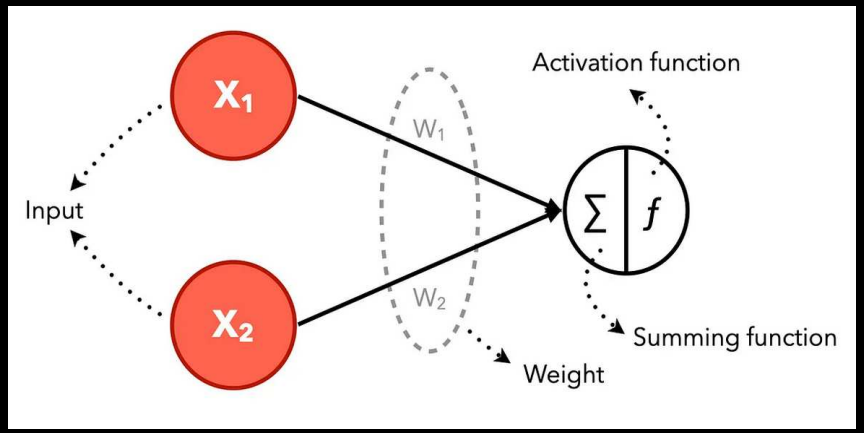
# Harvard Department of Government 2003
# Faraway Chapter 17, Neural Networks

JEFF GILL
*Visiting Professor, Fall 2024*

# Introduction to Neural Networks

- NN are a "black box" machine learning to explain some set of outcomes given a set of inputs.
- Also a classification problem.
- This means that the coefficients and values on the internal layers are not interpretable, and the only result with any value is the final result.
- The network analysis is neurological/cerebrial wherein learning is done through a serial training processes, which can be quite complex computationally.
- Unlike more conventional social science regression-style inference the (almost) only concern is explaining the outcome regardless of complexity or generalizability within.
- The most general way to think about NNs is as a two-stage regression/classification model represented by layers in a network diagram.

# What is an Artificial Neural Network

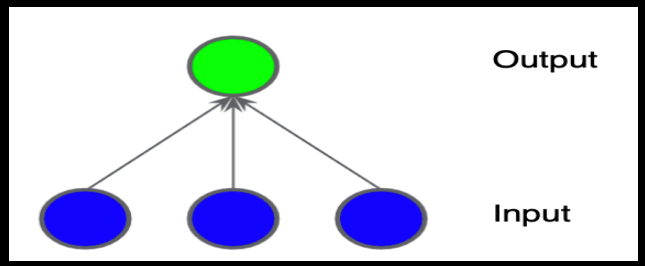



The Core Unit of a Neural Network

- *Not* a model of brain functions, but a computational method for using training data to classify and cluster on testing data at high speed
- Comprised of a node layers: an input layer, one or more hidden layers, and an output layer
- Each node (artificial neuron) connects to other nodes and this connection has an associated weight and valence threshold
- If the output of any individual node is above the specified threshold value, that node is activated sending data to the next layer of the network (otherwise not)

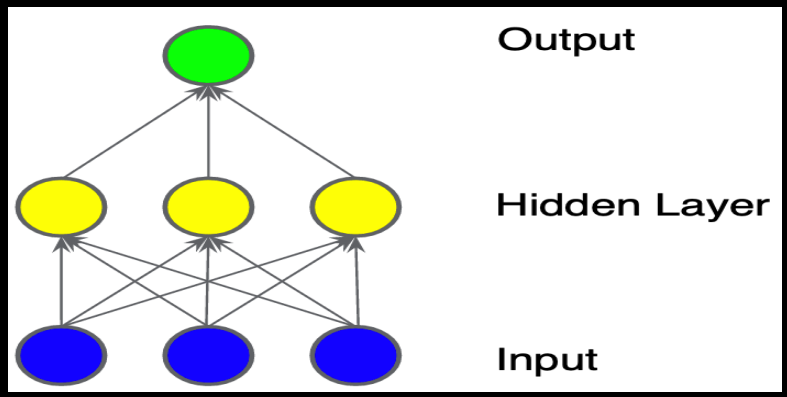
# Introduction to Neural Networks

- The starting point is a representation of a simple *linear* model as a graph.
- The blue circles are input features.
- The green circle is the *weighted* sum of the inputs, which is the output (classification) of interest).
- This looks like a causal diagram, but it is not meant to imply causality.



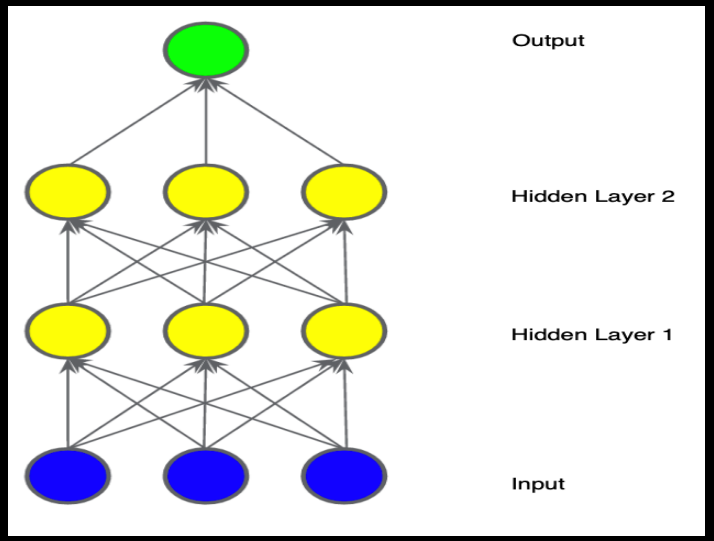
# Introduction to Neural Networks

- Now consider a single *hidden layer* of intermediate values from an intermediate weighting process that is conditional on each input features as a weighted sum of these.
- This is the yellow circles in the figure.
- The weights could be (and many are in complex models) equal to zero in the first level relationships.
- Now the output is a direct weighted linear sum of the hidden layer nodes.



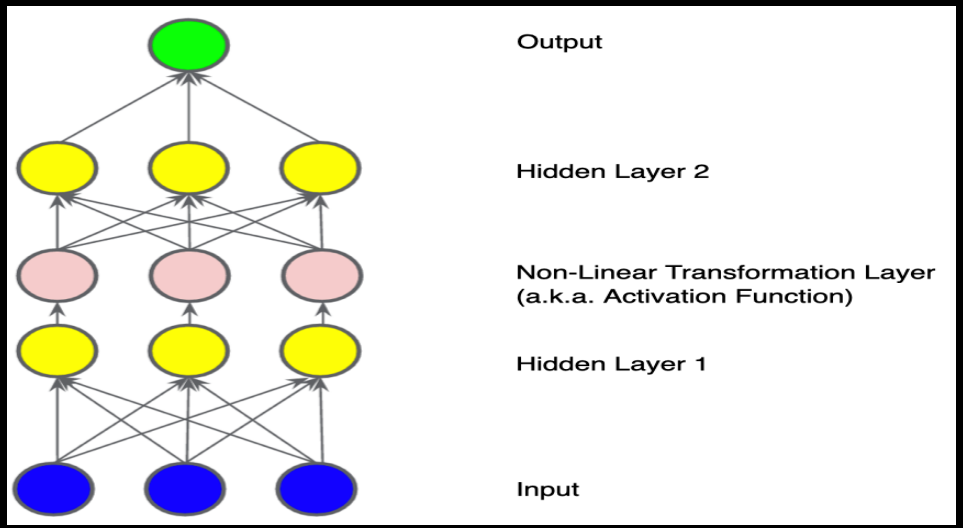
# Introduction to Neural Networks

- A single hidden layer is unusually simple for real problems, so we can have multiple iterations, each conditional on the previous hidden layer with weighting.
- This is still a linear construction involving matrix algebra.

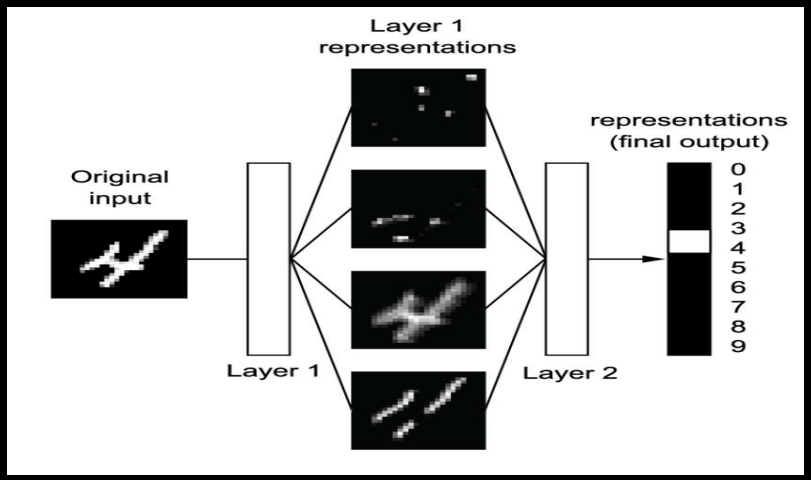


# Introduction to Neural Networks

- Of course we want to represent/solve nonlinear relationships and this is done by piping the linear weighted relationships through a nonlinear function called an *activation function.*
- In the figure there is a single layer of activation functions between two hidden layers.
- In practice we can model very complicated relationships between the inputs and the final output by stacking many linear and nonlinear layers inbetween, creating higher level functionality.



# Specific Example of a Neural Network




From: Deep Learning with Python, Second Edition by François Chollet, Manning.

# Neural Networks Development

- Phases: training, testing, validating
- The training data must have outcome labels: $\mathbf{y}$
- The testing data hides the labels and is used for prediction and comparison to training data labels: $\mathbf{y} - \hat{\mathbf{y}}$
- There are many forms of assessment for neural networks but they basically compare known outcomes to predicted outcomes in some way
- Deep learning is characterized by going backwards in the neural net constructed with the new goal of reducing $\mathbf{y} - \hat{\mathbf{y}}$ as much as possible

# Technical Basis of Neural Networks, Terms

- We start with an input matrix of information denoted $\mathbf{X}$.
- Consider a $K$-class outcome at the "top" of the diagram where the $k$th unit or category modeling the probability of this as an outcome.
- So the outcomes (targets) are denoted $Y_k, k = 1, 2, \ldots, K$, each denoted as a zero or one for a given target.
- The *derived features* in a hidden layer are created from weighted linear combinations of their inputs.
- Suppose we have a single hidden layer with $m = 1, 2, \ldots, M$ derived features denoted $Z_m$.
- Consider also an *activation function*, $\sigma$ that probabilistically or deterministically decides a connection is "turned on."

# Technical Basis of Neural Network, Steps

- For a single $X$ in $\mathbf{X}$ the target vector $Y_k$ are modeled as a linear combination of the $Z_m$ according to three steps.
- STEP 1: the derived features in the hidden layer are produced by a fitted linear regression model called the activation function:

$$Z_m = \sigma(\alpha_{0m} + X\alpha_m)$$

  where the "constant" term is called the bias unit or bias neuron and is more important in smaller models (characteristics with a value of 1 that the neural network did not previously have).
- STEP 2: the linear outcome component is produced by another fitted linear regression model:

$$T_k = \beta_{0k} + Z_m\beta_k$$

- STEP 3: to be as general as possible, allow a final transformation of the vector of outputs $T$:

$$f(\mathbf{X}) = g_k(T_k)$$

# Technical Basis of Neural Network, Details

- For regression the $g_k(T_k)$ function can be just the identity, but the softmax function is preferred because it provides estimates that sum to one:

$$g_k(T_k) = \frac{\exp(T_k)}{\sum_{\ell=1}^{K} \exp(T_\ell)}$$

  (notice that this is really logit).

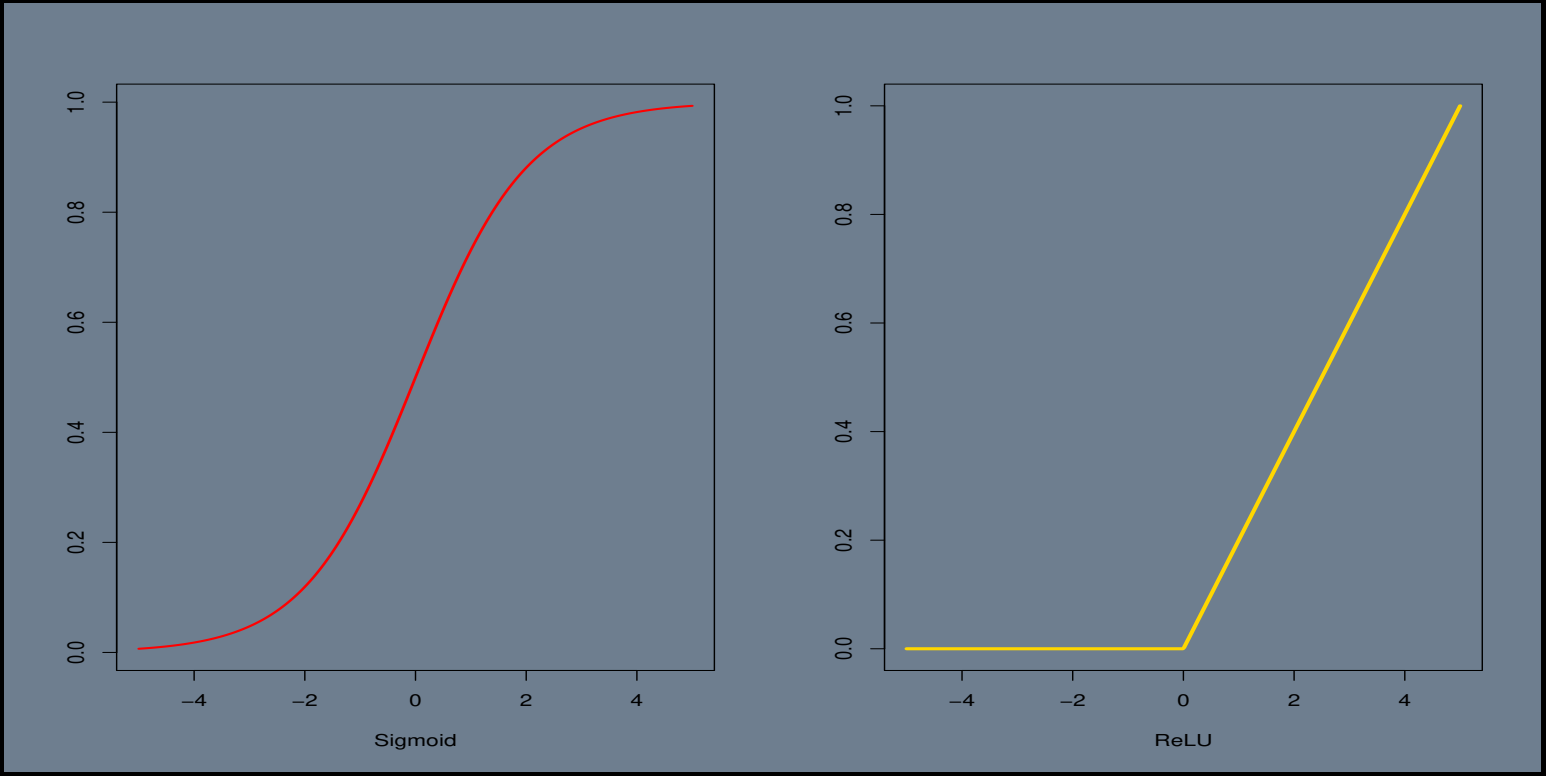
- The $Z_k$ in the middle are referred to as hidden units because they are not directly observable.

- The *activation function* ($\sigma$) is aptly named since they determine whether there will be an edge (connection) between a given $X$ and a given $Z$.

- If $\sigma$ is just the identity function then STEP 1 and STEP 2 just collapse to a simple linear model in the inputs.

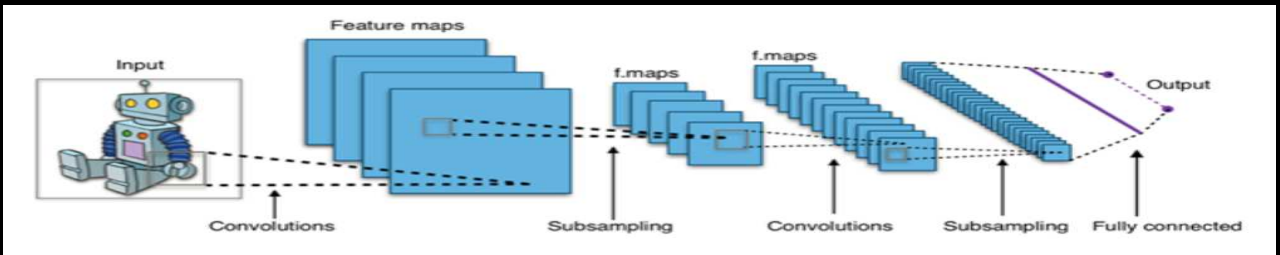
# Activation Functions

- There are two very common forms which are really like CDFs.
- *Sigmoid* (logit), $F(x) = [1 + \exp(-x)]^{-1}$.
- *Rectified Linear Unit* (ReLu), $F(x) = \max(0, x)$.
- As with anything in this realm there are many alternatives in use.



# Convolutional Neural Networds

- Thus far the assumption is that neural networks use only *feature vectors* from the raw data.
- This means that processed data arrive all at the same time with equal status for analysis.
- Suppose instead we to integrate the feature generation phased into the training phase of a NN.
- This means that it learns the features from the data together with the parameters of the neural network and not independently.
- This defines a CNN, where the most common application are to images and video.




From: Aldo Faisal, Cheng Soon Ong, and Marc Peter Deisenroth.

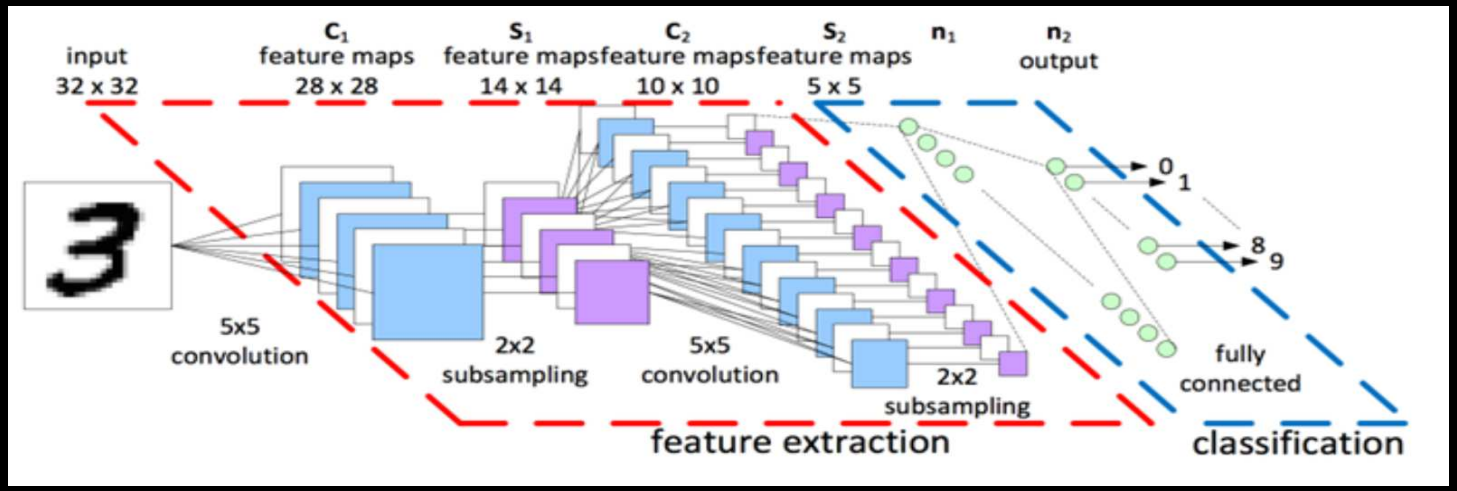
# Convolution Advantages with Images

- Suppose we have an image with $256 \times 256$ pixels.
- If we vectorize this it is an input vector of size $65536$ in $\mathfrak{R}$.
- Suppose we have $1,000$ neurons in the first layer, then the number of involved parameters would be close to $65,000,000$.
- Also vectorizing loses one entire dimension worth of information about how pixels are interrelated within the image.
- Using convolutions solves both the size problem and the vectorization problem.

# Sliding

- CNNs get their name from the idea of sliding (convolving) a small, manageable window over the full data image.
- By sliding over regions of the text we manage the large data problem by considering subsets, and we manage the relationshipal issue by seeing adjacency in two dimensions (or possibly more).
- There are two main components to this process:
  - convolutional layers comprise neurons that scan their input for patterns
  - downsampling layers, which are pooling layers used to reduce the feature map for efficiency.

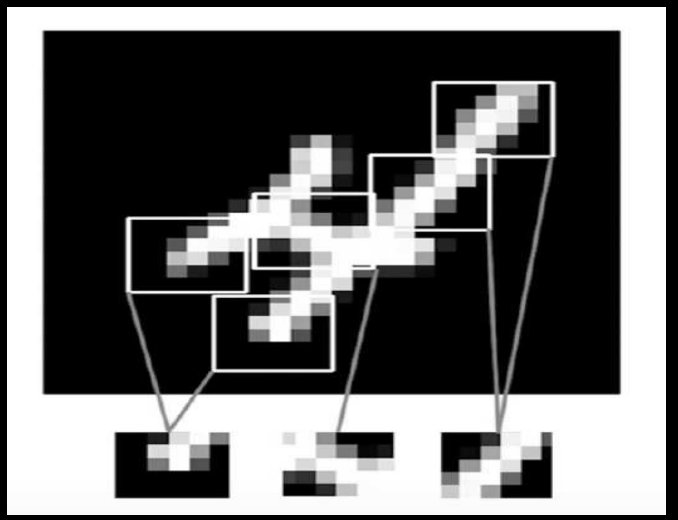
# Example Convolutional Neural Network Feature Extraction




From: Aldo Faisal, Cheng Soon Ong, and Marc Peter Deisenroth.

# Specific Convolution Operation

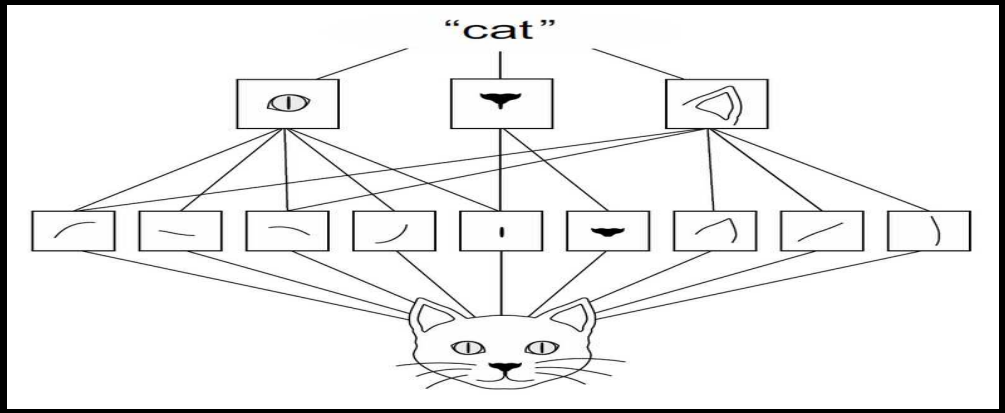
- The dense layers in regular neural networks learn global patterns in their input feature space.
- Conversely convolution layers learn local patterns and then assemble that knowledge.



# Pattern Details

- The patterns that CNN learn are translation-invariant: after learning a certain pattern in the lower-right corner of a picture, a CNN can recognize it anywhere.
- For example, in the upper-left corner a densely connected model would have to learn the pattern anew if it appeared at a new location.
- This makes CNNs data-efficient when processing images because the visual world is fundamentally translation-invariant: they need fewer training samples to learn representations that have generalization power since sub-images repeat..
- Also CNNs can learn spatial hierarchies of patterns.
- A first convolution layer will learn small local patterns such as edges, a second convolution layer will learn larger patterns made of the features of the first layers, and continue.
- This aspect allows CNNs to highly efficiently learn increasingly complex and abstract visual concepts since visuals are fundamentally spatially hierarchical.

# Symbolic Illustration

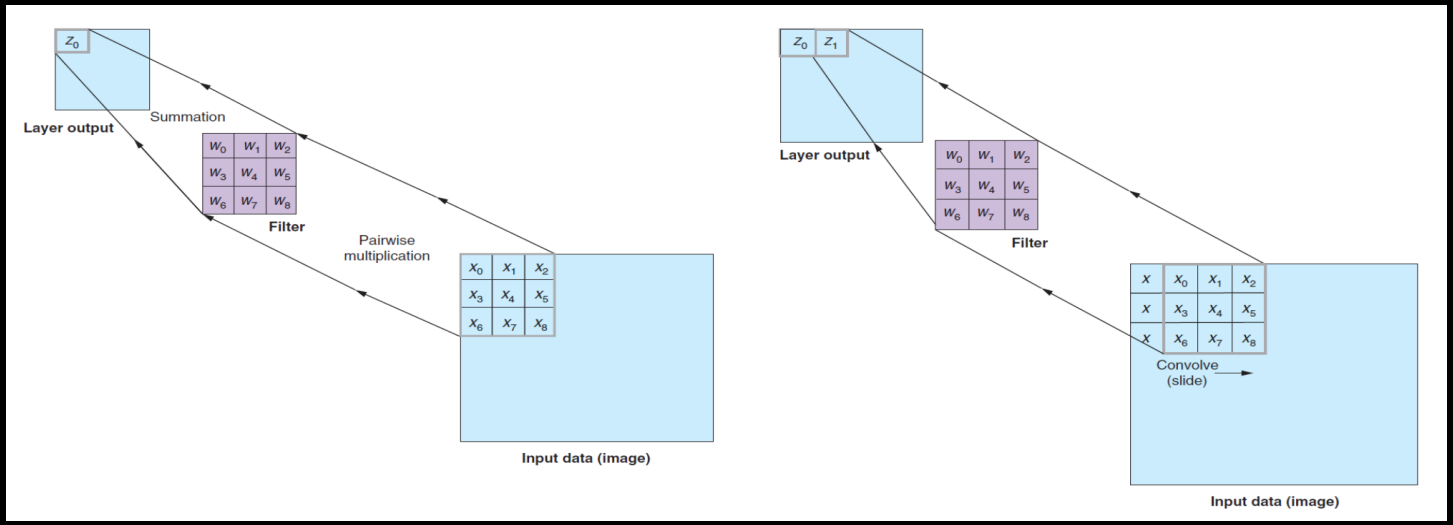



From Chollet, Francois. *Deep learning with Python.* Simon and Schuster, 2021.

# The Convolution Step

- The neurons of the hidden layer can be considered as elements of a two-dimensional array.
- Each of these neurons corresponds to a single parameter that is associated with the corresponding element of the array.
- To perform convolutions we slide the filter completely over the input matrix.
- This means that filters have two components:
  - ▻ a set of weights that are exactly like the regular weights that feed into neurons
  - ▻ an activation function as described previously.
- These filters are usually $3 \times 3$ in size, but other shapes can be specified.
- Note that each filter in a convolutional neural net is unique, but each filter element is fixed within an image subset.
- The layer output or output matrix is called the feature map array.

# Example Convolutional Neural Network Feature Extraction




From: Aldo Faisal, Cheng Soon Ong, and Marc Peter Deisenroth.

# CNN Algorithmic Details

- Each hidden layer corresponds to a single or multiple filter matrix.
- The parameters in the hidden layer are shared by all the input pixels but we do not have a dedicated set of parameters per input element (pixel).
- The outputs of the hidden layer encode local neighborhood correlation information from the various areas withing the input image.
- The output is also an image array so we can think of it as the input to a second hidden layer and in this way built a network with many layers each one performing convolutions.

# CNN Algorithmic Details

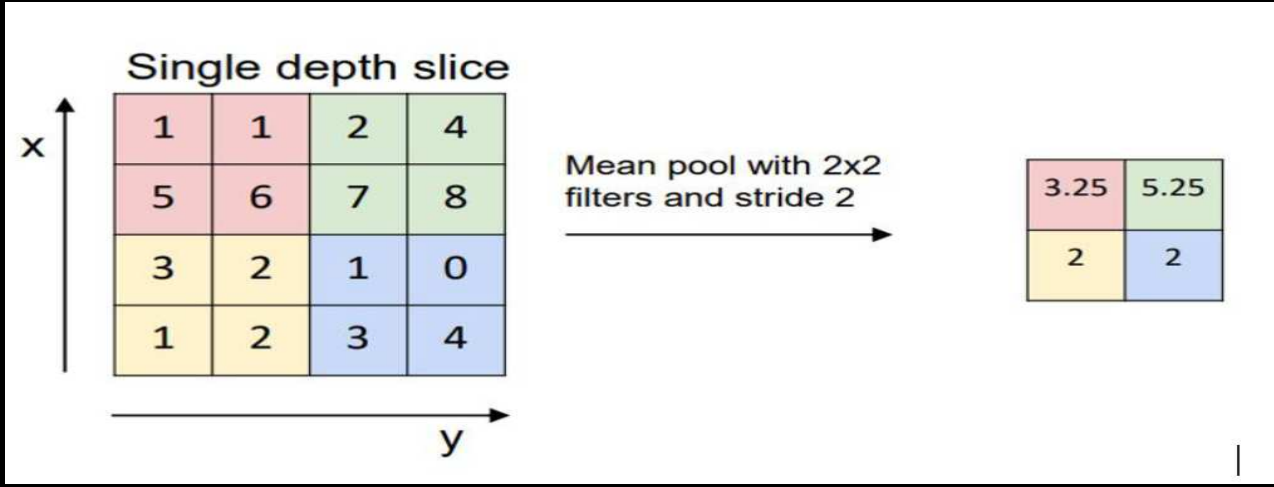
- Convolutions operate over rank-3 tensors called feature maps, with two spatial axes (height and width) as well as a depth axis (also called the channels axis).
- For a given RGB image, the dimension of the depth axis is 3 since the image has three color channels: red, green, and blue.
- For a black-and-white picture 1 given by levels of gray.
- The convolution operation extracts patches from its input feature map and applies the same transformation to all of these patches, producing an output feature map.
- The filter window size is a parameter in the model chosen by the researcher and is highly conditional on the image at hand.

# CNN Algorithmic Schematic

# CNN Pooling

- Spatial pooling (Down sampling) reduces the dimensionality of each map but retains critical information.
- The pooling operation consists of choosing a single value to represent all the pixels that exist within the window being considered.
- *Max Pooling:* among all pixels that lie withing the window, the one with the maximum pixel value is selected.
- *Mean Pooling:* average all pixel values within the window as the selection.
- Pooling means that the representation can become approximately invariant to small translations of the input.

# Mean Pooling




From: Aldo Faisal, Cheng Soon Ong, and Marc Peter Deisenroth.

# Convolutional Steps

- There are $n$ filters and $n$ new images now.
- The question is how to handle that.
- Most often with neural networks with start with a labeled dataset as a training exercise.
- The goal is then to predict a label (identifier) for a new image being considered.
- The easiest next step is to take each of those filtered images and string them out as input to a feed-forward layer.
- No matter how many layers (convolutional or non-convolutional) one adds to the network, once there is a final output one can compute the error and backpropagate that error all the way back through the network.

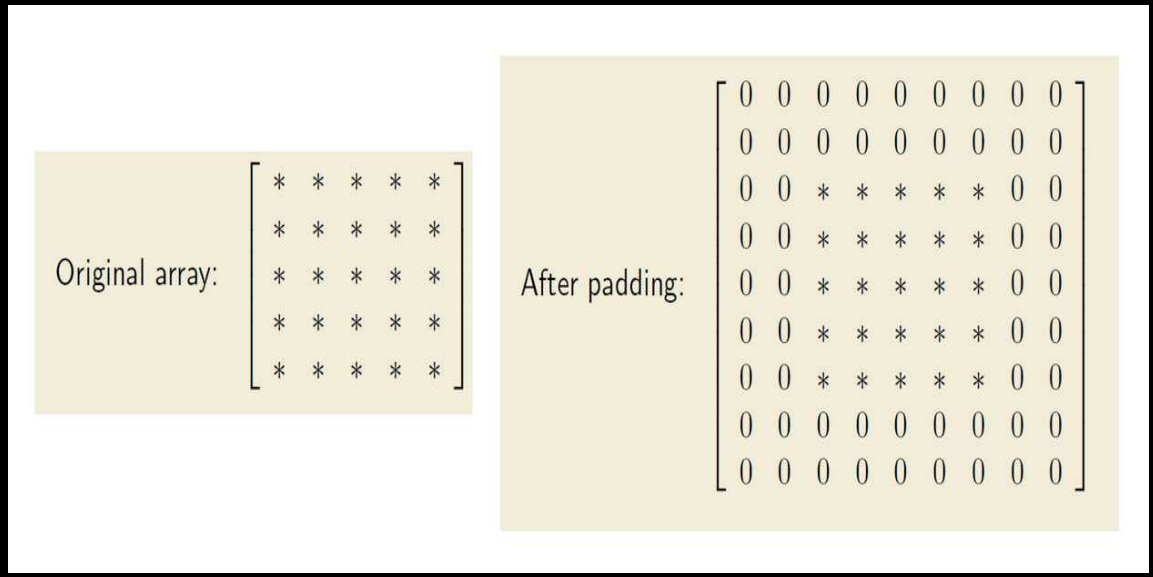
# Stride

- This the technical term for the step size in the convolutional process.
- The distance traveled during the sliding phase is a tuning parameter.
- It shouldn't be as large as the filter itself.
- Each individual snapshot usually has an overlap with its neighboring snapshot by design.
- The distance each convolution travels across the image is known as the stride and is usually set to 1.
- Moving one pixel or anything less than the width of the filter will create overlap in the various inputs to the filter from one position to the next.
- Larger strides that have no overlap between filter applications will lose the blurring effect of one pixel or in one case relating to its neighbors.

# Padding

- Suppose we start with a $3 \times 3$ filter in the upper-left corner of an input image and stride one pixel at a time across, stopping when the rightmost edge of the filter reaches the rightmost edge of the input.
- The output image will be two pixels narrower than the source input.
- The problem with strategy is that the data in the edge of the original input is under-sampled as the interior data points are passed into each filter multiple times from the overlapped filter positions.
- On a large image, this may not be a critical issue, but as soon as one uses this on for example a Tweet, under-sampling a word at the beginning of a 10-word dataset could drastically change the outcome.
- The solution strategy is padding: adding enough data to the input's outer edges so that the first real data point is treated just as the innermost data points are.
- The shortcoming of this strategy is that it is adding potentially unrelated data to the input, which in itself can change the outcome.
- There are strategies for minimizing this effect.

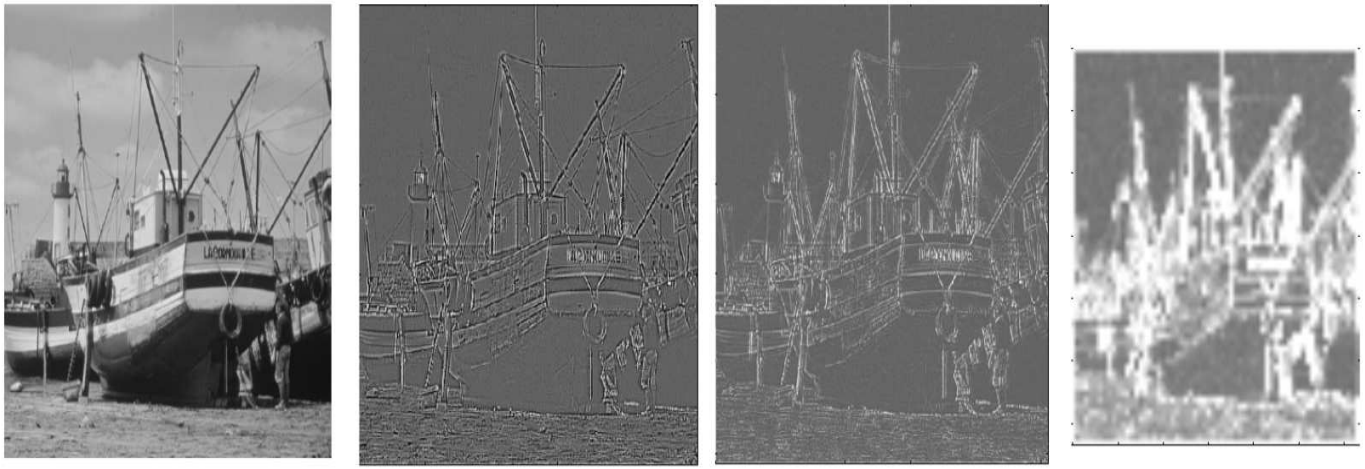
# Visual Image of Padding



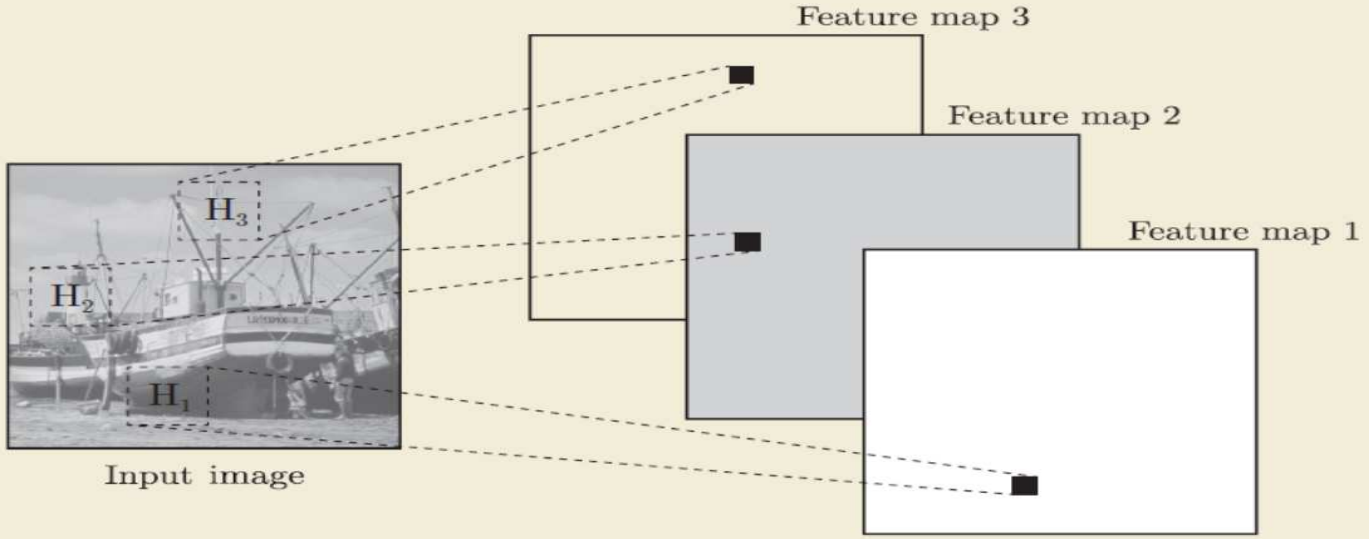
# Receptive Field

- Each pixel in the feature map array receives input from within a specific region of the previous (input) layer.
- This is known as the corresponding receptive field.
- Depth is the number of kernel matrices (filters) that are employed.
- For each filter, a corresponding feature map image array results.

# Case Study: A Boat

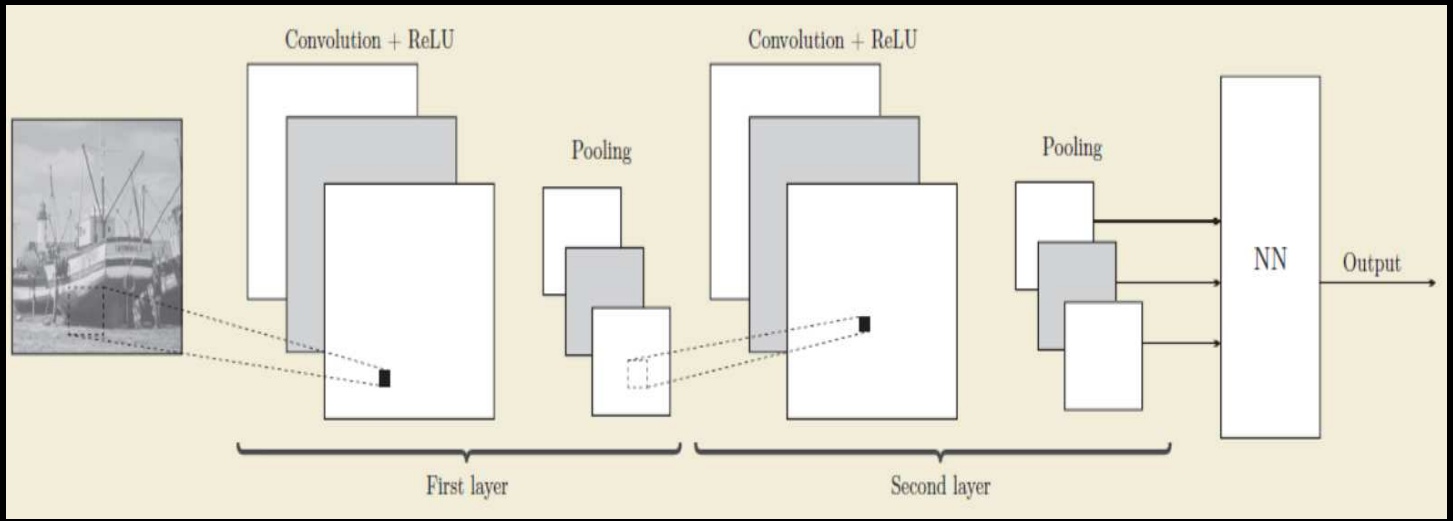


# Case Study: A Boat




The depth of the feature map array is **three**

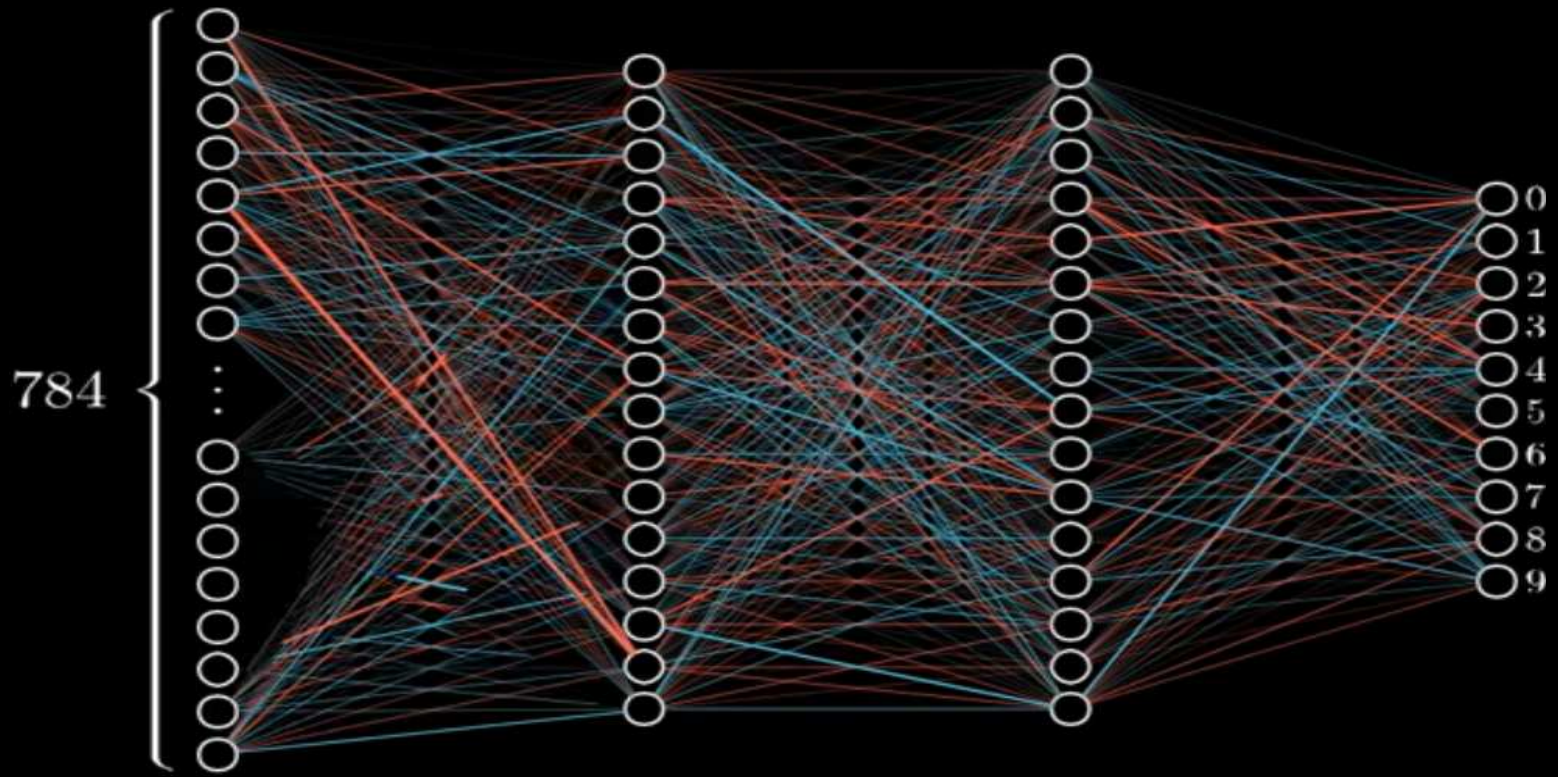
# Case Study: A Boat



# What is Deep Learning

- One view: AI $\supset$ Machine Learning $\supset$ Deep Learning
- Deep learning algorithms establish initial parameters from the data and then train the computer to learn independently by recognizing data patterns using multiple layers of processing
- These multiple layers can be in the single digits or the millions and each is a form of a neural network that are connected together and jointly estimated with "backpropogation"
- The goal is to establish an optimal set of weights for each connection between each layer in total
- Using a training dataset the key is minimizing the classification difference between $\mathbf{y}$ and $\hat{\mathbf{y}}$ by going forward through NN to get a difference and then going backwards to reweight/tune
- Achievements: near-human image classification, near-human speech transcription, near-human handwriting transcription, high quality text to speech, successful commercialization (Assistant and Alexa), autonomous driving, better search results, superhuman GO and chess competing

# Deep Learning Illustration (Again)




From Chollet, Francois. *Deep learning with Python.* Simon and Schuster, 2021.

- Now instead of 784 inputs and 10 outputs imagine multiple millions of each
- Now instead of 2 hidden layers imagine multiple millions

# Neural Networks and Deep Learning

- For a long time researchers struggled to find a way to train neural nets, without success then in 1986, David Rumelhart, Geoffrey Hinton and Ronald Williams published the paper: "Learning Internal Representations by Error Propagation" introducing the backpropagation training algorithm, which is still the standard today
- Their idea is based on *Gradient Descent* using an efficient technique for computing the gradients automatically
- In two passes through the network (one forward, one backward), the backpropagation algorithm computes the gradient of the network's error with regards to every single model parameter
- It finds out how each connection weight and each bias term should be tweaked in order to minimize the error
- With these gradients, it performs a regular Gradient Descent step, and the whole process is repeated until the network converges to the solution

# Deep Learning Algorithmic Forward Pass Details

- Process mini-batch one at a time going through the full training set multiples where each pass is called an epoch.
- These mini-batches are then passed to the input layer of the network, which then sends it to the first hidden layer.
- The initial weights of the hidden layers are randomly assigned.
- Then compute the output of all the neurons in this layer for every mini-batch instance.
- This result is then passed to the first hidden layer.
- Then repeat this process at this next layer, and so on until the last layer is reached: the output layer.
- All of these intermediate results are recorded for the backward pass.
- This completes the forward pass.
- Now calculate the network error: $\mathbf{y} - \hat{\mathbf{y}}$.

# The Chain Rule

- This provides a means of differentiating nested functions of the form $f \circ g = f(g(x))$.
- The case of $g(x)^{-1} = (4x^3 - 2x)^{-1}$fits this categorization because the inner function is $g(x) = 4x^3 - 2x$ and the outer function is $f(u) = u^{-1}$.
- Usually $u$ is used as a placeholder here to make the point that there is a distinct subfunction.
- To correctly differentiate such a nested function, we have to account for the actual *order* of the nesting relationship, done by:

$$\frac{\partial}{\partial \mathbf{X}} f(g(x)) = f'(g(x))g'(x),$$

  provided that $f(x)$ and $g(x)$ are both differentiable functions.
- We can also express this in the other standard notation: if $y = f(u)$ and $u = g(x)$ are both differentiable functions, then

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx},$$

  which may better show the point of the operation.
- If we think about this in purely fractional terms, it is clear that $du$ cancels out of the right-hand side, making the equality obvious.

# The Chain Rule

- Let us use this new tool to calculate the function $g(x)^{-1}$ from above ($g(x) = 4x^3 - 2x$):

$$\frac{\partial}{\partial \mathbf{X}} g(x)^{-1} = (-1)(4x^3 - 2x)^{-2} \times \frac{\partial}{\partial \mathbf{X}}(4x^3 - 2x)$$

$$= \frac{-(12x^2 - 2)}{(4x^3 - 2x)^2}$$

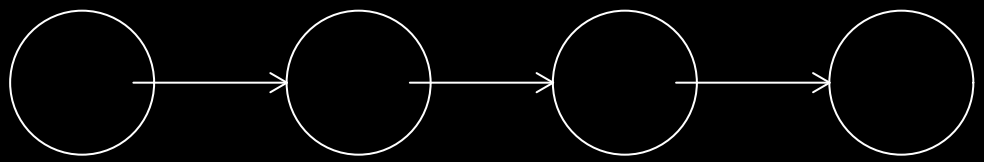
$$= \frac{-6x^2 + 1}{8x^6 - 8x^4 + 2x^2}.$$

# Deep Learning Algorithmic Backward Pass Details

- Starting with $\mathbf{y} - \hat{\mathbf{y}}$ for the smaller number of nodes in the output layer measure these error contributions backward for each connection in the layer below using the chain running backward until the input layer is reached.
- The reverse pass efficiently measures the error gradients across every connection weight in the layer below by propogating the error gradient backward through the networ..
- Thus the last step performs a Gradient Descent step to adjust all of the connection weights in the network using all of the individual error gradients just calculated.
- Now the Deep Learning network is fully trained and can be applied to different test data.

## Backpropogation Calculus ("neurons that fire together wire together")

- Start with the simplest type of network with one neuron per layer, so there are 3 weights and 3 biases, labeled: $\omega_1, b_1, \omega_2, b_2, \omega_3, b_3$.



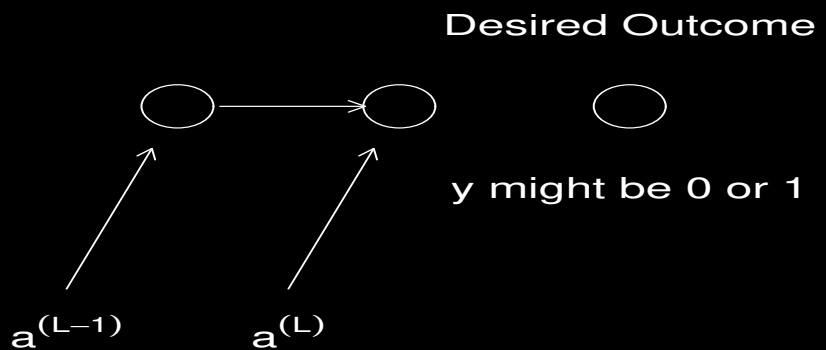
- Goal: to understand how sensitive the cost function is to these variables through the function:

$$J(\omega_1, b_1, \omega_2, b_2, \omega_3, b_3).$$

- This way we know which adjustments to these terms is going to cause the most "efficient" decrease to the const function.

# Backpropogation Calculus

- For simplicity let's focus on the connection of the last two neurons.




- The cost of this (simple) network is:

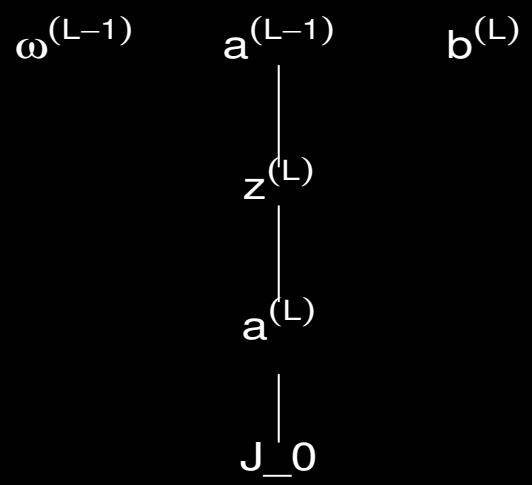
$$J_0(\omega_1, b_1, \omega_2, b_2, \omega_3, b_3) = (a^{[L]} - y)^2$$

where the "0" subscript indicates that this the cost of just a training sample.

# Backpropogation Calculus

- Let's see how $a^{[L]}$ is calculated...

$$a^{(L)} = \phi(\omega^{(L)} a^{(L-1)} b^{(L)} = \phi(z^{(L)}).$$




- Note: $\omega^{(L)}, z^{(L)}, a^{(L)}, J_0$ are all numbers.

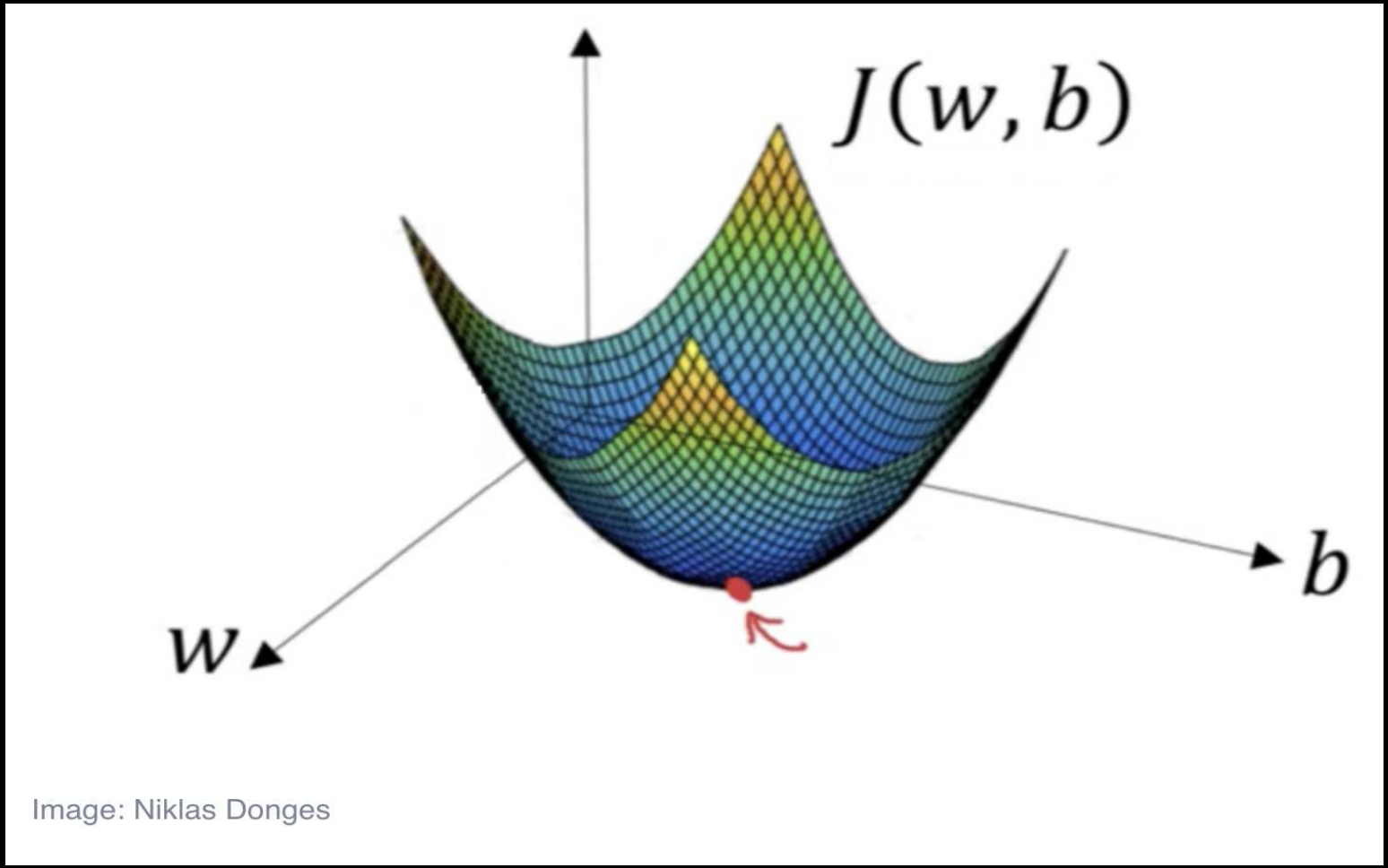
# Gradient Descent for Machine Learning

- A gradient is a derivative that measures how much an (output) function changes for small changes in an input.
- In the deep learning sense it measure the change in all the weights for a small change in an error.
- Since it is a derivative, higher values mean that the slope is steeper and lower values mean that the slope is flatter.
- Higher slopes mean that the model can learn faster and zero slopes mean that no learning takes place.
- We want to get to the minimum (of errors) in a multidimensional sense, which is to minimize the function $J(w, b)$, where $w$ is a set of weights, then a Gradient Step looks like:

$$b = a - \gamma \nabla f(a),$$

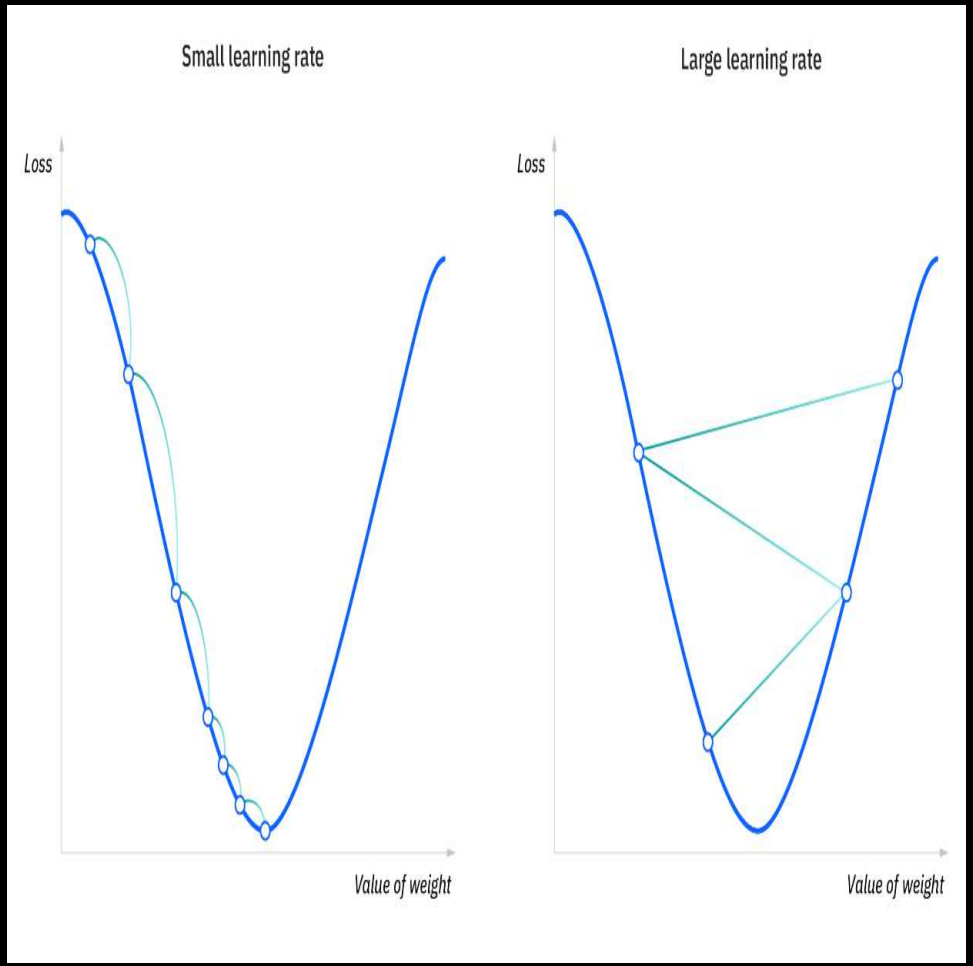
where $b$ is the next location, $a$ is the current location, $\gamma$ is the learning rate or step size, and $\nabla f(a)$ is the direction of the step.

# Gradient Descent Illustration



# Learning Rate

- This is the size of the steps that are taken to reach the function minimum.
- It is usually a small value that is evaluated and updated based on the behavior of the cost func- tion.
- A large value results in larger steps but risk overshooting the minimum.
- A small value is more precise, but lowers the efficiency since it takes more cycles to reach the minimum.




Source: IBM.