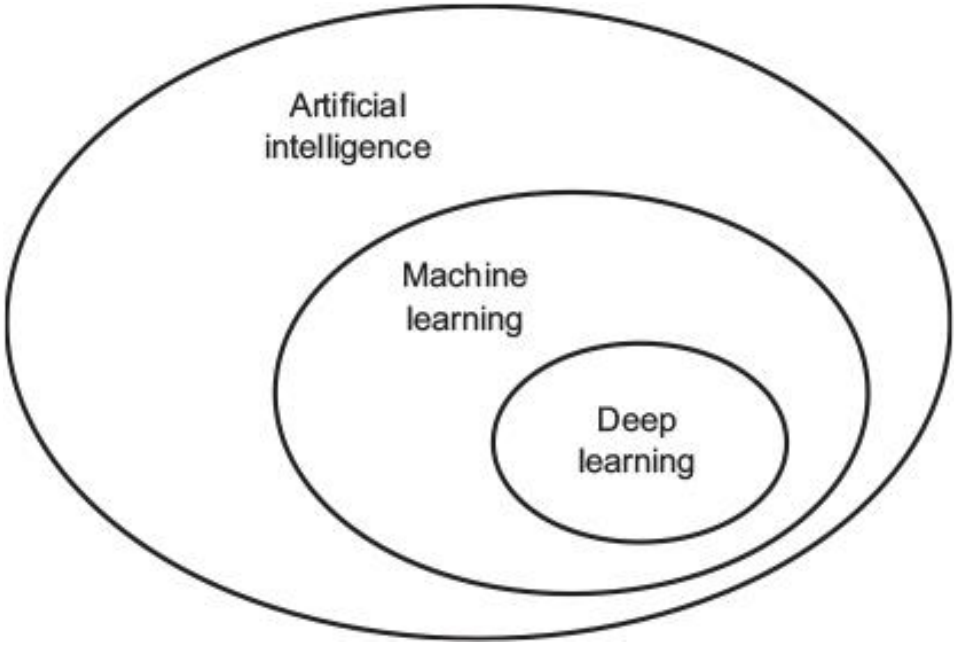
Introduction to Machine Learning: R/keras

Tensors, MNIST classification

Methods Seminar, SS20

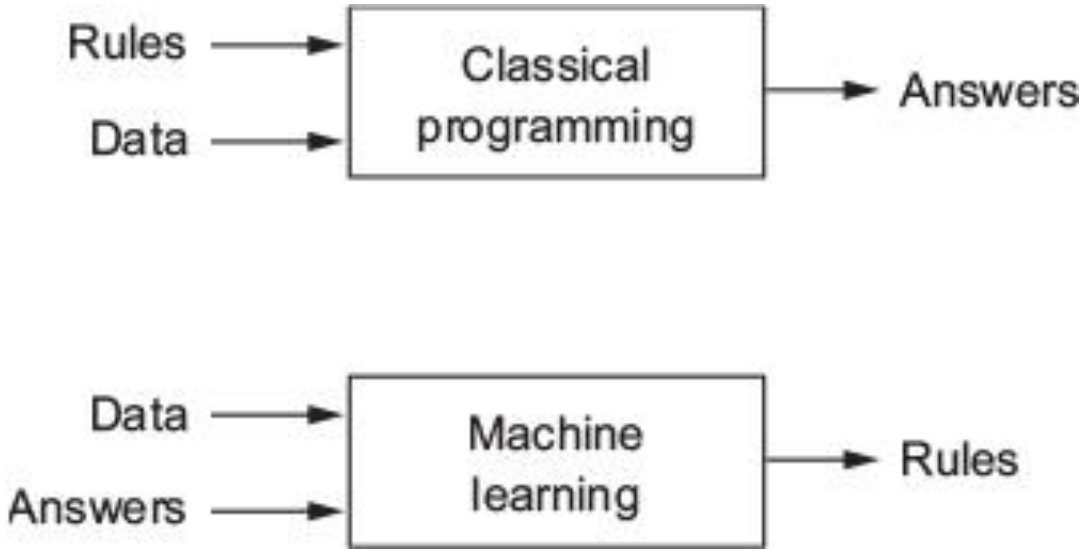
Lead: Prof. Mark Robinson
Responsible: Ashley Polhemus and Sona Hunanyan

Artificial intelligence, machine learning, and deep learning



- **AI** (born in 1950s): the effort to automate intellectual tasks normally performed by humans
- **ML** (born in 1990s): could a computer go beyond "whatever we know how to order it to perform" and learn on its own how to perform a specified task?
- **DL** (became prominent in early 2010s): define it after we know what machine-learning algorithms do…

More on machine learning



- ML system is *trained* rather than explicitly programmed
- ML finds statistical structure in data that eventually allows the system to come up with rules for automating the task
- ML is tightly related to mathematical statistics, but
  - tends to deal with large, complex datasets
  - exhibits comparatively little mathematical theory
  - ideas are proven empirically more often than theoretically

What do we need to do machine learning?

Input data points

- Speech recognition task – files of people speaking
- Image tagging task – pictures

- ***Examples of the expected output***
  - Speech recognition task – human-generated transcripts of sound files
  - Image tagging task - tags such as “dog,” “cat,” and so on.
- ***A way to measure whether the algorithm is doing a good job*** - to determine the distance between the algorithm’s current output and its expected output. This measurement is used to adjust the way the algorithm works. This adjustment step is called learning.

Learning representations from data

Meaningfully transform data

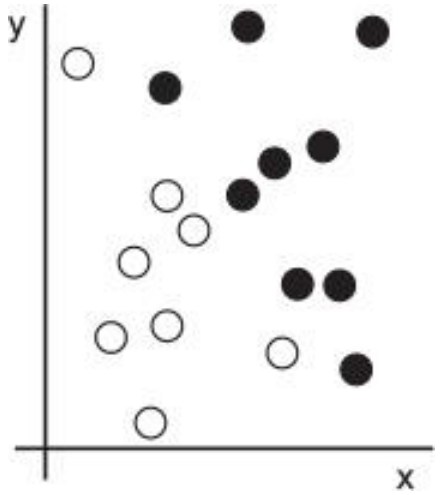
meaningful outputs  to learn useful *representations* of the data

What's a *representation*?

A color image  RGB format  select all red pixels in the image

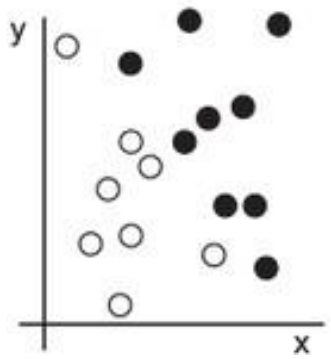
 HSV format  make the image less saturated

Learning representations from data: an example

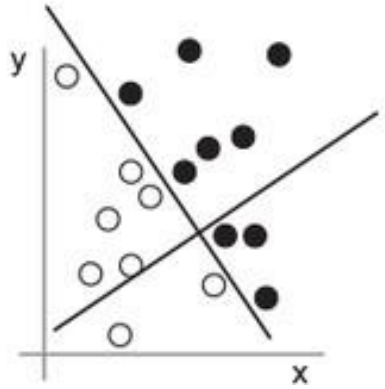


Task: Take (x, y) coordinates of the points and output whether that point is likely to be black or white.

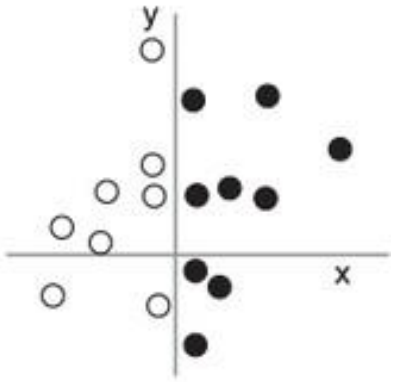
1: Raw data



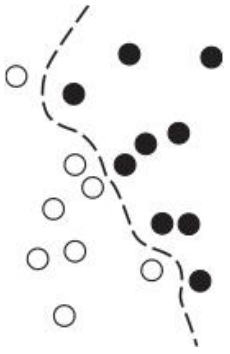
2: Coordinate change



3: Better representation

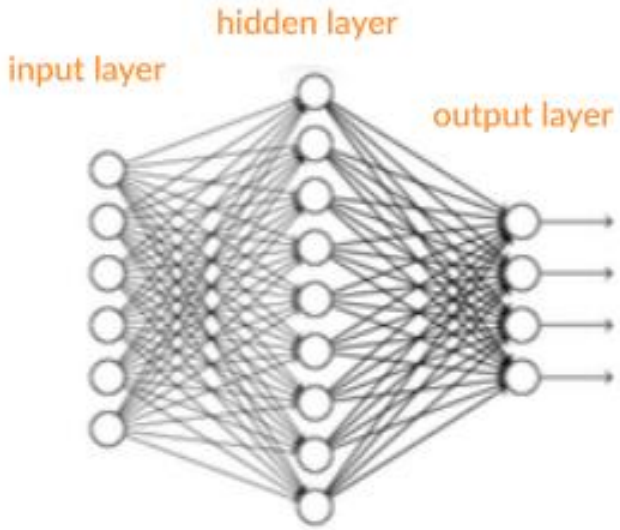


Alternative: Decision Boundary

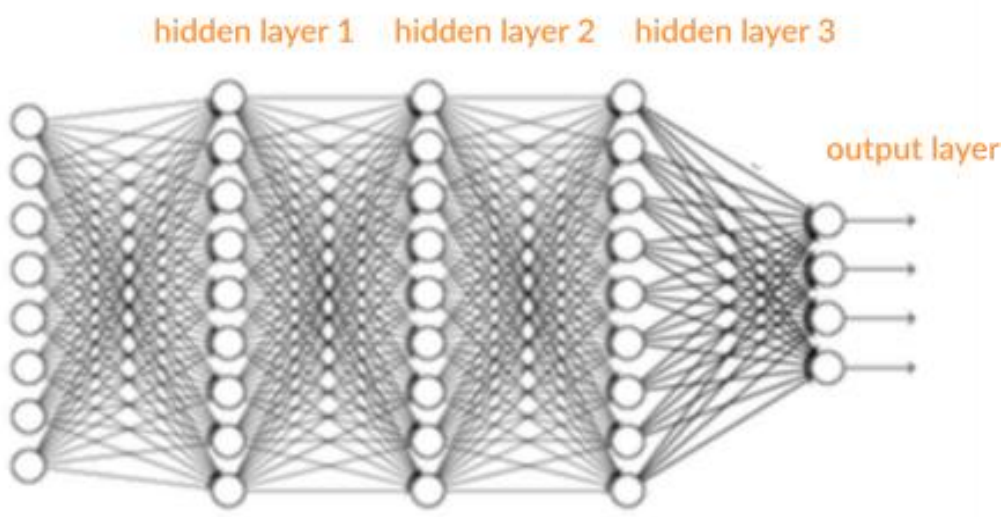


The ”deep” in deep learning

- Stands for the idea of successive layers of representations
- How many layers contribute to a model of the data is called the *depth* of the model.



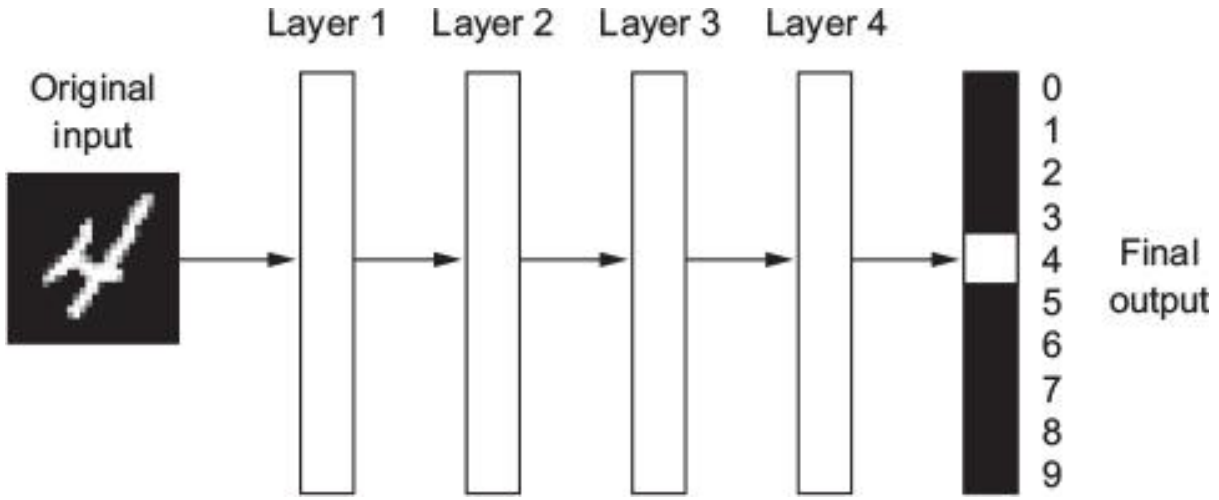
“Shallow” Model



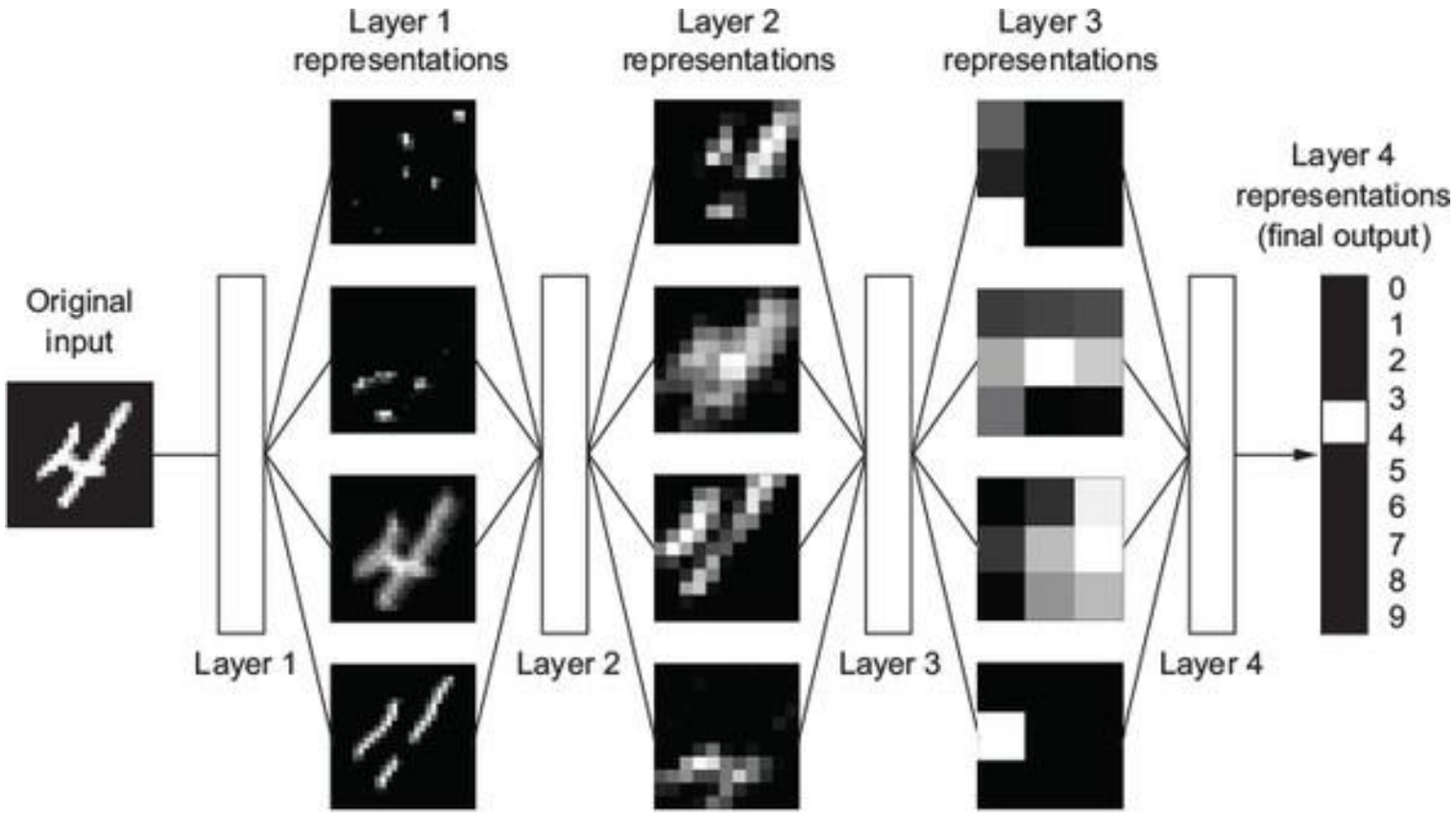
“Deep” Model

Basics of a neural network

- The layered representations are learned via models called neural networks
- The term *neural network* is a reference to neurobiology, however, DL models are NOT models of the brain, only that some concepts in DL were developed by drawing inspiration from our understanding of the brain.

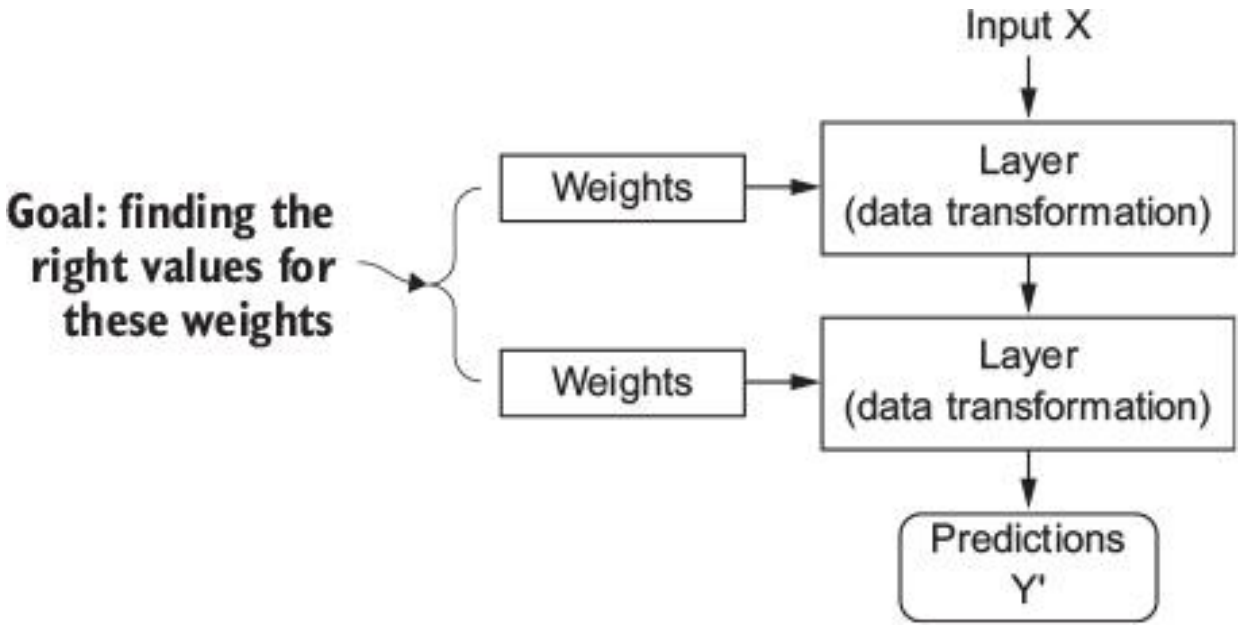


Deep representations learned by a digit-classification model

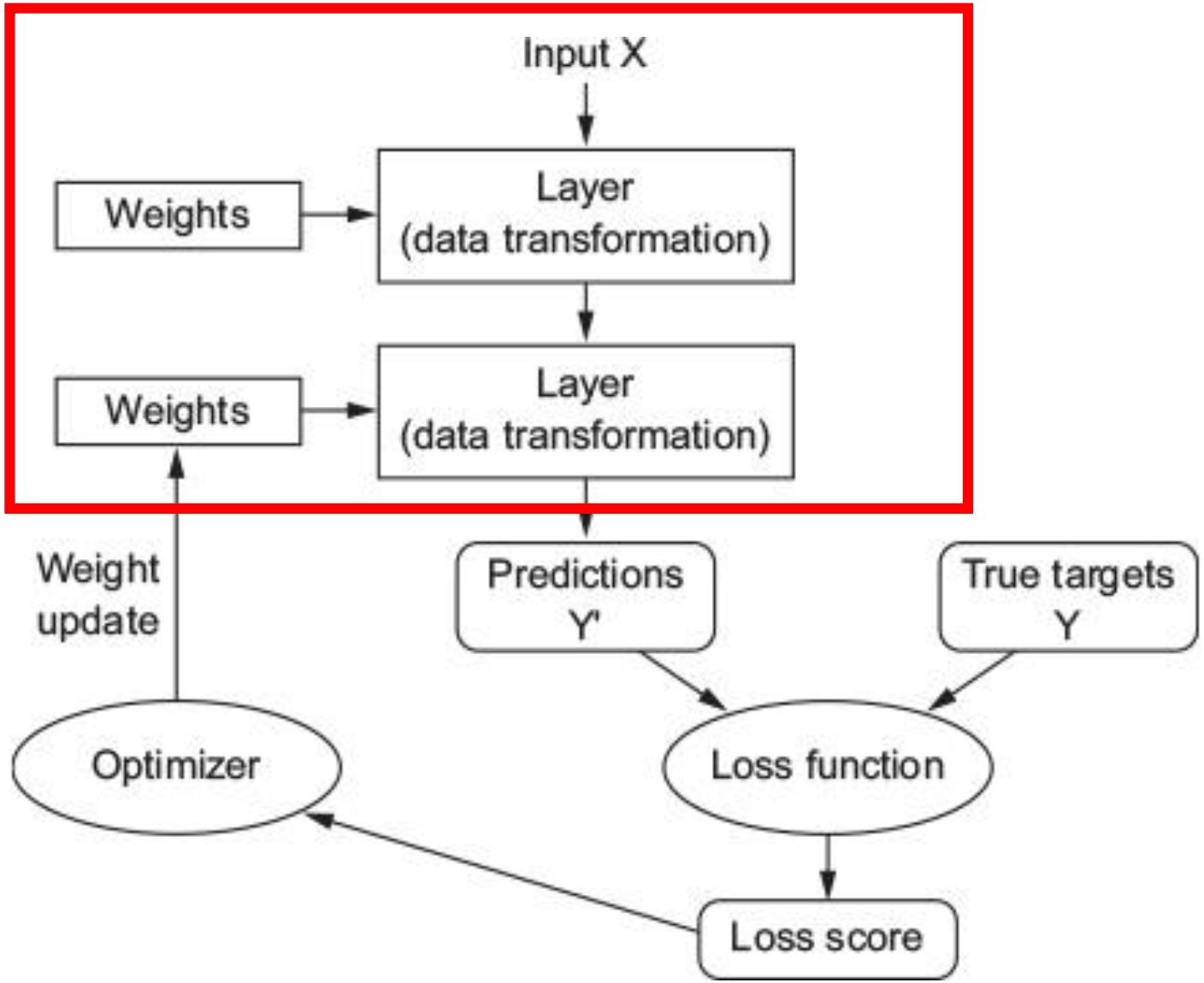


How deep learning works? Step 1

- The specification of what a layer does to its input data is stored in the layer's weights (parameters), equivalently, the transformation implemented by a layer is *parameterized by its weights*.
- Learning means finding a set of values for the weights of all layers in a network, such that the network will correctly map example inputs to their associated targets.



Building blocks: Tensors, weights, and activation functions

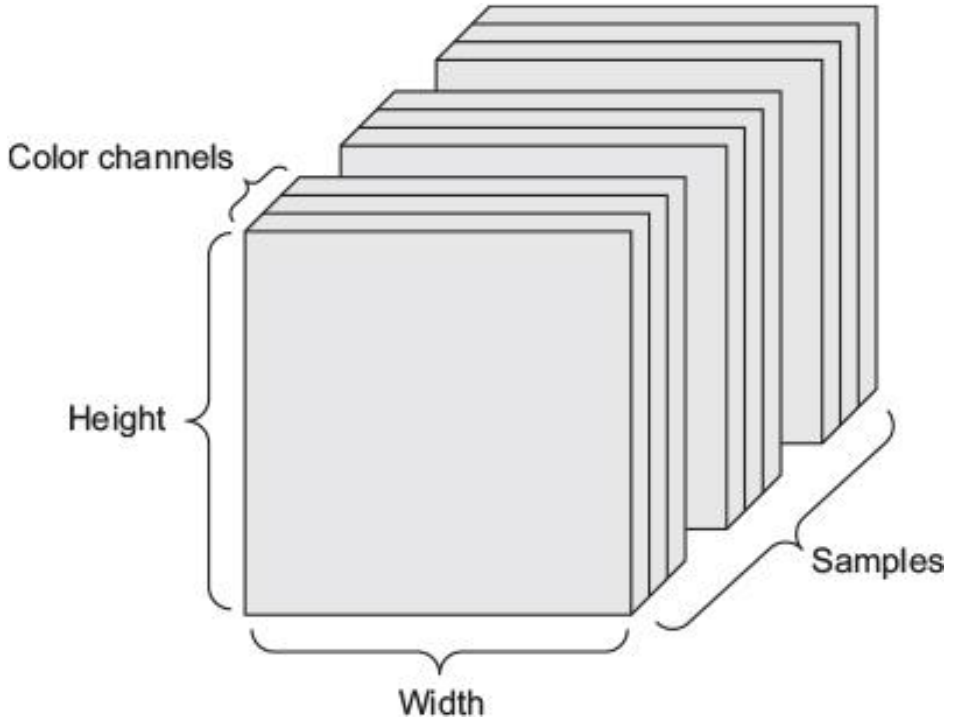


What is a tensor?

Multidimensional array of integers upon which you can perform simple calculations to transform input data in a layer of a neural network

Defined by

- Number of axes (Rank)
- Shape
- Data type



Common tensor shapes:

- 2D – Vector data (samples, features)
- 3D – Timeseries data (samples, time, features)
- 4D – Images (samples, height, width, channels)
- 5D – Videos (samples, frames, height, width, channels)

What Rank and Shape would the following tensors have?

An actuarial dataset of 1 million people, including each person's age, area code, and income

2D, (1000000, 3)

A batch of 128 grayscale images of size 256 × 256

4D, (128, 256, 256, 1)

A dataset of tweets, where we encode each tweet as a sequence of 140 characters out of an alphabet of 128 unique characters

3D, (# samples, 140, 128)

Tensor Operations

Simple operations applied to tensors to manipulate data in a layer

- Element- wise operations (addition, subtraction, multiplication)
- Relu() *Same as max(x, 0)*
- Dot product

Dot Product

Vector - Vector Dot Product

- For all indices in vector x, sum the products of x[[i]] and y[[i]]

- For each row in matrix X and each column in matrix Y, compute the vector dot product of each row of X and column of Y

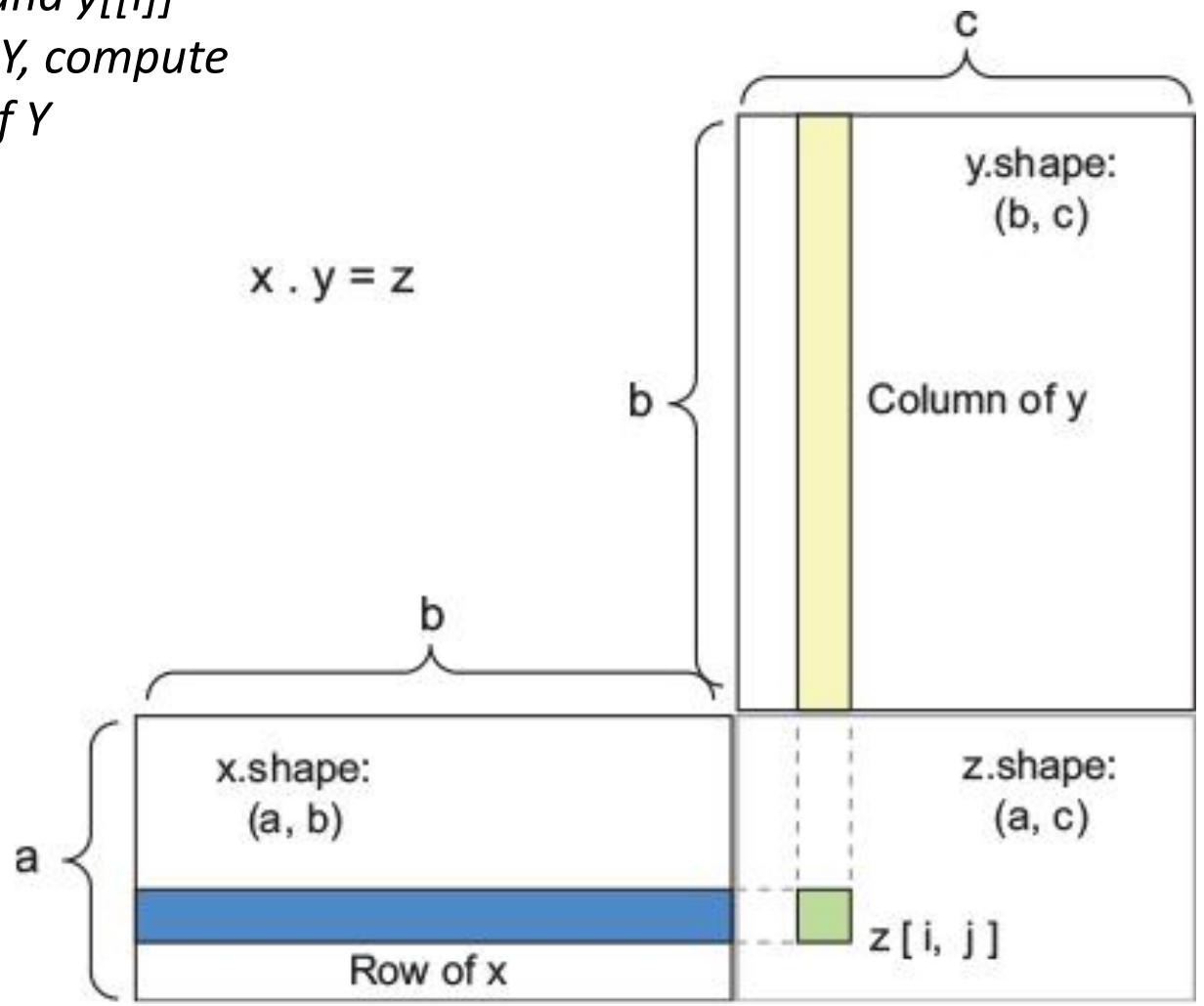
Properties

- asymmetric
- relies on shape compatibility

Outputs

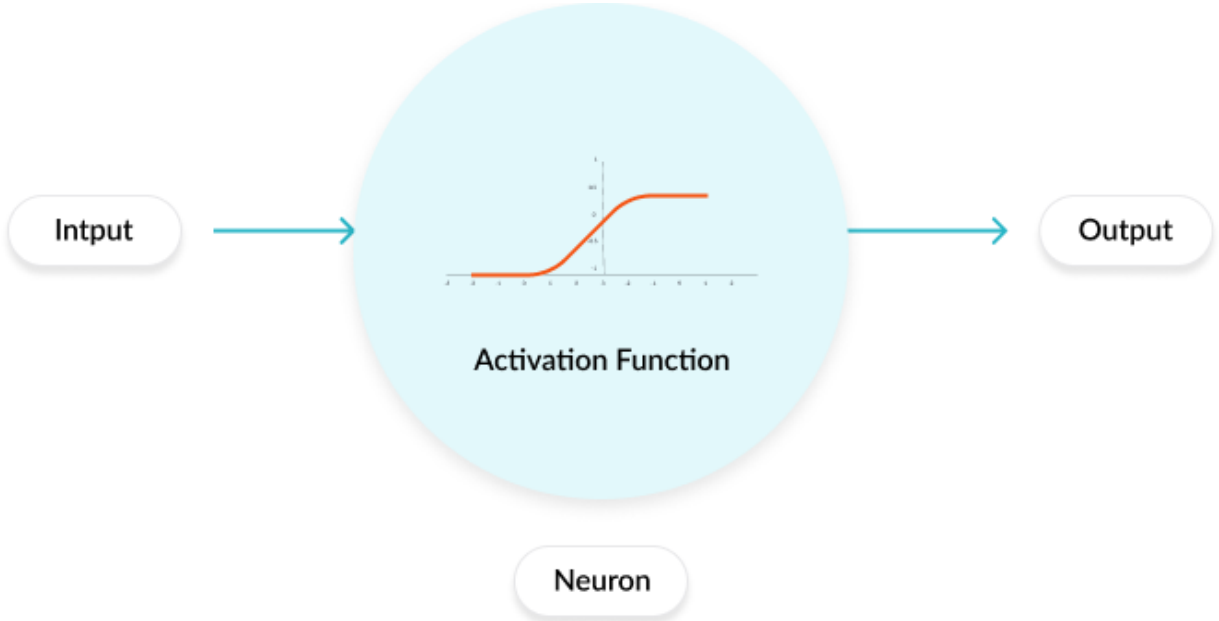
(a, b, c, d) . (d) -> (a, b, c)

(a, b, c, d) . (d, e) -> (a, b, c, e)



Activation Functions

- In a neural network each neuron has a weight and multiplying the input number with the weight gives the output of the neuron, which is transferred to the next layer.
- The activation function is a mathematical “gate” in between the input feeding the current neuron and its output going to the next layer.



Mathematical representation of an activation function

Perceptron are the simplest NN consisting of n number of inputs and one neuron.

The process of passing the data through the neural network is known as forward propagation.

- Step1: For $x = (x_1, x_2, \ldots, x_n)$ and w = $(w_1, w_2, \ldots, w_n)$, $x \cdot y$
- Step2: Add bias b, $z = x \cdot w + b$
- Step3: Pass the value of z to a non-linear activation function (Sigmoid)

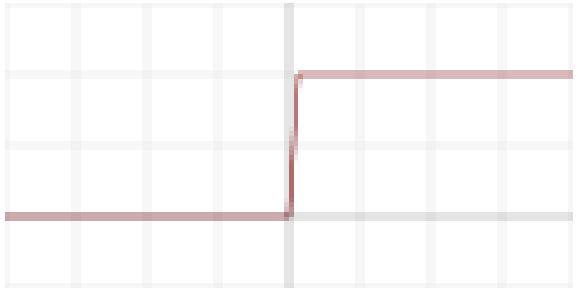
$$\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}$$

Three types of activation functions

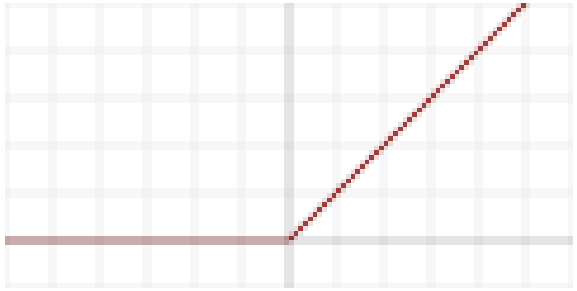
- Binary step function
- Linear activation functions
- Non-linear activation functions
  - Sigmoid/Logistic
  - TanH / Hyperbolic Tangent
  - ReLU (Rectified Linear Unit)
  - Leaky ReLU
  - Parametric ReLU
  - Softmax
  - Swish

Desirable Properties of an activation function

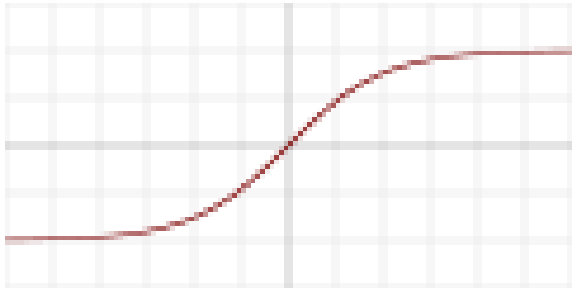
- Non-linear
- Range – Balance between stability and efficiency
- Continuously differentiable – Enables gradient-based optimization and backpropogation
- Approximates f(x) = x near the origin – Enables efficient learning when weights are initialized with small random values



Binary Step Function

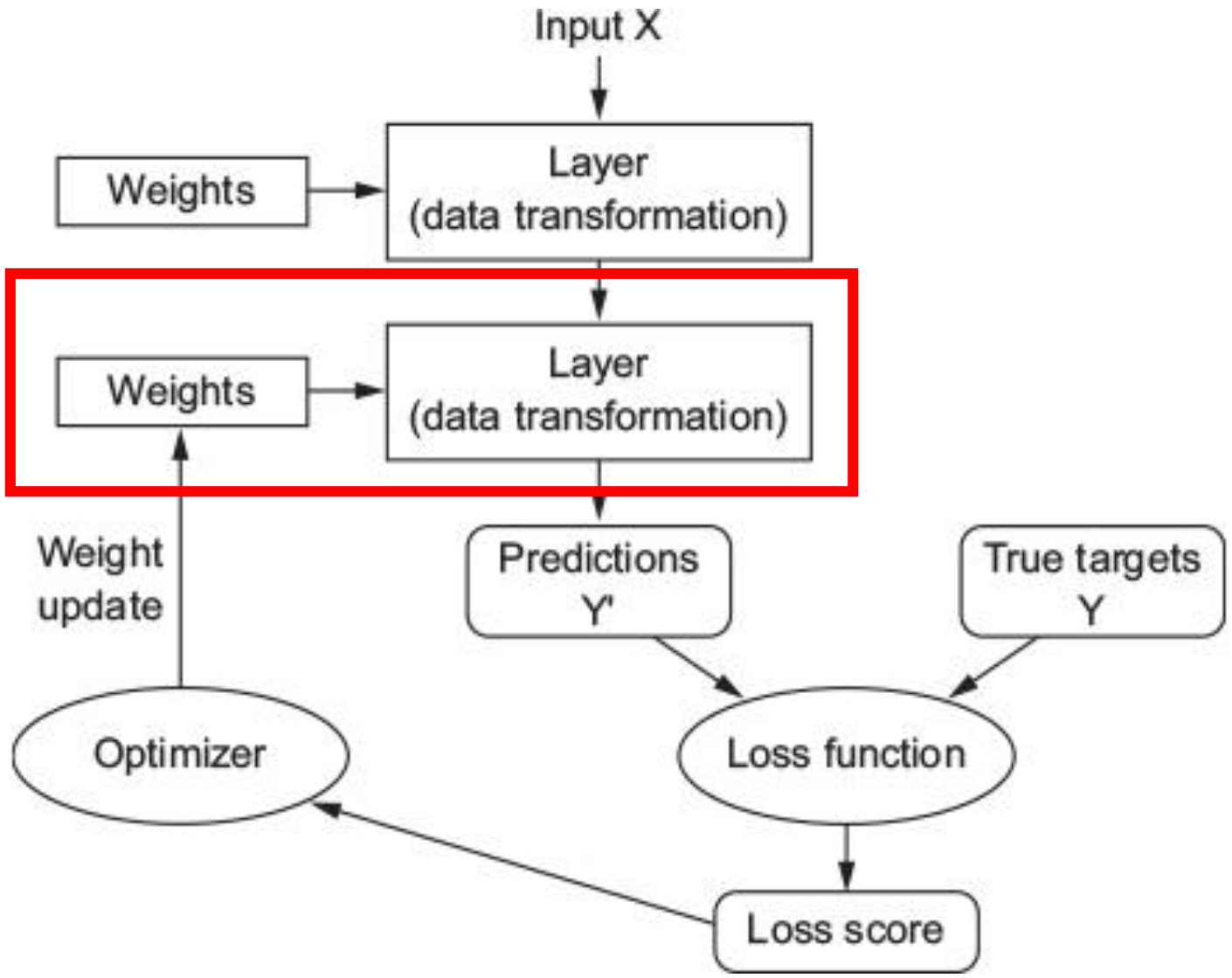


ReLU

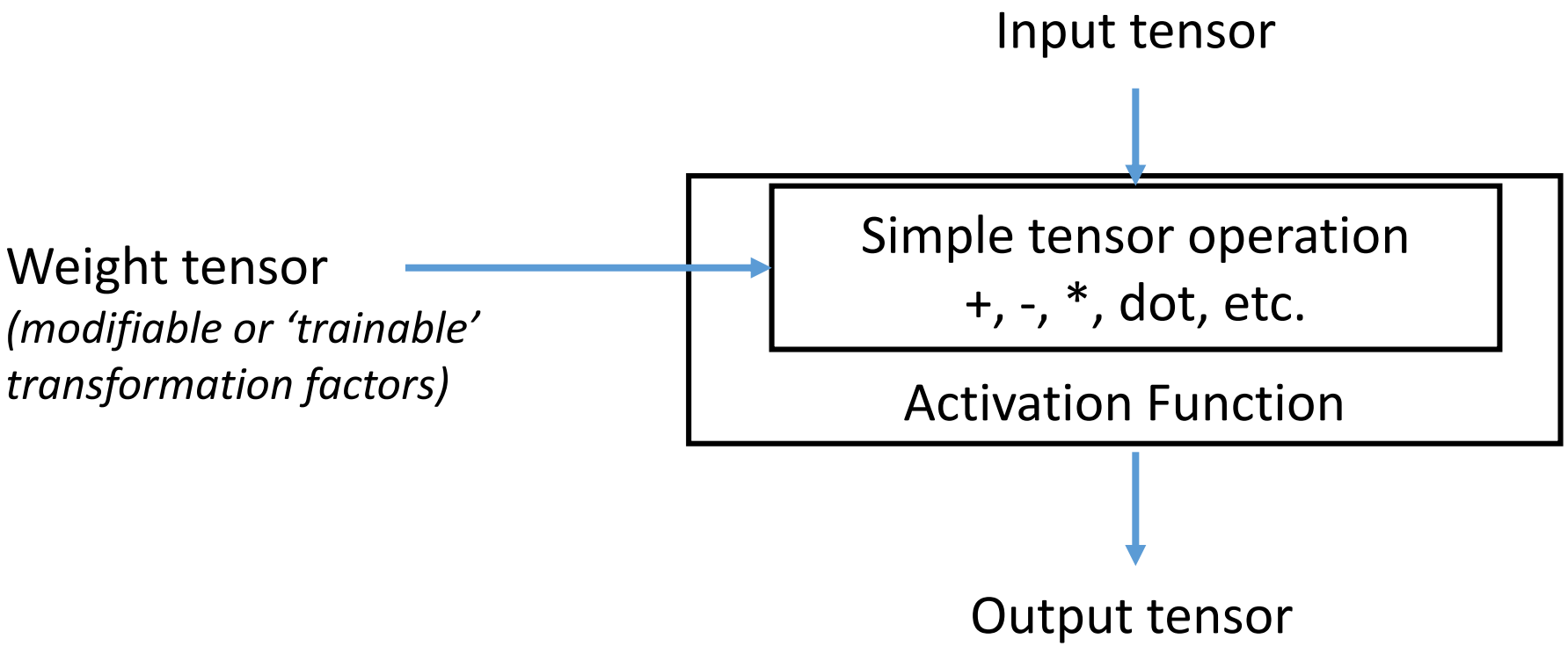


TanH

Unpacking Deep Learning ‘Layers’



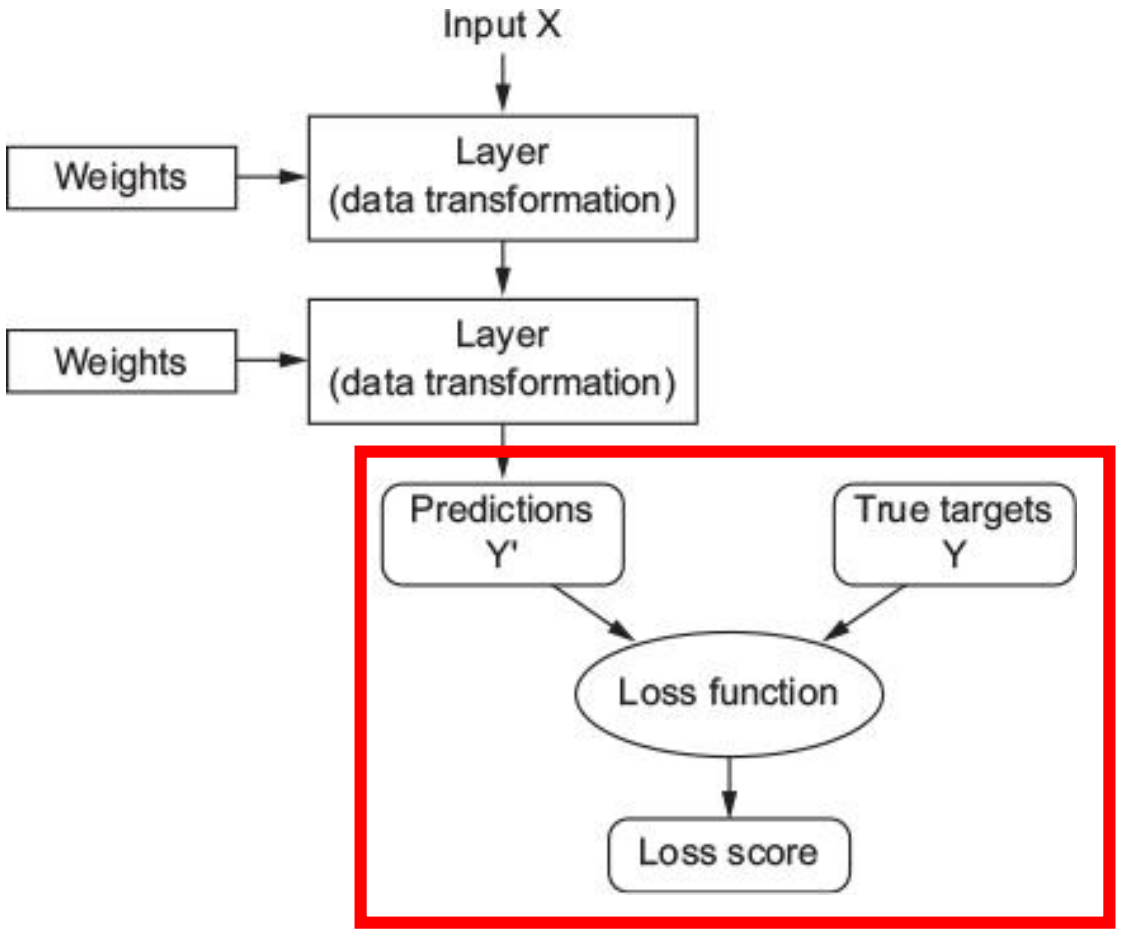
Unpacking Deep Learning ‘Layers’



How deep learning works? Step 2

- To control the output of a neural network, you need to be able to measure how far this output is from what you expected.

- This is done by a loss function (objective function) that takes the predictions of the network and the true target and computes a *distance score*, capturing how well the network has done.



What is a loss function?

Prediction: The model's output (i.e., a classification)

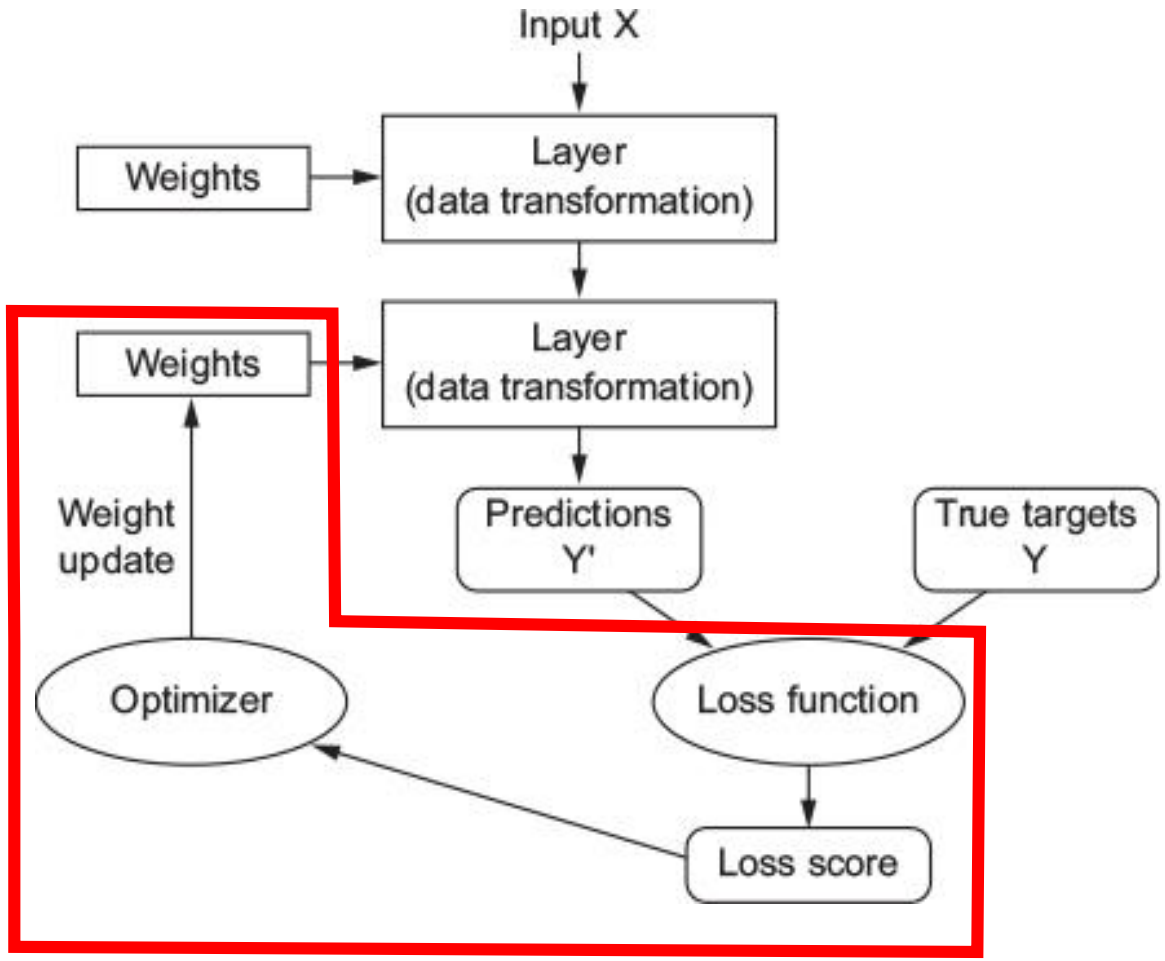
Loss Function: Function calculating chosen *metrics* of the model's performance (i.e., accuracy).

Loss Score: Quantifies the difference between the desired/true values and those generated by the model

Lower loss score –> better prediction

How deep learning works? Step 3

- The fundamental trick in deep learning is to use this score to adjust the value of the weights in a direction that will lower the loss score.
- This adjustment is the job of the optimizer, which implements the *Backpropagation* algorithm

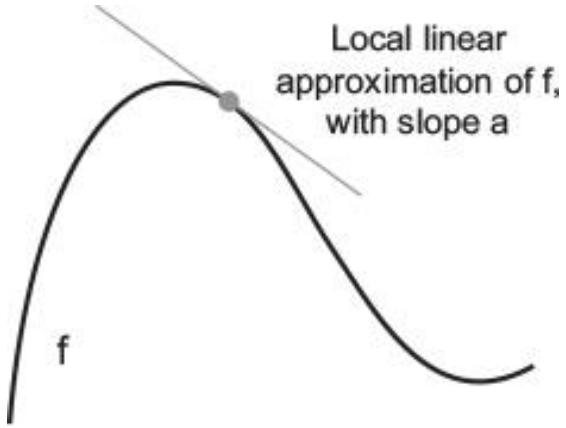


What is gradient descent?

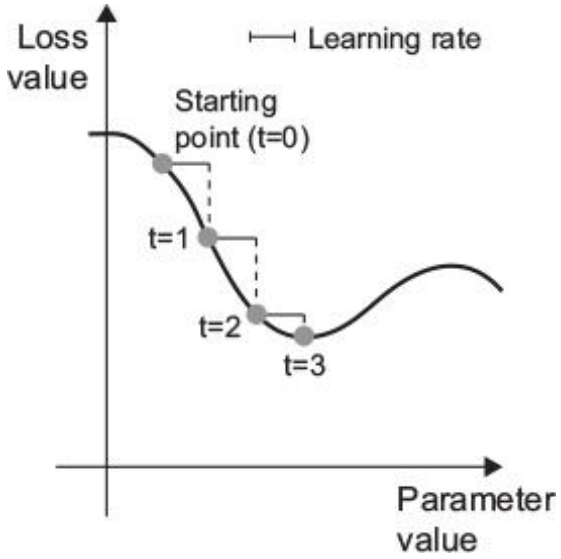
Goal: Minimize loss function

To do this, the optimizer adjusts the weights a little in the direction of the gradient of the loss function

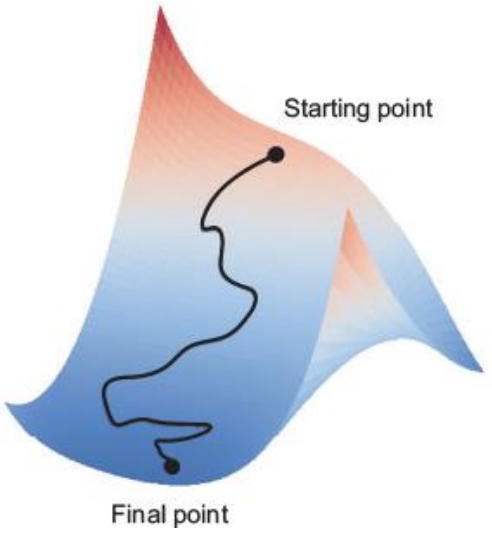
1. Find the derivative ('gradient') of the loss function



2. Adjust weights in the direction of the gradient (backpropagation, via chain rule)

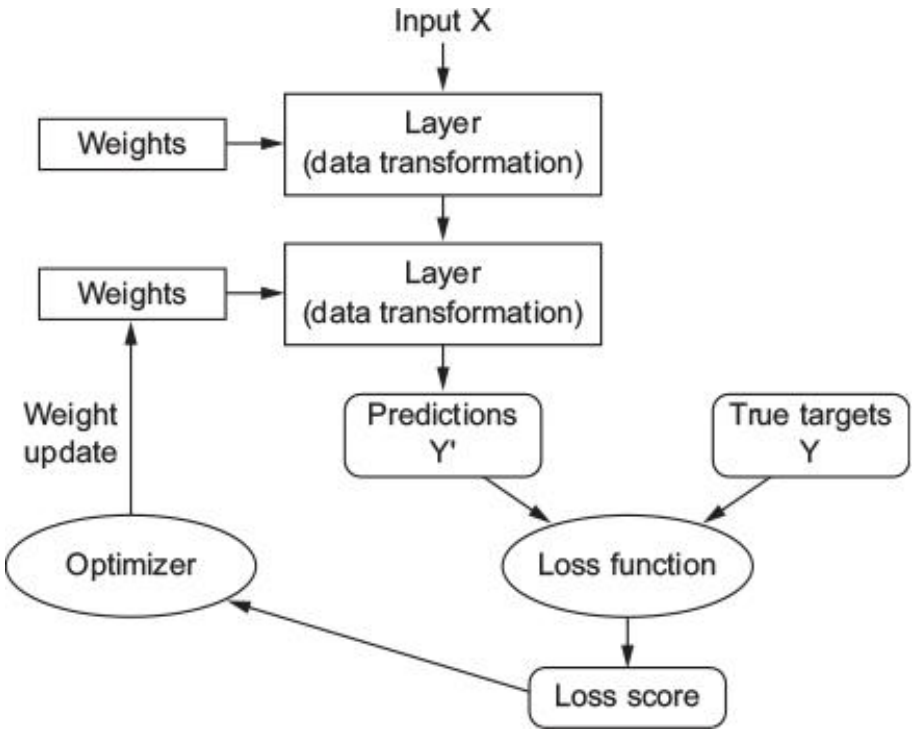


3. Appropriate adjustments and batch sizes allow you to find global min of loss()



Summing it all up…

- A *deep learning model* contains successive layers of data representations which are trained to transform input data based on an original set of inputs and outputs
- A *tensor* is a multi-dimensional array which can be used to transform data, or itself be transformed via *simple tensor operations*
- A deep learning model's *input* and *output* are tensors containing data, while *weights* are tensors to used to transform input and output tensors. Tensor operations are performed sequentially in (many) *layers*.
- An *activation function* determines whether and which neurons are fired given the inputs and outputs of the layers
- An *optimizer* adjusts the model's weights according to the gradient of a *loss function* to minimize the *loss score*
- Weights are adjusted through *backpropagation*, applying the chain rule to adapt weights according to the loss function's derivative.



Translating between Statistics and Machine Learning

	Statistics	Machine Learning
Approach	Data Generating Process	Algorithmic Model
Driver	Math, Theory	Fitting Data
Focus	Hypothesis Testing, Interpretability	Predictive Accuracy
Data Size	Any Reasonable Set	Big Data
Dimensions	Used Mostly for Low Dimensions	High Dimensional Data
Inference	Parameter Estimation, Predictions, Estimating Error Bars	Prediction
Model Choice	Parameter Significance, In-sample Goodness of Fit	Cross-validation of Predictive Accuracy on Partitions of Data
Popular Tools	R	Python
Interpretability	High	Low

https://pbs.twimg.com/media/ENNEGGhX0AEWNYO.jpg

Translating between Statistics and Machine Learning

Statistics	Machine learning
Covariates	Features
Outcome variable	Target
Model	Network, graphs
Parameters	Weights
Model for discrete var.	Classifier
Model for continuous var.	Regression
Log-likelihood	Loss
Multinomial regression	Softmax
Measurement error	Noise
Subject/observation	Sample/instance
Dummy coding	One-hot encoding
Measurement invariance	Concept drift

Statistics	Machine learning
Prediction	Supervised learning
Latent variable modeling	Unsupervised learning
Fitting	Learning
Prediction error	Error
Sensitivity	Recall
Positive predictive value	Precision
Contingency table	Confusion matrix
Measurement error model	Noise-aware ML
Structural equation model	Gaussian Bayesian network
Gold standard	Ground truth
Derivation–validation	Training–test
Experiment	A/B test

https://pbs.twimg.com/media/EQBFqE1U8AEVh0i.png

Helpful Resources

- Deep Learning with R
- https://livebook.manning.com/book/deep-learning-with-r/chapter-1/101

- An introduction to the math behind neural networks:
- https://hackernoon.com/a-6ur13zzx

- How to choose Activation Functions:
- https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/

- A tutorial for comparing Optimizers:
- https://medium.com/@onlytojay/mnist-cnn-optimizer-comparison-with-tensorflow-keras-163735862ecd

In-Class Exercise

- Keras Setup
- MNIST example

What is Keras?

Keras (https://keras.rstudio.com) is a deep-learning framework that provides a convenient way to define and train almost any kind of deep-learning model.

- Keras has the following key features:
- It allows the same code to run seamlessly on CPU or GPU.
- It has a user-friendly API that makes it easy to quickly prototype deep-learning models.
- It has built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.
- It supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, and so on. This means Keras is appropriate for building essentially any deep-learning model, from a generative adversarial network to a neural Turing machine.

Installation instructions

The Keras R package is compatible with R versions 3.2 and higher, to check which version do you have installed, type

+ version #in Rstudio

The documentation for the R interface is available at https://keras.rstudio.com

The main Keras project website can be found at https://keras.io.

Install Anaconda for Python

For Windows users:
Install Anaconda for Python 3.x (https://www.anaconda.com/download/#windows) **before** installing Keras.

Linux and macOS users can install Anaconda from the same web page by choosing the corresponding tab.

Install Keras

install the keras R package from GitHub as follow

+ devtools::install_github("rstudio/keras")

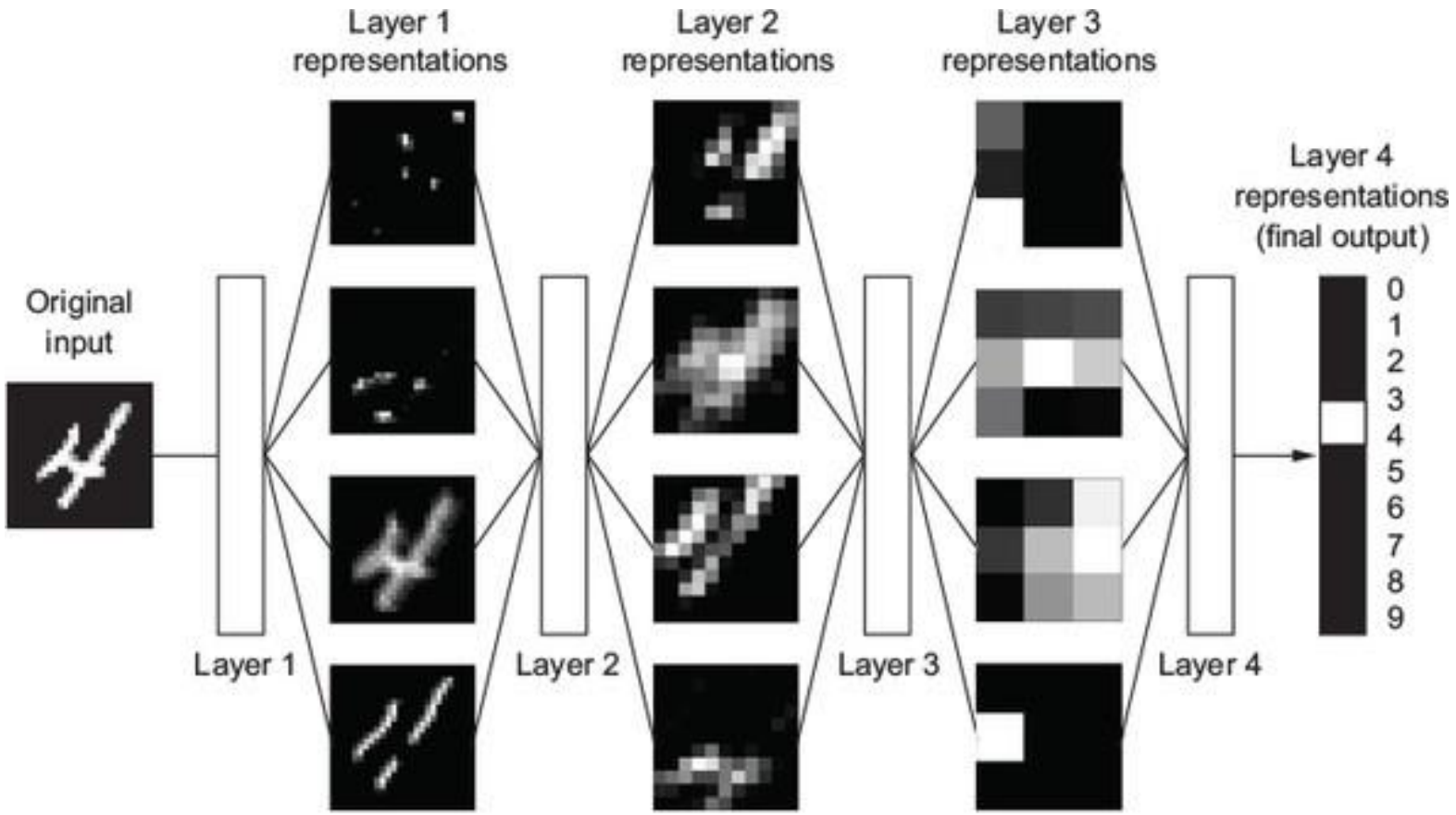
#The Keras R interface uses the TensorFlow backend engine by default.

To install both the core Keras library as well as the TensorFlow backend use the

+ library(keras)

+ install_keras() #linux and macOS users can choose method = "virtualenv“ or "conda"

MNIST Example



Exercise – Optimizers & Activation Functions

Optimizers

'Adadelta',
'Adagrad',
'Adam',
'Adamax',
'Nadam',
'RMSprop',
'SGD’

https://medium.com/@onlytojay/mnist-cnn-optimizer-comparison-with-tensorflow-keras-163735862ecd

Activation Functions

‘sigmoid’

‘softmax’

‘tanh’

‘relu’

https://towardsdatascience.com/exploring-activation-functions-for-neural-networks-73498da59b02