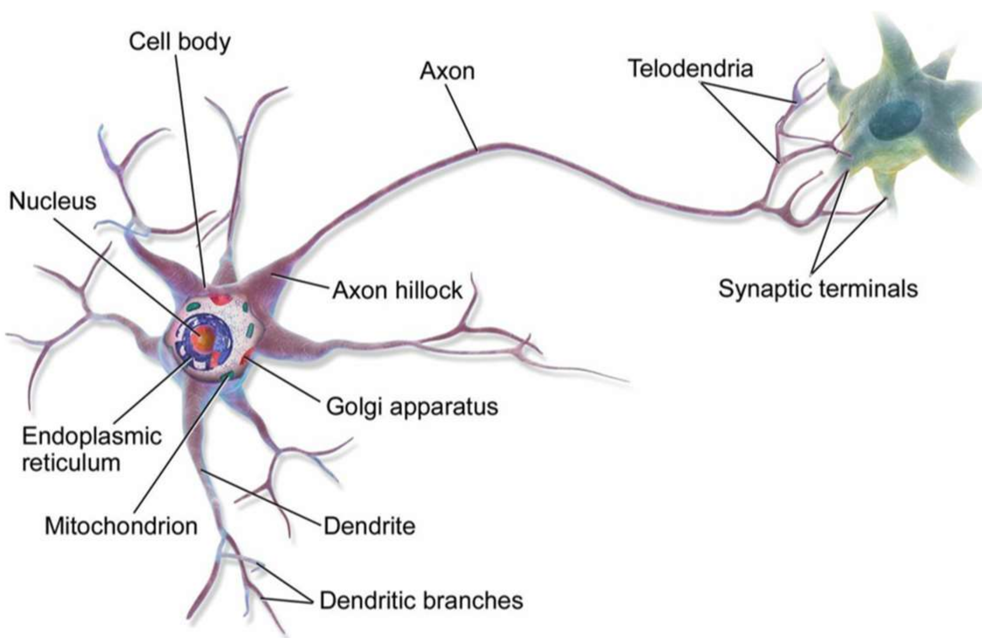
2018 Turing Award for Neural Networks

- The ACM **Turing Award**, often referred to as the "Nobel **Prize** of Computing," carries a $1 million **prize**.
- 2018 Turing Award is given for "the conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing."
- Hopfield and Geoffrey E. Hinton were awarded the 2024 Nobel Prize in Physics

Components of Neural Networks

Biological Neurons

- Perhaps the brain is the nature's ultimate inspiration for invention?
- It starts with an unusual looking cells found in animal cerebral cortex (brain)
- Neuron receives short **electrical impulses** from other neurons via synapses.
- When receives a **sufficient** number of signals, **fires its own signals**



Biological Neural Networks (BNNs)

- Individual neurons seems simple
- They are organized in a vast network of billions in consecutive layers
- Each neuron connected to thousands of other neurons.

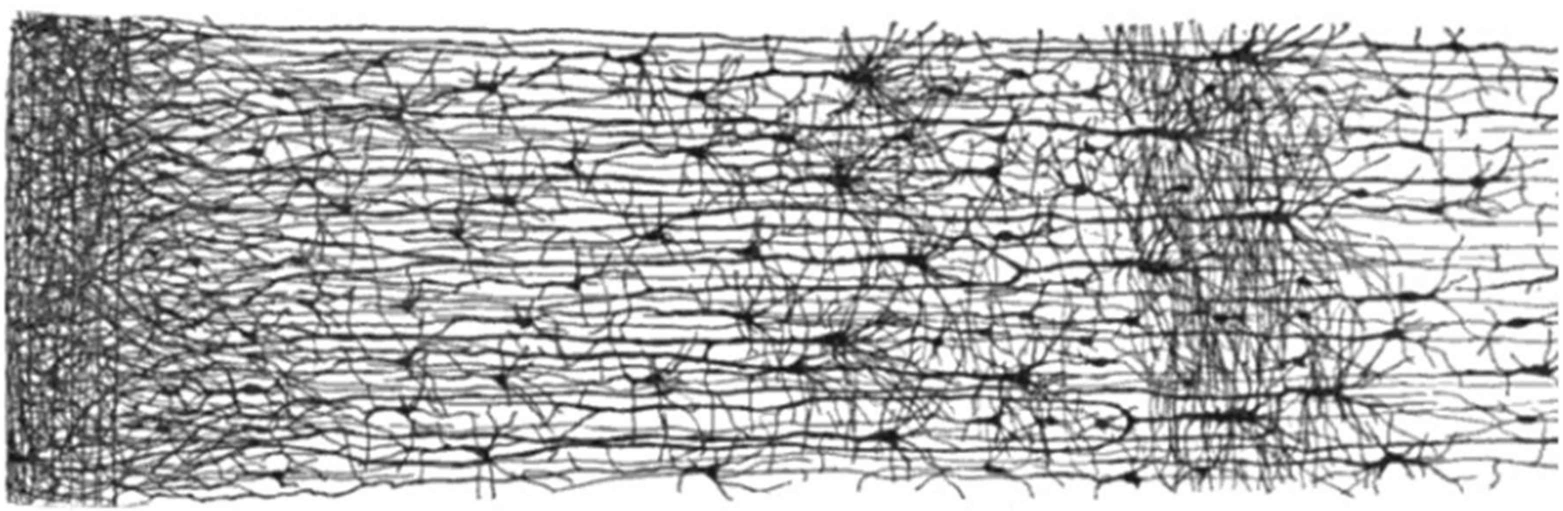


Figure 10-2. Multiple layers in a biological neural network (human cortex)[5]

Brain’s architecture for an intelligent machine

- Key idea of **artificial neural networks (ANNs)**
- ANNs are at the very **core** of Deep Learning
- Deep Learning is here to stay:
    - Classifying billions of images (**Google Images**)
    - Powering speech recognition (**Siri**)
    - Recommending the best videos to millions of user (**YouTube**)
    - Beating the world’s champion in the game of Go (**AlphaGo**)

What neural networks has achieved so far?

- Near human-level image classification
- Near human-level speech recognition
- Near human-level handwriting transcription
- Improved machine translation
- Improved text-to-speech conversion
- Near-human-level autonomous driving
- Improved ad targeting
- Improved search results
- Ability to answer natural language questions
- Super-human Go playing
- And much more...

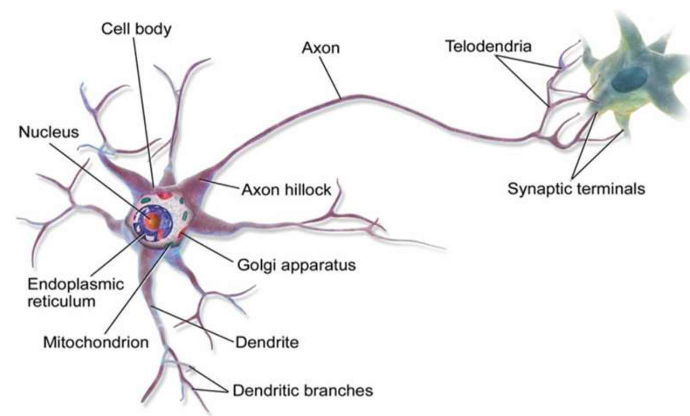
Back to other ML methods

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

bishop et al.2006 ch. 5

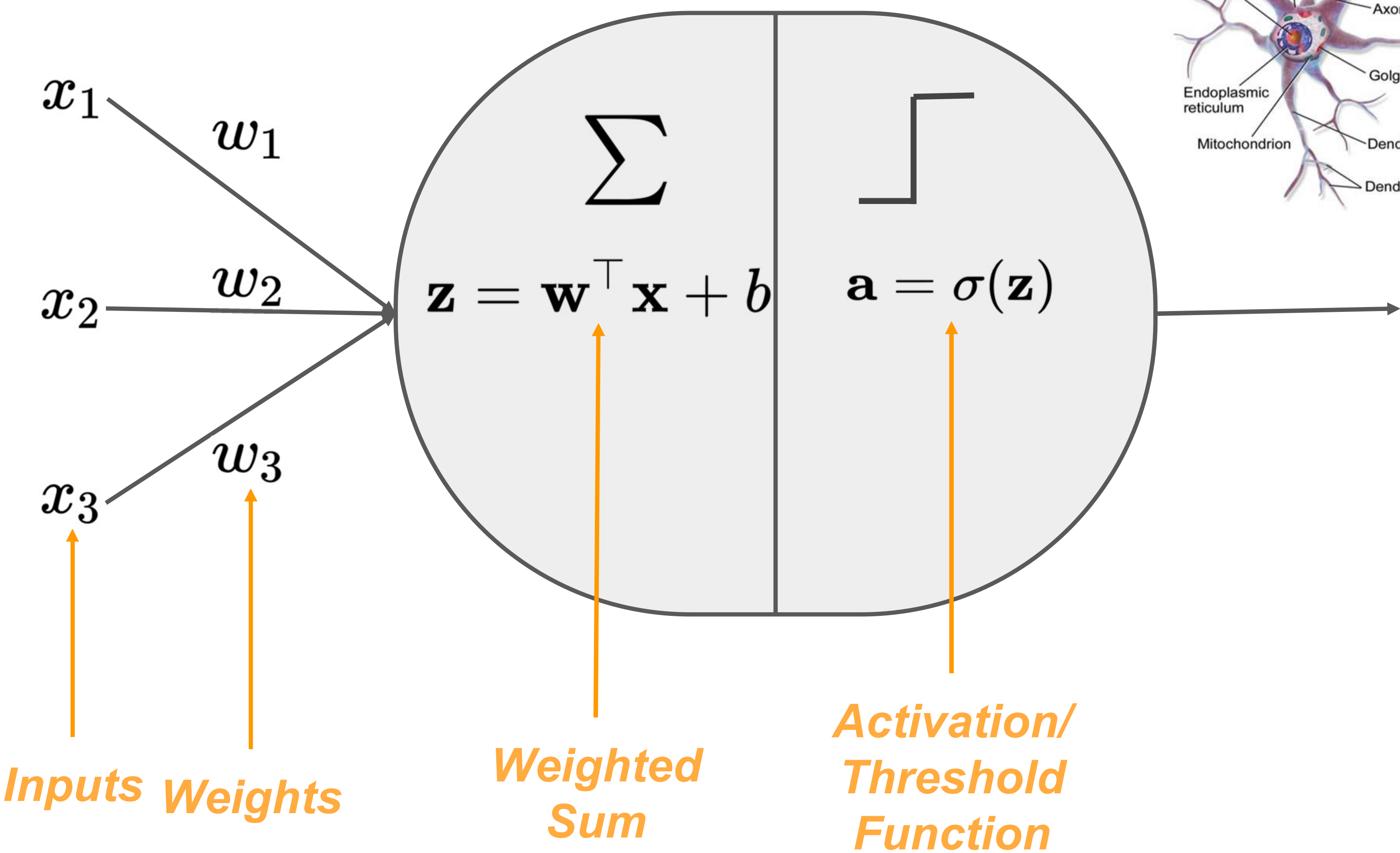
Back to other ML methods

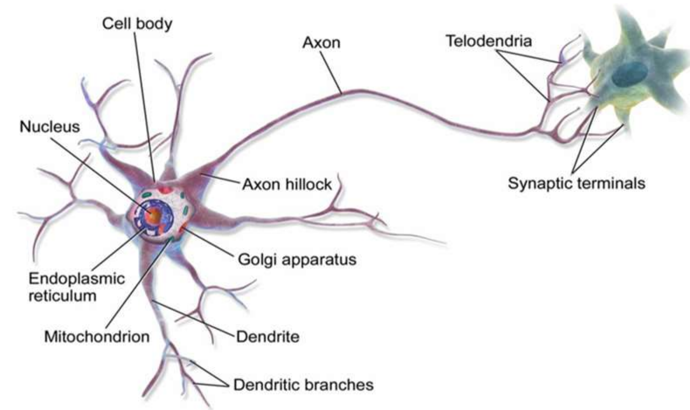
$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$



bishop et al.2006 ch. 5

$\phi_j(\mathbf{x})$





parametric forms for the basis functions
in which the parameter values are adapted during training.

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

$$\phi_j(\mathbf{x}) = h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right).$$

bishop et al.2006 ch. 5

Back to other ML methods

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

$$\phi_j(\mathbf{x}) = h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right).$$

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

bishop et al.2006 ch. 5

Back to other ML methods

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

$$\phi_j(\mathbf{x}) = h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right).$$

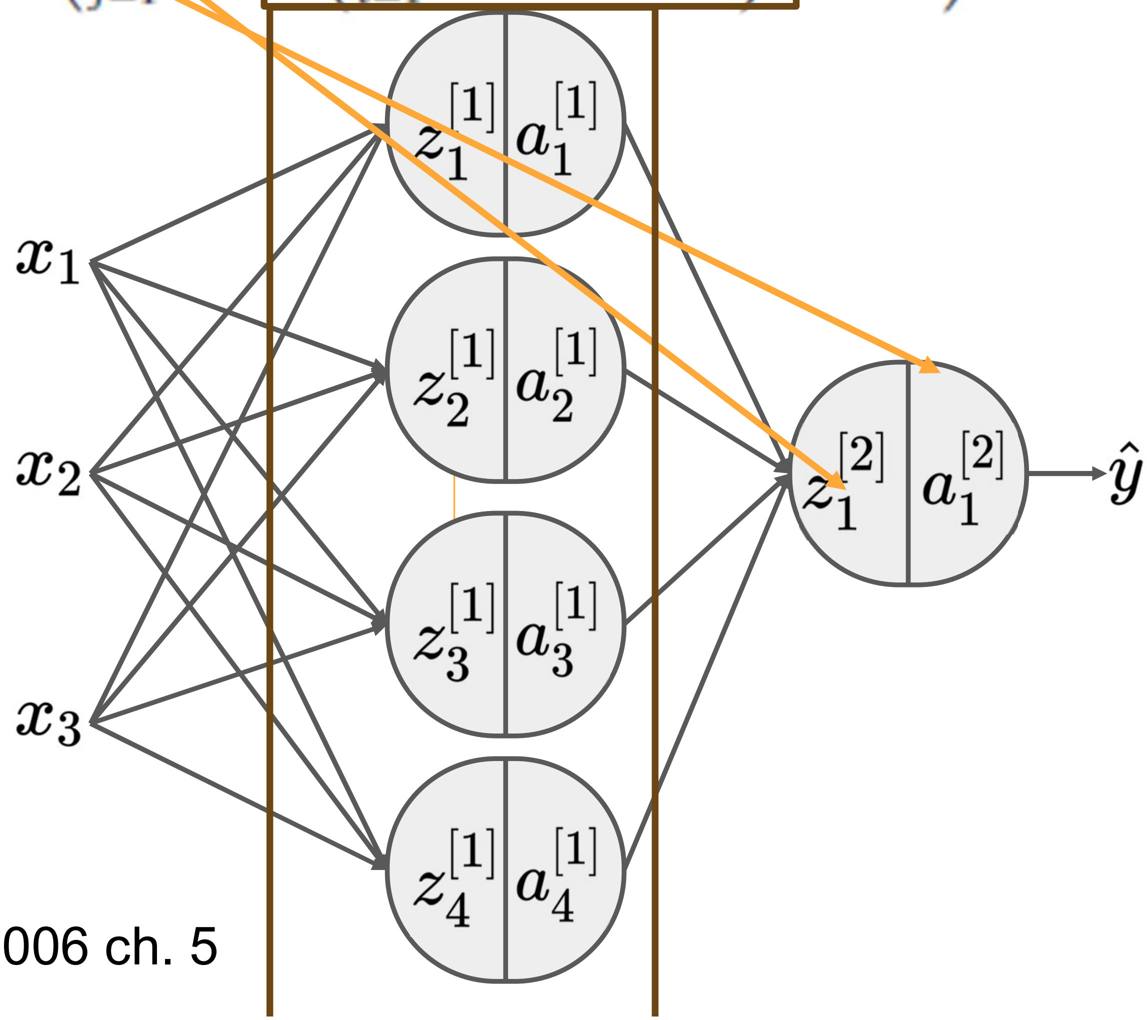
Non-linear activation function

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} \boxed{h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right)} + w_{k0}^{(2)}\right)$$

bishop et al.2006 ch. 5

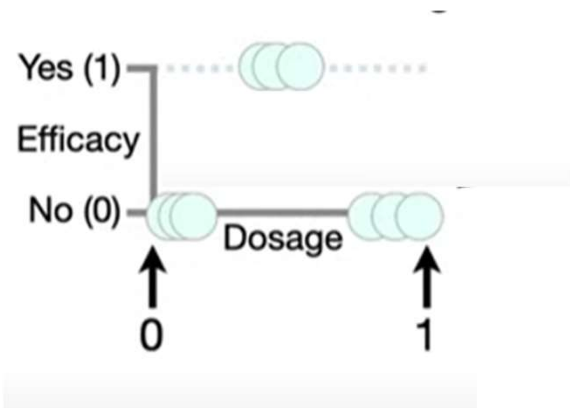
Linear transformation

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\boxed{\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}}\right) + w_{k0}^{(2)}\right)$$



bishop et al.2006 ch. 5

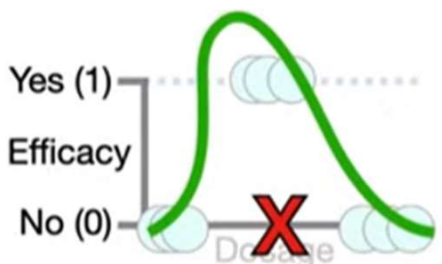
Why does NN work?!

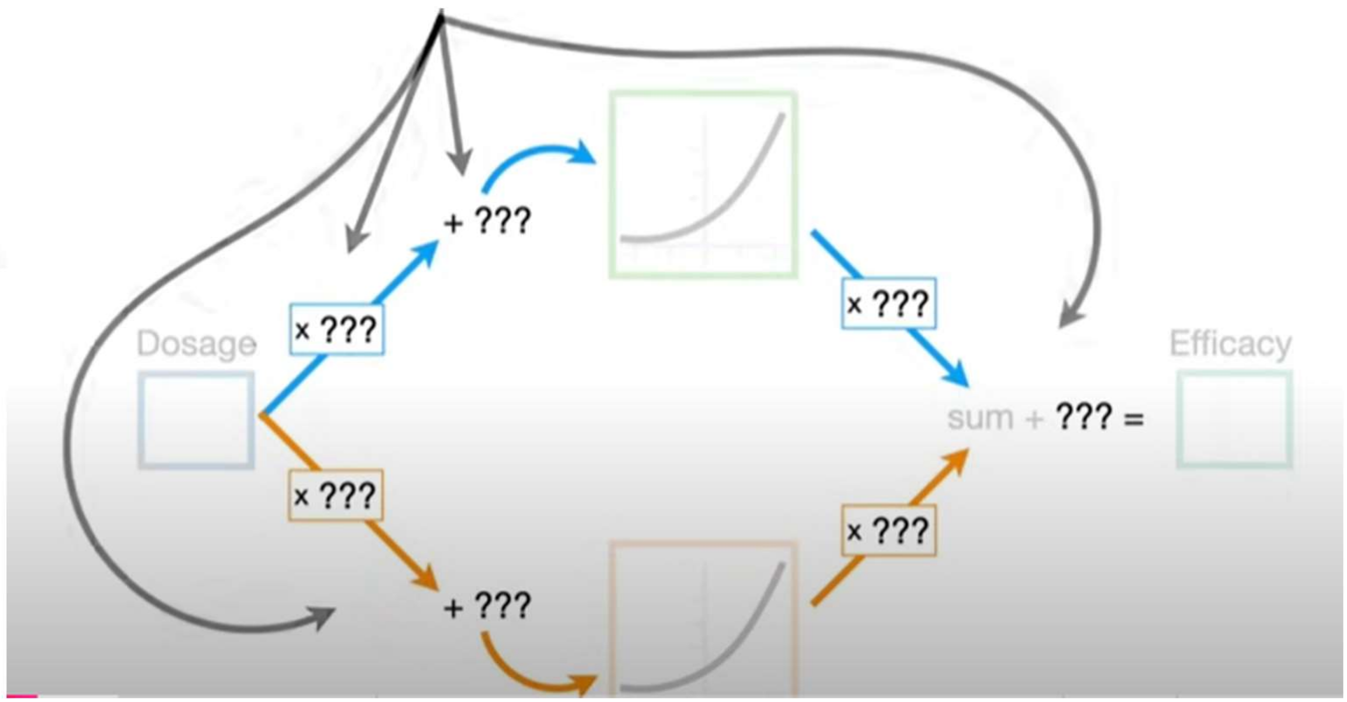


https://www.youtube.com/watch?v=CqOfi41LfDw&ab_channel=StatQuestwithJoshStarmer

Why does NN work?!

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

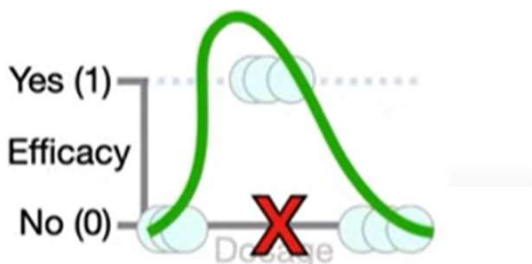


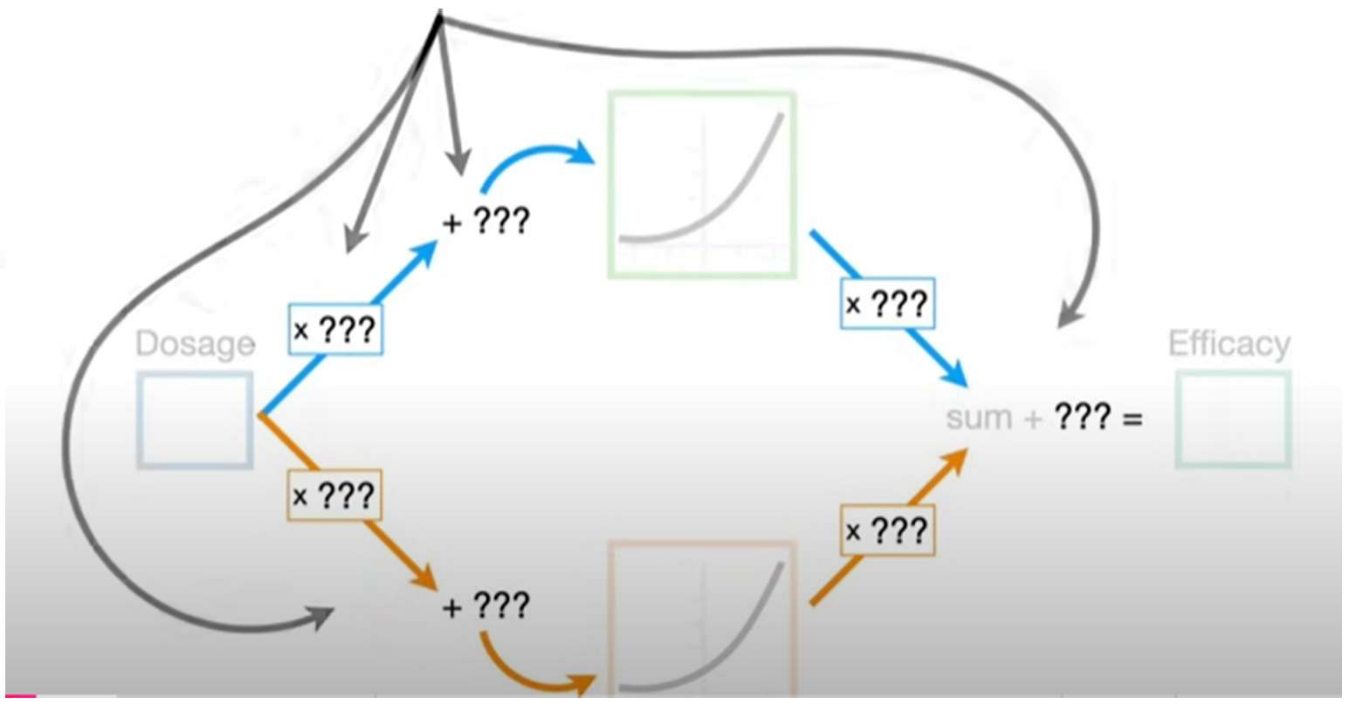


https://www.youtube.com/watch?v=CqOfi41LfDw&ab_channel=StatQuestwithJoshStarmer

Why does NN work?!

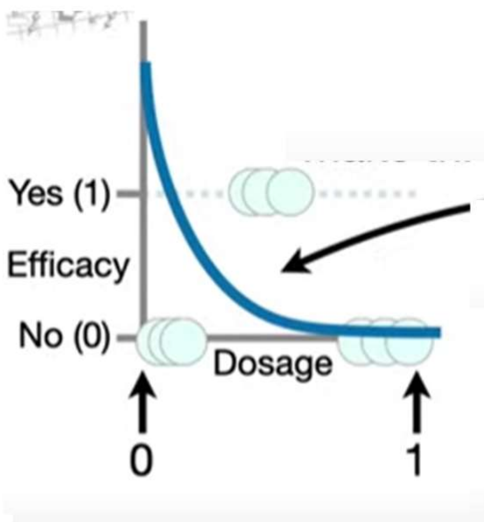
$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

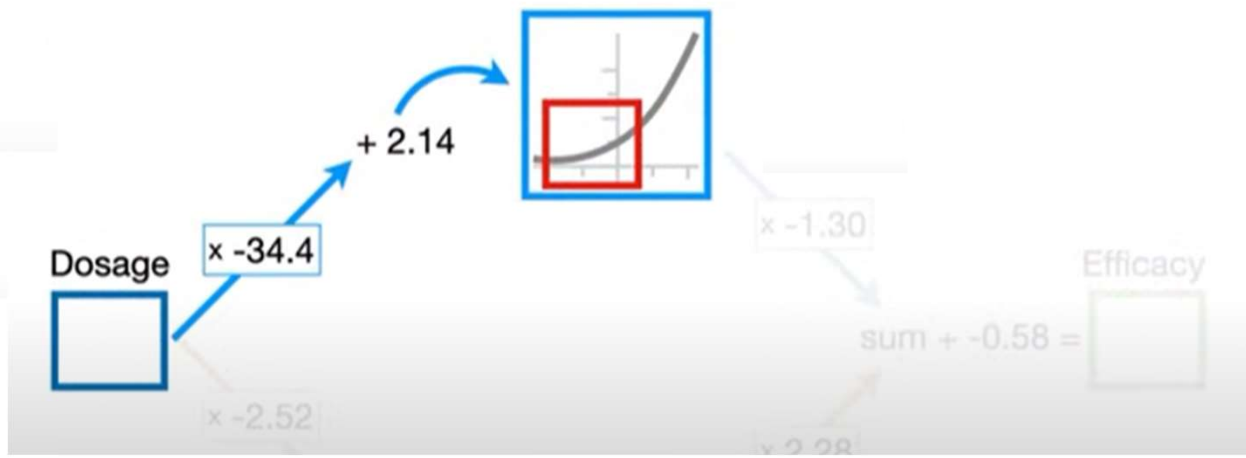




https://www.youtube.com/watch?v=CqOfi41LfDw&ab_channel=StatQuestwithJoshStarmer

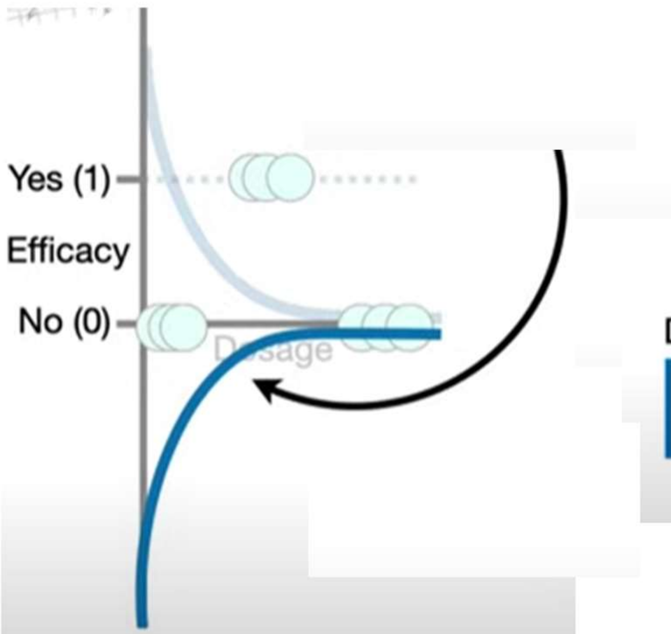
Why does NN work?!

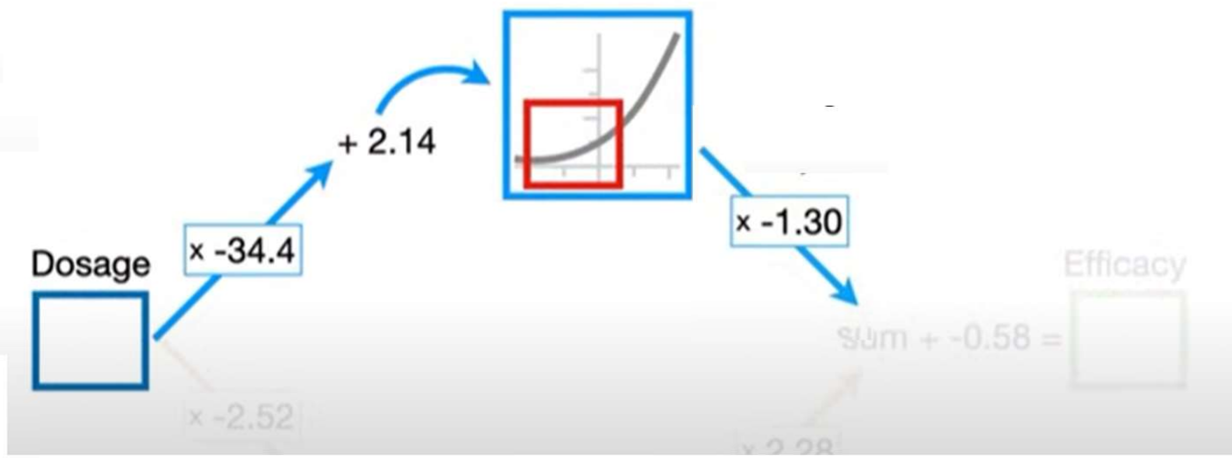




https://www.youtube.com/watch?v=CqOfi41LfDw&ab_channel=StatQuestwithJoshStarmer

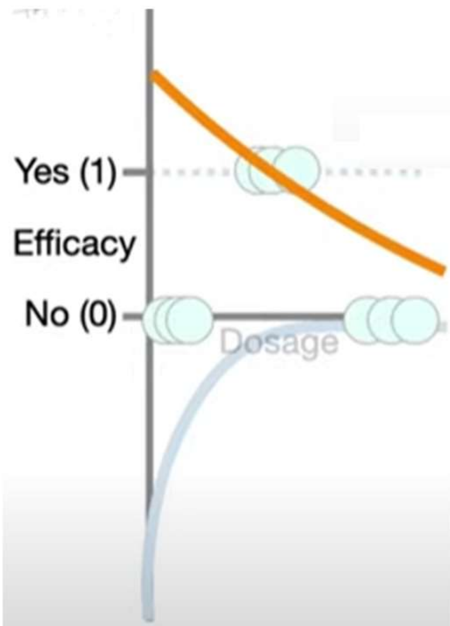
Why does NN work?!

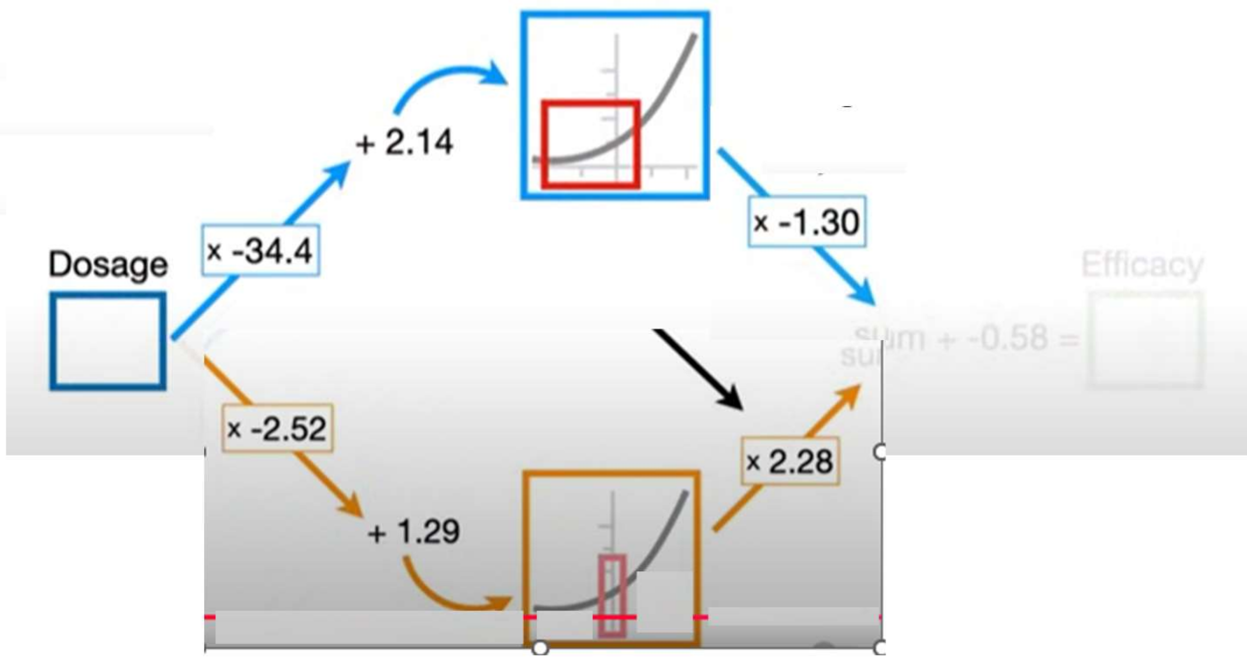




https://www.youtube.com/watch?v=CqOfi41LfDw&ab_channel=StatQuestwithJoshStarmer

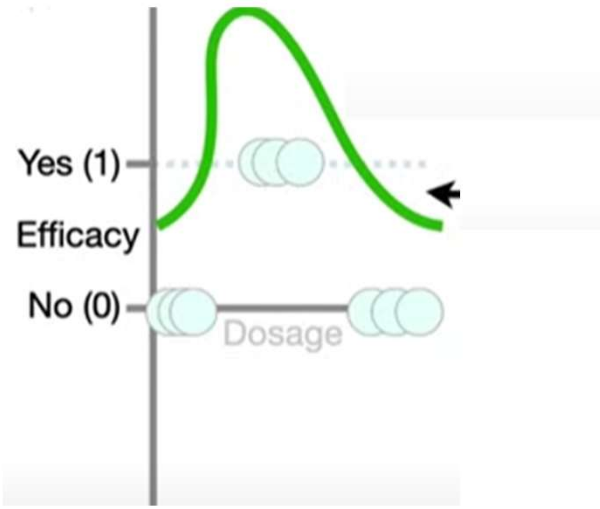
Why does NN work?!

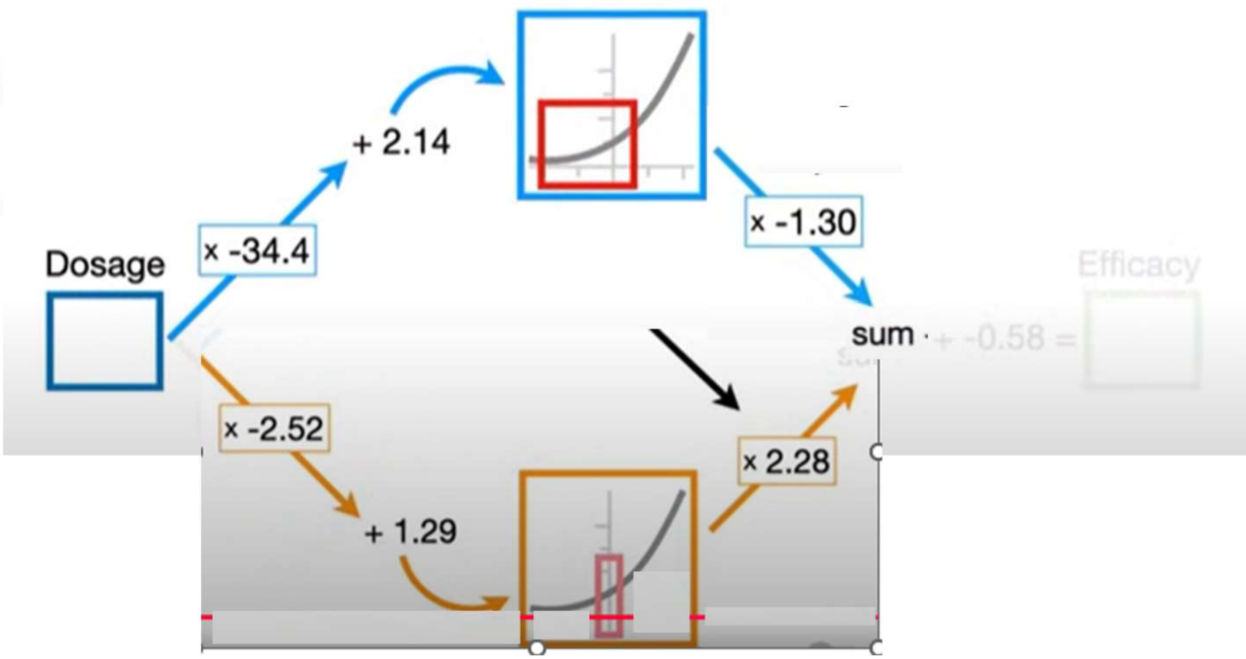




https://www.youtube.com/watch?v=CqOfi41LfDw&ab_channel=StatQuestwithJoshStarmer

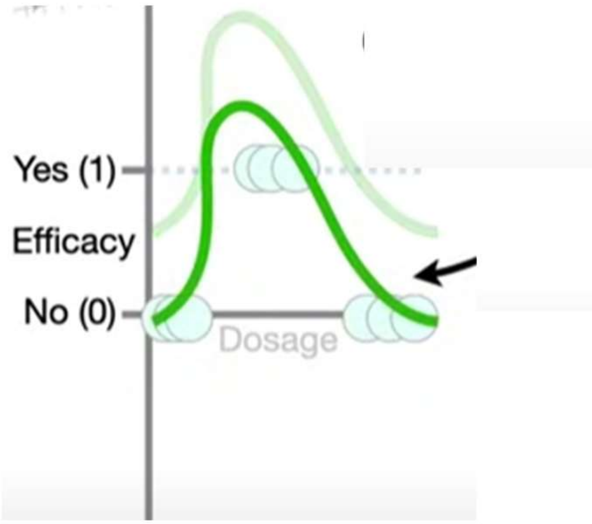
Why does NN work?!

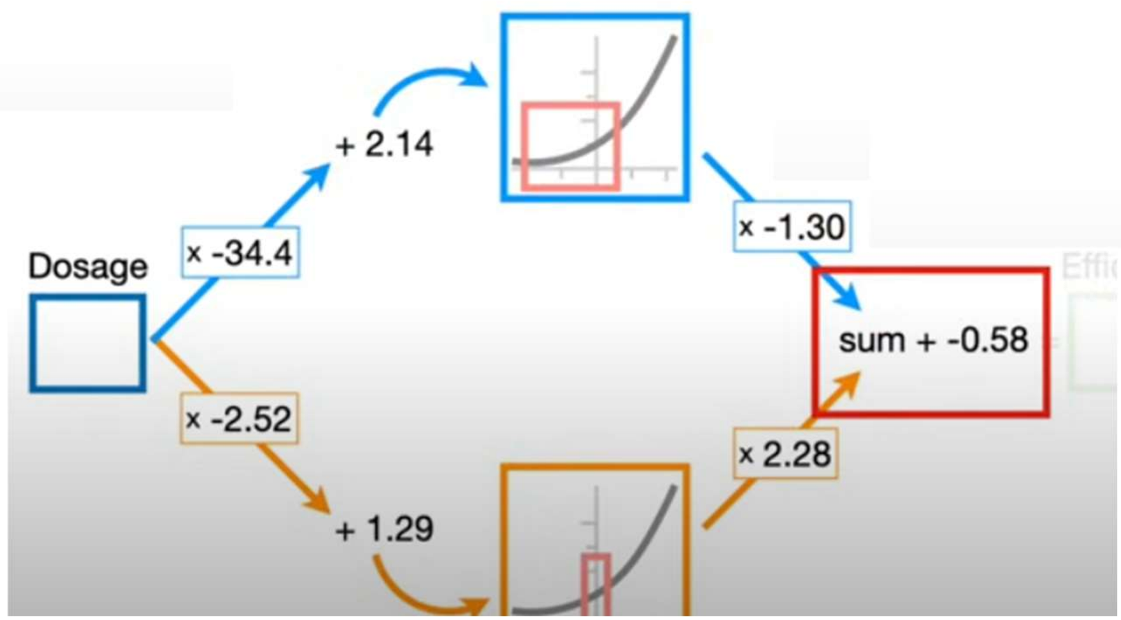




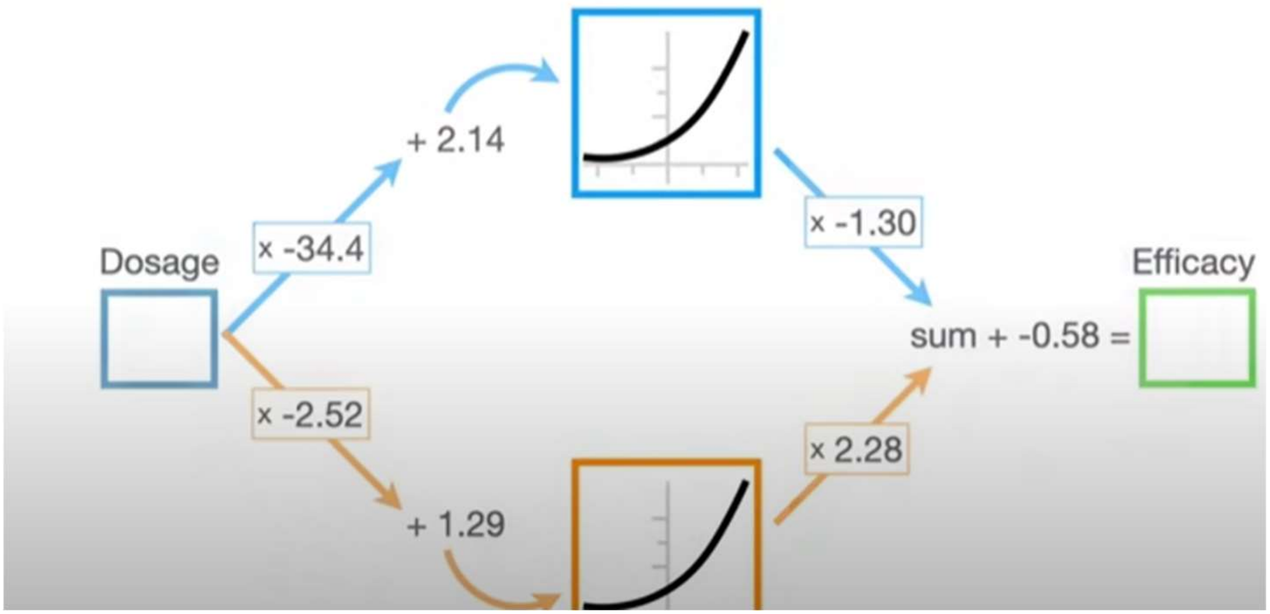
https://www.youtube.com/watch?v=CqOfi41LfDw&ab_channel=StatQuestwithJoshStarmer

Why does NN work?!





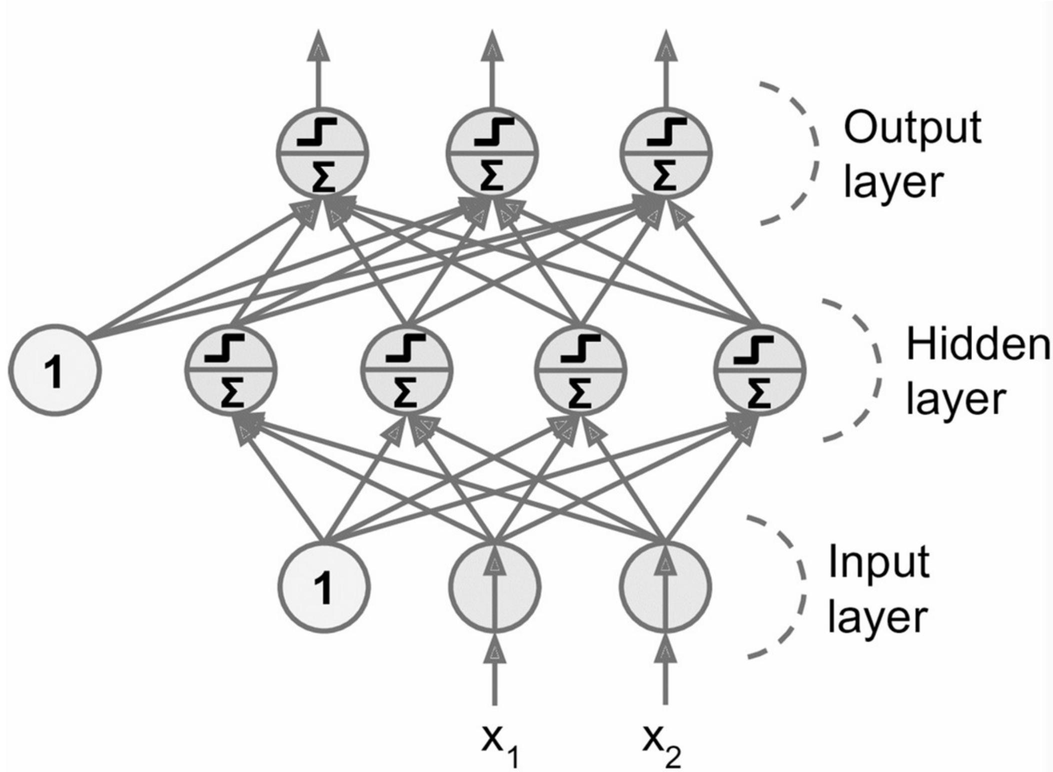
https://www.youtube.com/watch?v=CqOfi41LfDw&ab_channel=StatQuestwithJoshStarmer



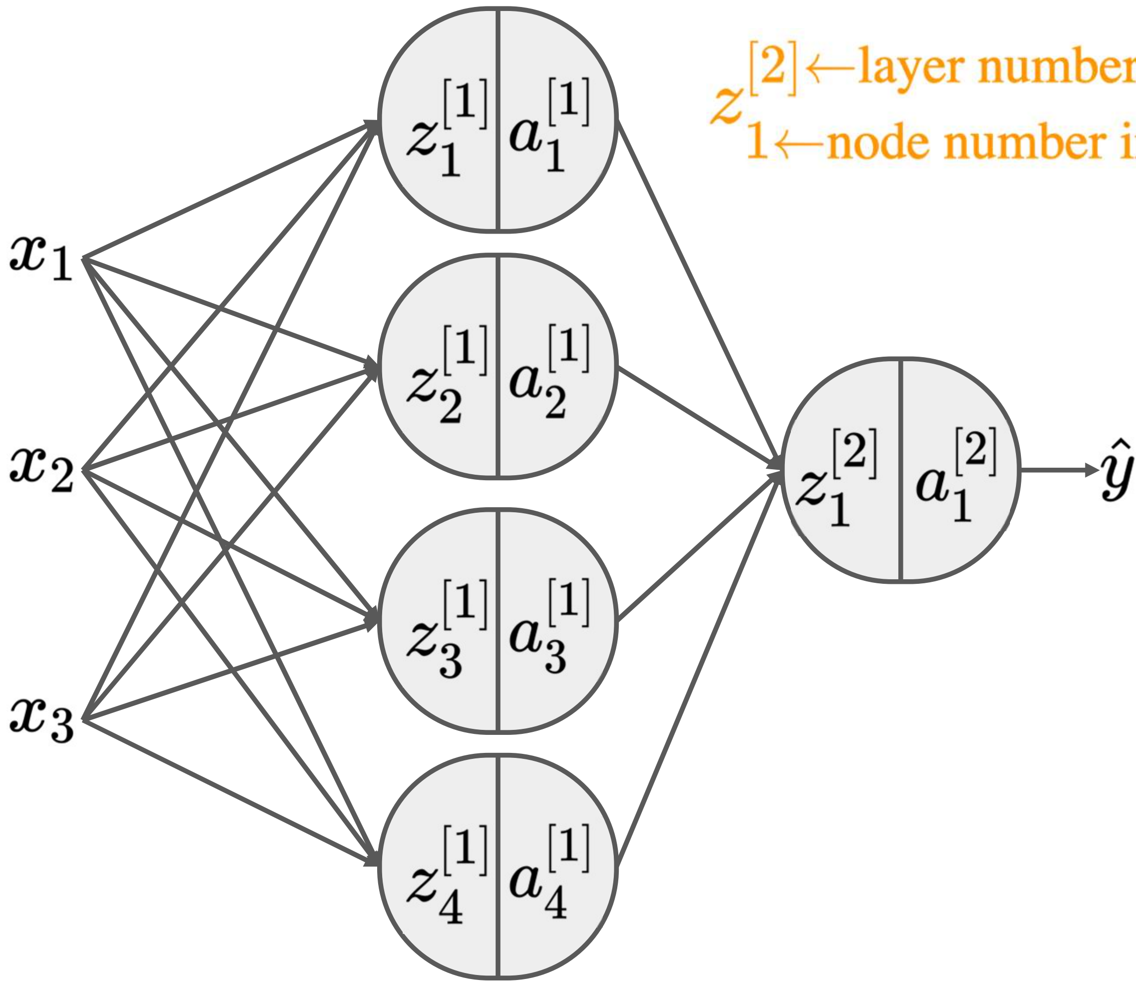
Dosage
x -34.4
+ 2.14
x -1.30
Efficacy
sum + -0.58 =
x -2.52
x 2.28
+ 1.29

Multi-Layer Perceptron (MLP) (Feed-Forward NN)

- Composed of:
  - 1 (passthrough) input layer
  - 1 or more layers of hidden layers
  - 1 final layer output layer.
- When an MLP has 2+ hidden layers, it is called **Deep Neural Network** (DNN)



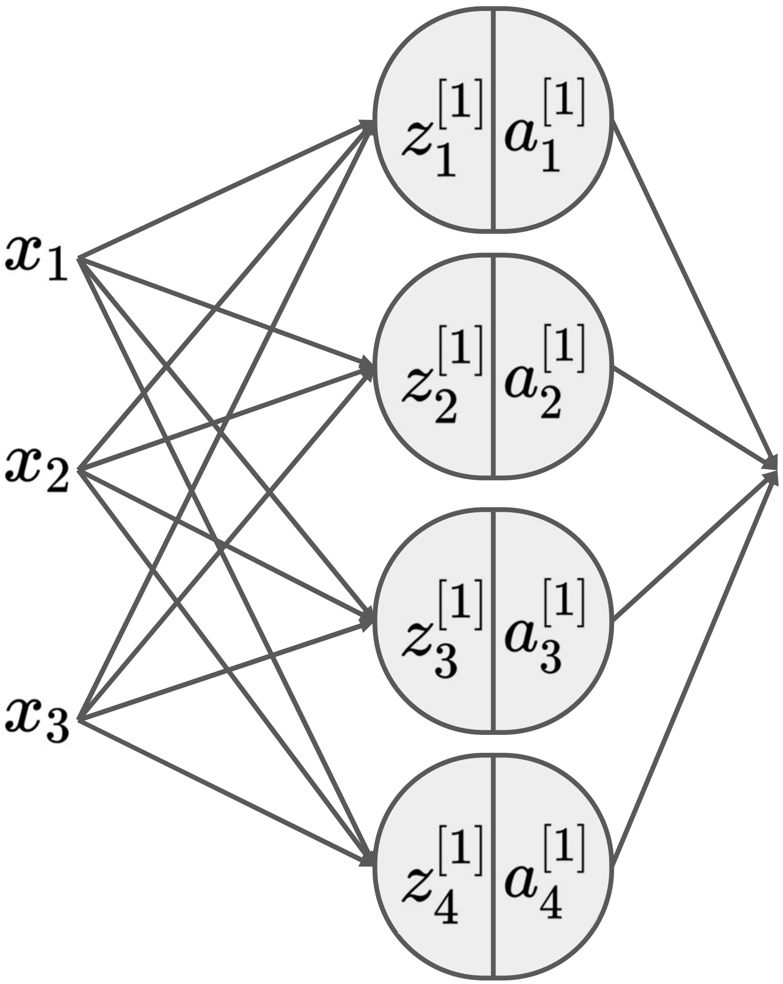
Representation of a MLP (Feed-Forward NN)



$z_1^{[2]}$ [2]←layer number, 1←node number in the layer

$$
\begin{aligned}
z_1^{[1]} &= \mathbf{w}_1^{[1]\top}\mathbf{x} + b_1^{[1]} \\
a_1^{[1]} &= \sigma\left(z_1^{[1]}\right) \\
z_2^{[1]} &= \mathbf{w}_2^{[1]\top}\mathbf{x} + b_2^{[1]} \\
a_2^{[1]} &= \sigma\left(z_2^{[1]}\right) \\
z_3^{[1]} &= \mathbf{w}_3^{[1]\top}\mathbf{x} + b_3^{[1]} \\
a_3^{[1]} &= \sigma\left(z_3^{[1]}\right) \\
z_4^{[1]} &= \mathbf{w}_4^{[1]\top}\mathbf{x} + b_4^{[1]} \\
a_4^{[1]} &= \sigma\left(z_4^{[1]}\right) \\
z_1^{[2]} &= \mathbf{w}_1^{[2]\top}\mathbf{a}^{[1]} + b_1^{[2]} \\
a_1^{[2]} &= \sigma\left(z_1^{[2]}\right) \\
\hat{y} &= a_1^{[2]}
\end{aligned}
$$

Vectorizing by stacking them vertically

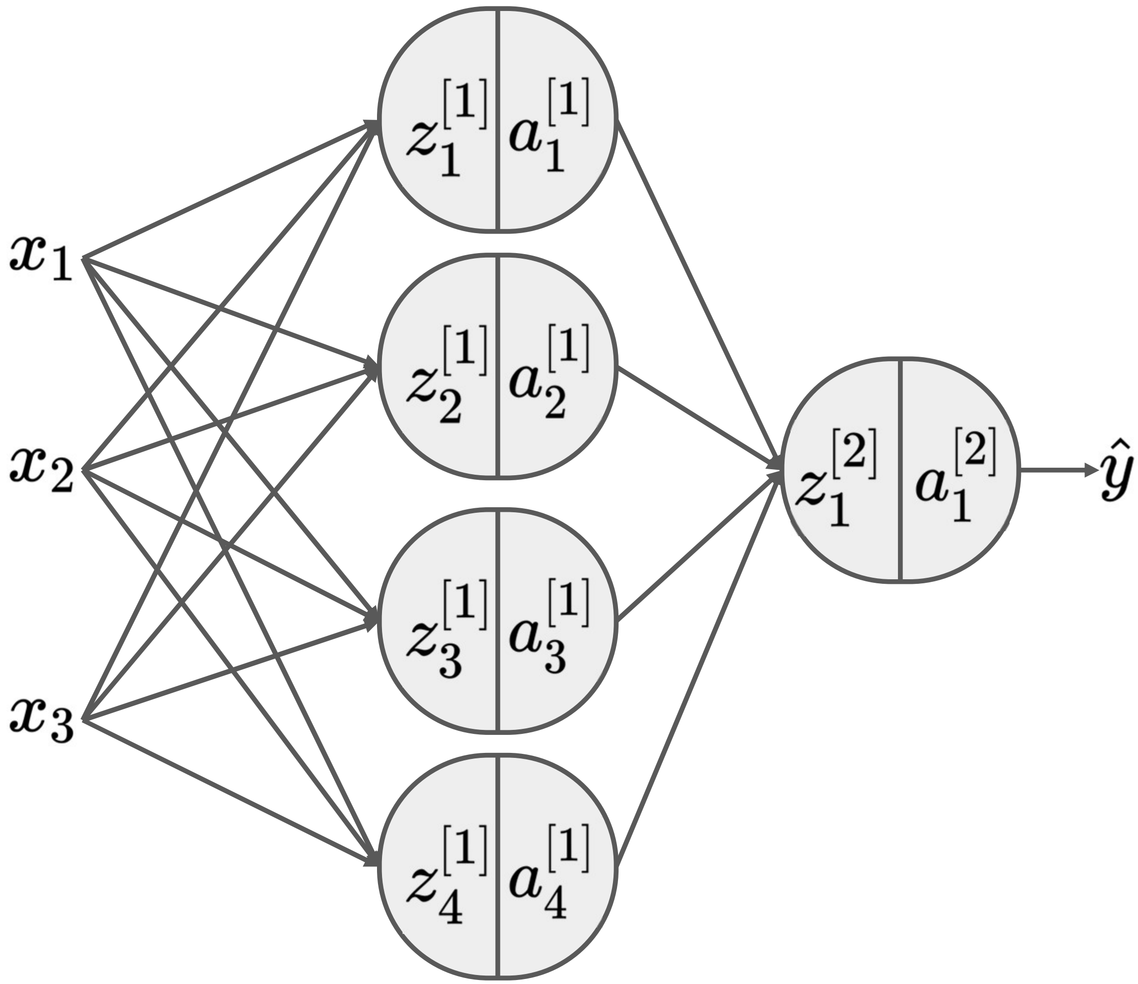


$$
\begin{aligned}
z_1^{[1]} &= \mathbf{w}_1^{[1]\top}\mathbf{x} + b_1^{[1]} \qquad & a_1^{[1]} &= \sigma\left(z_1^{[1]}\right)\\
z_2^{[1]} &= \mathbf{w}_2^{[1]\top}\mathbf{x} + b_2^{[1]} \qquad & a_2^{[1]} &= \sigma\left(z_2^{[1]}\right)\\
z_3^{[1]} &= \mathbf{w}_3^{[1]\top}\mathbf{x} + b_3^{[1]} \qquad & a_3^{[1]} &= \sigma\left(z_3^{[1]}\right)\\
z_4^{[1]} &= \mathbf{w}_4^{[1]\top}\mathbf{x} + b_4^{[1]} \qquad & a_4^{[1]} &= \sigma\left(z_4^{[1]}\right)
\end{aligned}
$$

$\mathbf{z}^{[1]}$ $\mathbf{W}^{[1]}$ $\mathbf{b}^{[1]}$ $\mathbf{a}^{[1]}$ $\sigma\left(\mathbf{z}^{[1]}\right)$

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]} \qquad \mathbf{a}^{[1]} = \sigma\left(\mathbf{z}^{[1]}\right)$$

Dimensionality of vectorized components



$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$

4x3 3x1 4x1

4x1

$$\mathbf{a}^{[1]} = \sigma(\mathbf{z}^{[1]})$$

4x1 4x1

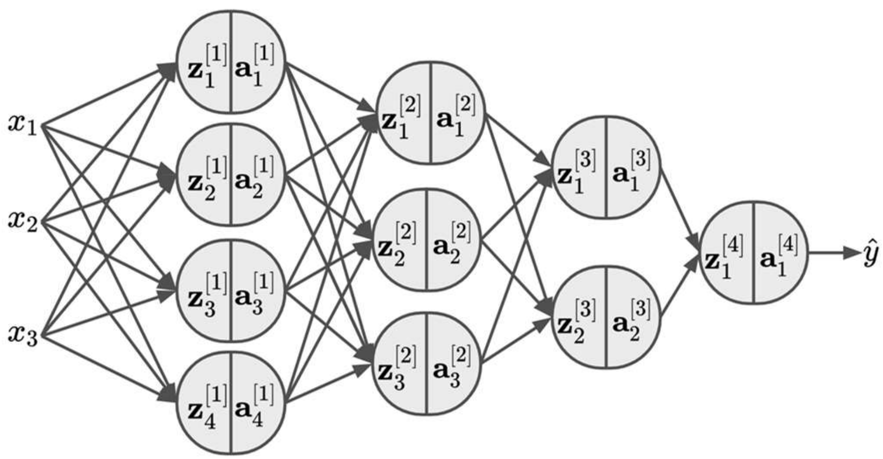
$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$

1x1 1x4 4x1 1x1

$$\mathbf{a}^{[2]} = \sigma(\mathbf{z}^{[2]})$$

1x1 1x1

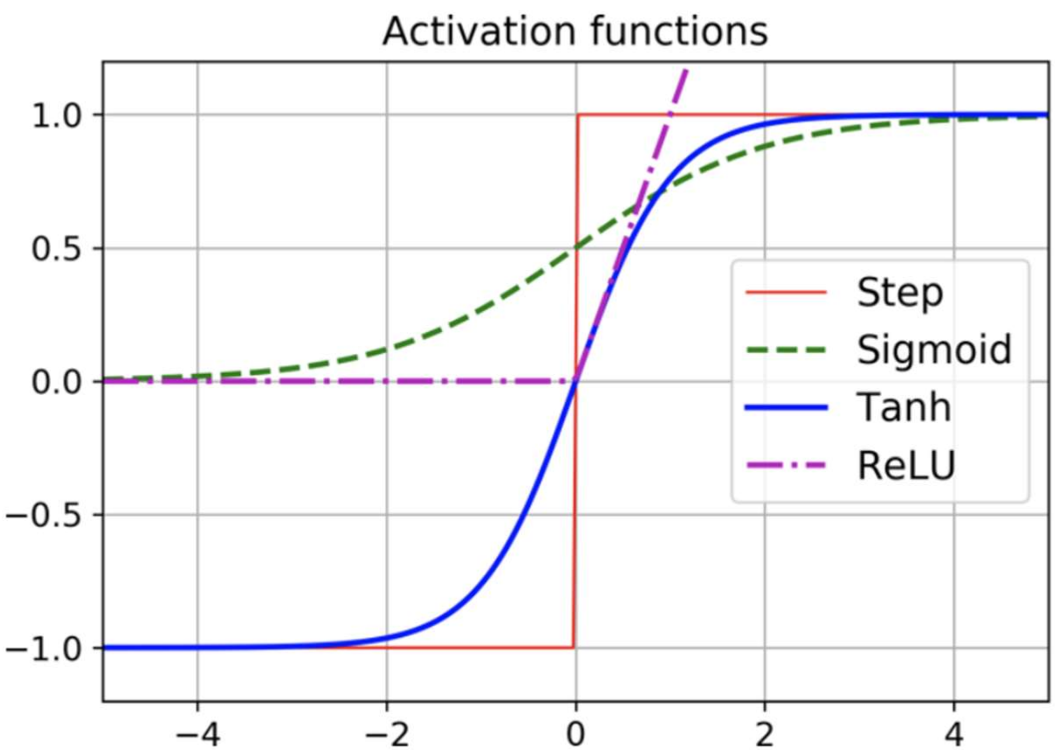
More hidden layers



3. Activation Functions

Activation function a = σ(z)

There are a number of activation functions available:



Step: $a = 1 \text{ if } z > 0,$
$-1 \text{ if } z < 0$

Sigmoid: $a = \frac{1}{1+e^{-z}}$

Tanh: $a = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

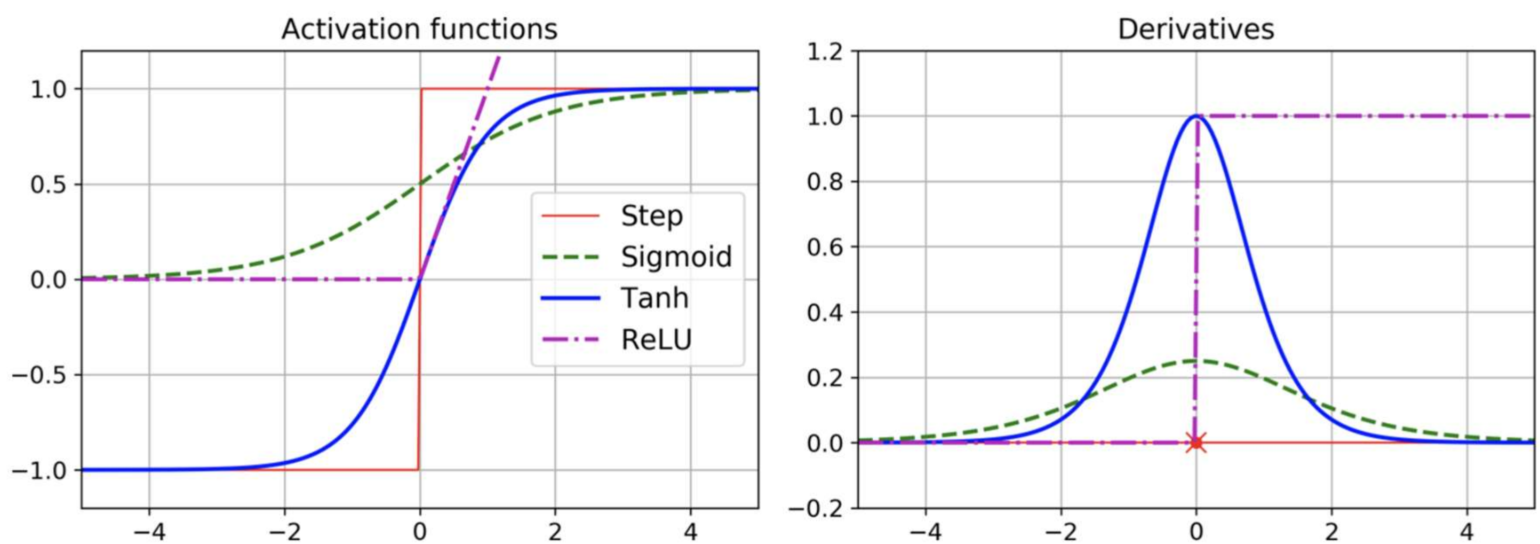
ReLU: $a = \max(0, z)$

Why activation functions are necessary

- Without activation function, each layer would consist of **linear** operations (a dot product and an addition), so the layer can only learn linear transformations of the input data.
- Adding a deep stack of linear layers would still implement a linear operation $\Rightarrow$ we need **non-linearity** to gain access to a much richer representation of the input data
- An activation function computes a **non-linear** transformation.
- Most neural networks describe the features using an affine (linear) transformation controlled by the learned parameters, followed by a fixed (non-linear) activation function.

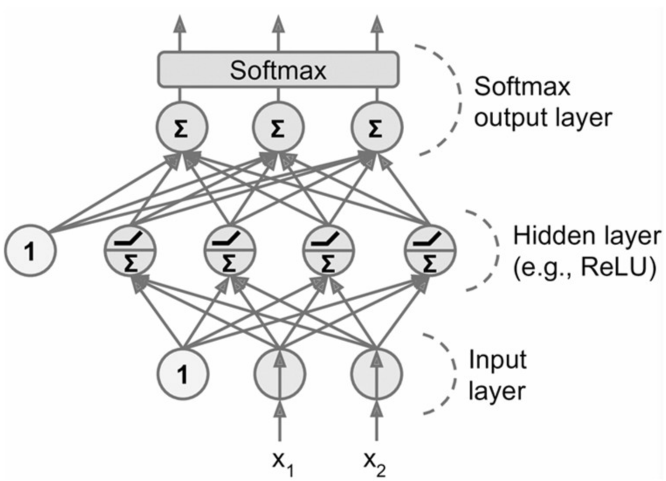
Learning with activation functions

- Step Function has **zero** derivative → does not work with Gradient Descent
- Use Sigmoid function (and other below) with well-defined non-zero derivative → allow **Gradient Descent** to make progress
- ReLU makes the derivatives **large** and **consistent** whenever it is active (> 0)



Softmax Function

- **Any time** you wish to represent a **probability distribution** over a discrete variable with n possible values, you may use the **softmax function**
- Softmax is most used as outputs, which is sum to 1, of a classifier of n classes
- Softmax *exponentiates* and *normalizes* the inputs to obtain desired outputs
- Softmax can be seen as a generalization of the sigmoid (for binary variable)
- Softmax provides a "softened" version of the **argmax** (returns a one-hot vector)

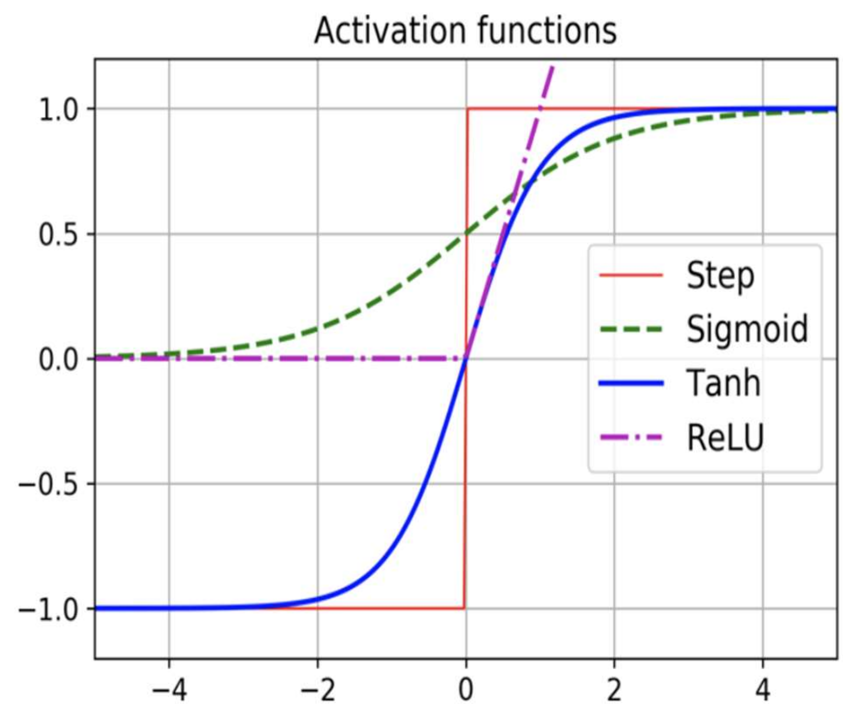


$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \text{ where } z_i = \log P(y = i|\mathbf{x})$$

$$\log \text{softmax}(\mathbf{z})_i = z_i - \log \sum_j \exp(z_j)$$
$$\approx z_i - \max_j z_j$$

Tips on Activation Functions

- **Step** function does not work with Gradient-based Learning
- **ReLU** is faster to compute and easy to optimize as its behavior is closer to linear
- **Sigmoid** and **Hyperbolic Tangent (tanh)** function saturate at 1
- Training with **tanh** is easier than **sigmoid** as it's similar to the identity function near 0
- For classification tasks, **Softmax** function is a good choice
- For regression tasks, **no need** for activation function



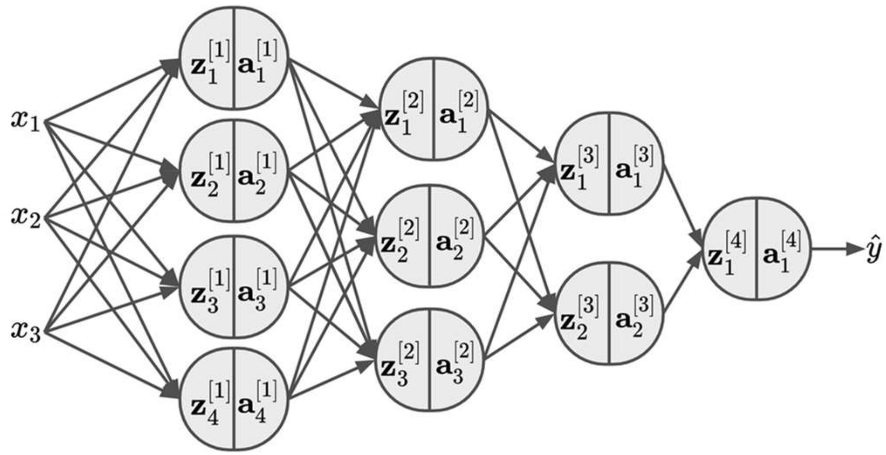
Typical MLP architecture for regression

Hyperparameter	Typical value
# input neurons	One per input feature (e.g., 28 x 28 = 784 for MNIST)
# hidden layers	Depends on the problem, but typically 1 to 5
# neurons per hidden layer	Depends on the problem, but typically 10 to 100
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU, see Chapter 11)
Output activation	None, or ReLU/softplus (if positive outputs) or logistic/tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)

Typical MLP architecture for classification

Hyperparameter	Binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression
# output neurons	1	1 per class
Output layer activation	Logistic	Softmax
Loss function	Cross entropy	Cross entropy

Summary: Network Architecture Design



- In general, most neural network architectures arrange in a *chain* structure, with each layer being a function of the layer that preceded it.
- The main consideration is choosing the **depth** of the network and the **width** of each layer
- Deeper networks often use fewer units per layer, far few parameters, but they also tend to be **harder to optimize**
- The ideal network architecture for a task must be found via *experimentation* guided by monitoring the *validation error*.

Next Lecture

Decision Trees