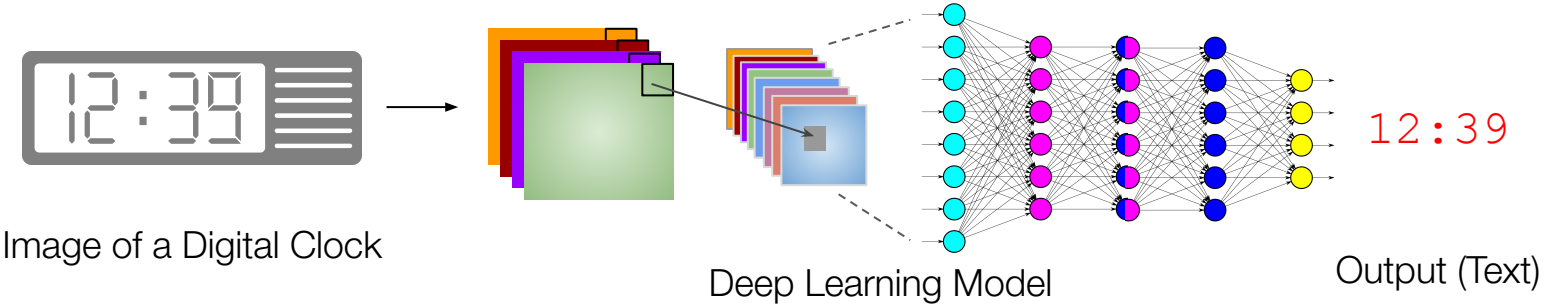
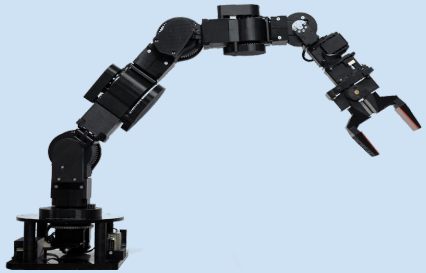
Module II: Fundamentals of Deep Learning

Projects


12:39
Image of a Digital Clock
Deep Learning Model
12:39
Output (Text)
Picture of Your Face
Deep Learning Model
Happy
Sad
Crying
Laughing
Output

1.1 AI, ML & DL

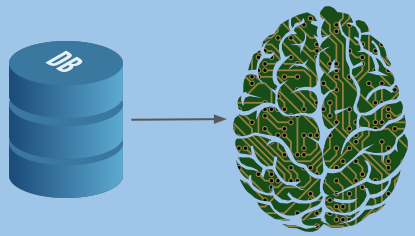
Artificial Intelligence



a very broad field

1950s

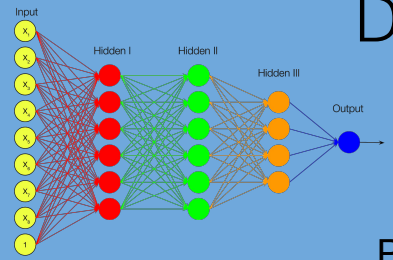
Machine Learning



Fundamentals of
“learning from data”

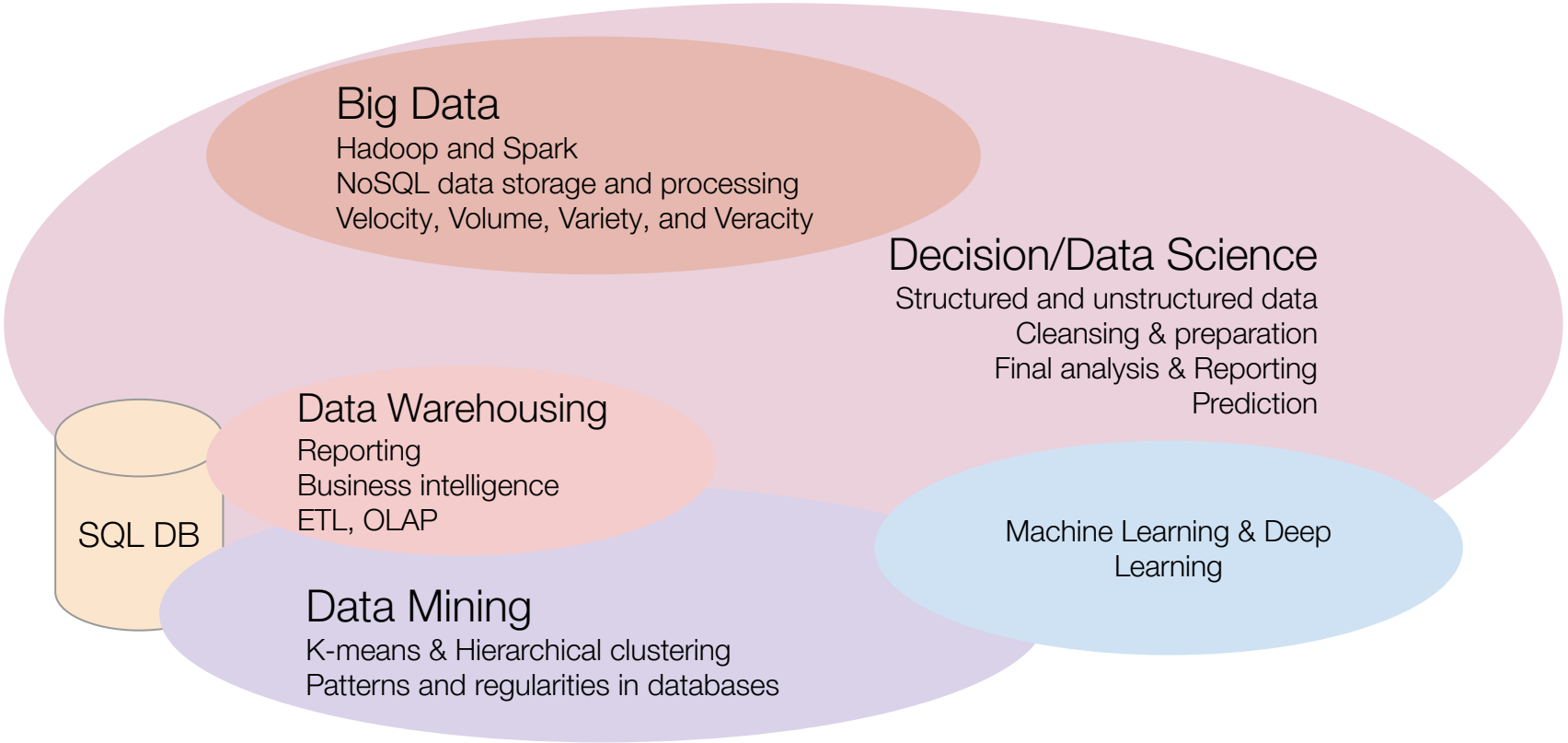
1980s

Deep Learning



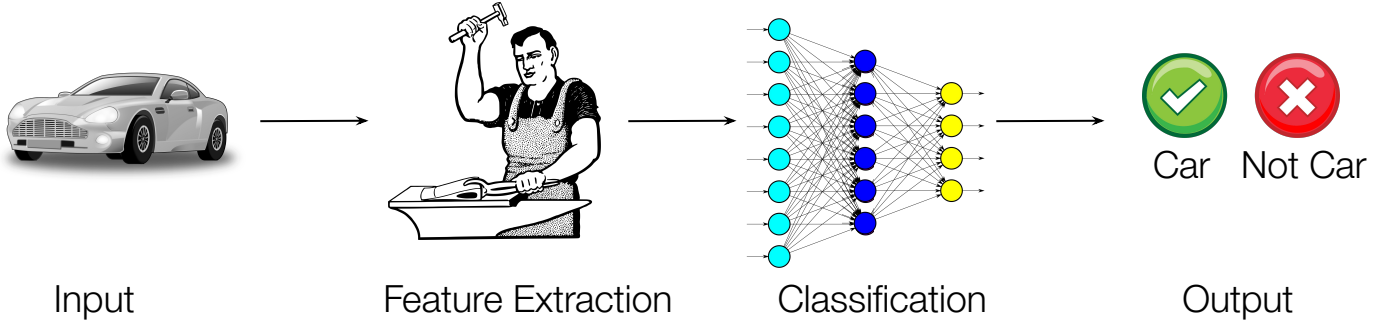
Powerful & trending
methods

2010s

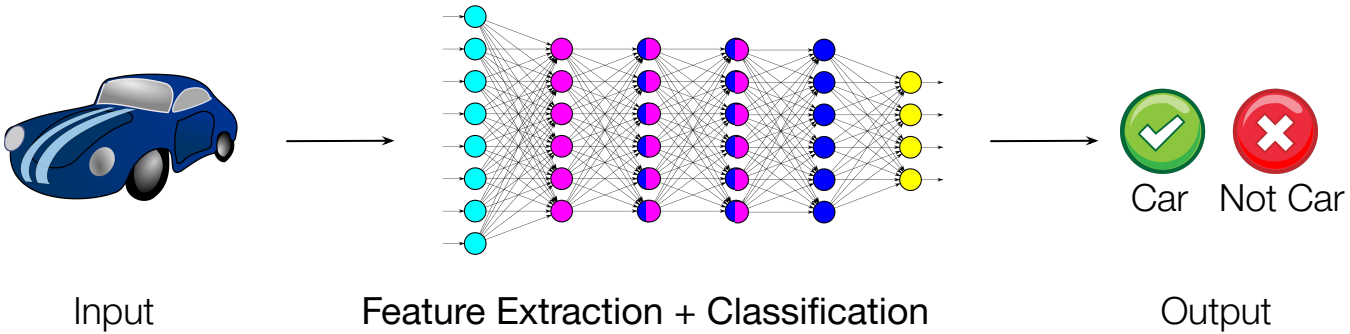

Big Data
Hadoop and Spark
NoSQL data storage and processing
Velocity, Volume, Variety, and Veracity
Decision/Data Science
Structured and unstructured data
Cleansing & preparation
Final analysis & Reporting
Prediction
Data Warehousing
Reporting
Business intelligence
ETL, OLAP
SQL DB
Machine Learning & Deep Learning
Data Mining
K-means & Hierarchical clustering
Patterns and regularities in databases

1.1 ML & DL

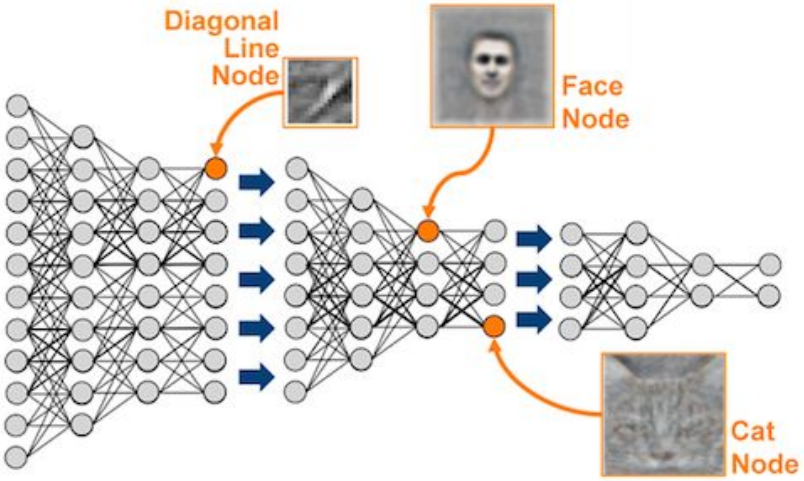
Machine Learning

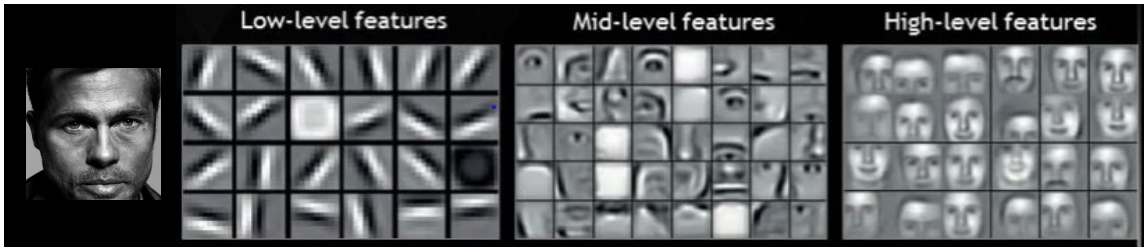


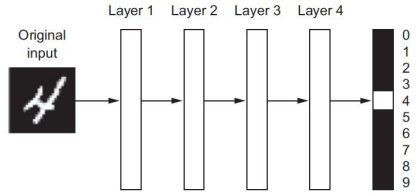
Deep Learning

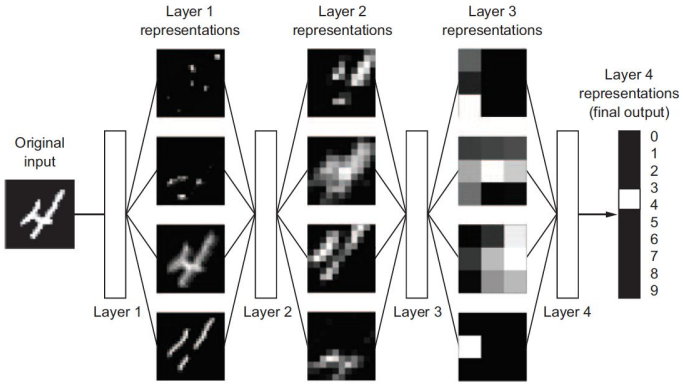


1.1.4 DL is Hierarchical Feature Learning



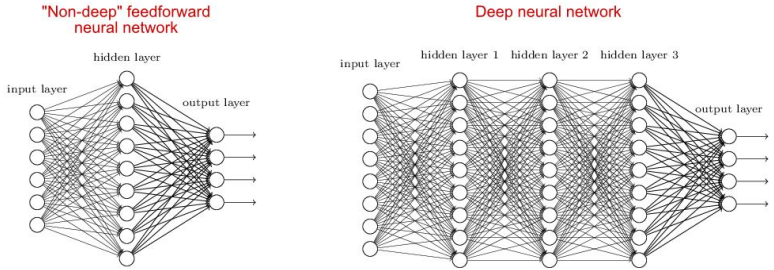






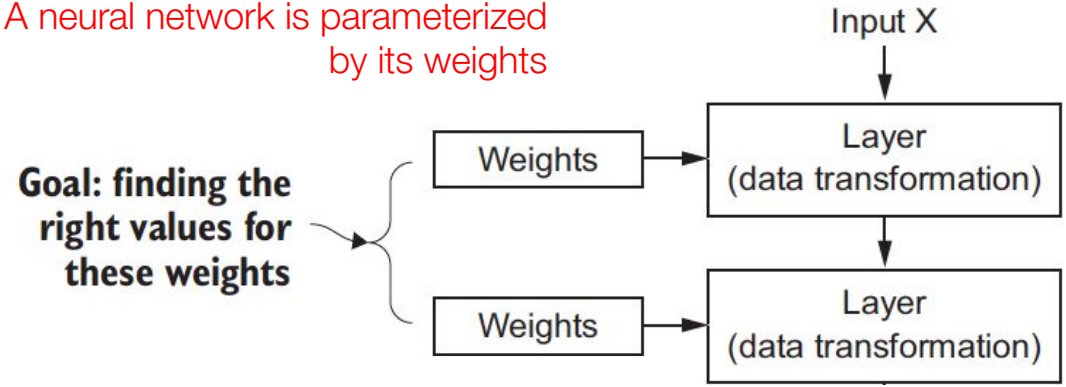
1.1.4 The “Deep” in Deep Learning

- Shallow Learning
  - ML methods that focus on learning only one or two layers of representations of the data
  - After two hidden layers, performance starts to drop
- Deep Learning
  - Other appropriate names for the field could have been
  - Modern deep learning often involves tens or even hundreds of successive layers of representations
    - they’re all learned automatically from exposure to training data
- Alternative names
  - layered representations learning and hierarchical representations learning
- Deep learning is a mathematical framework for learning representations from data



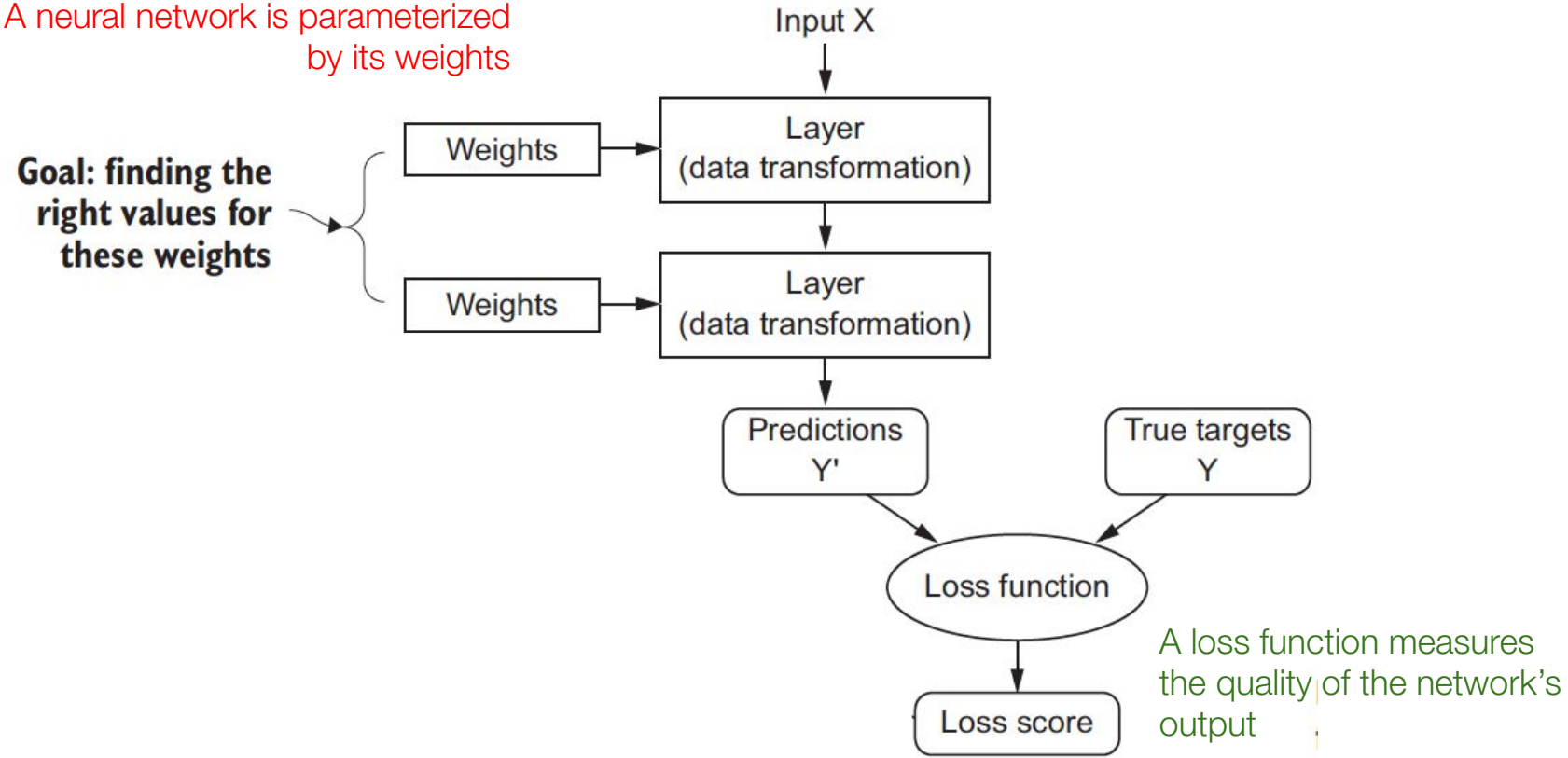
1.1.5 How Does Machine Learning (or DL) Work?

A neural network is parameterized by its weights



1.1.5 How Does Machine Learning (or DL) Work?

A neural network is parameterized by its weights

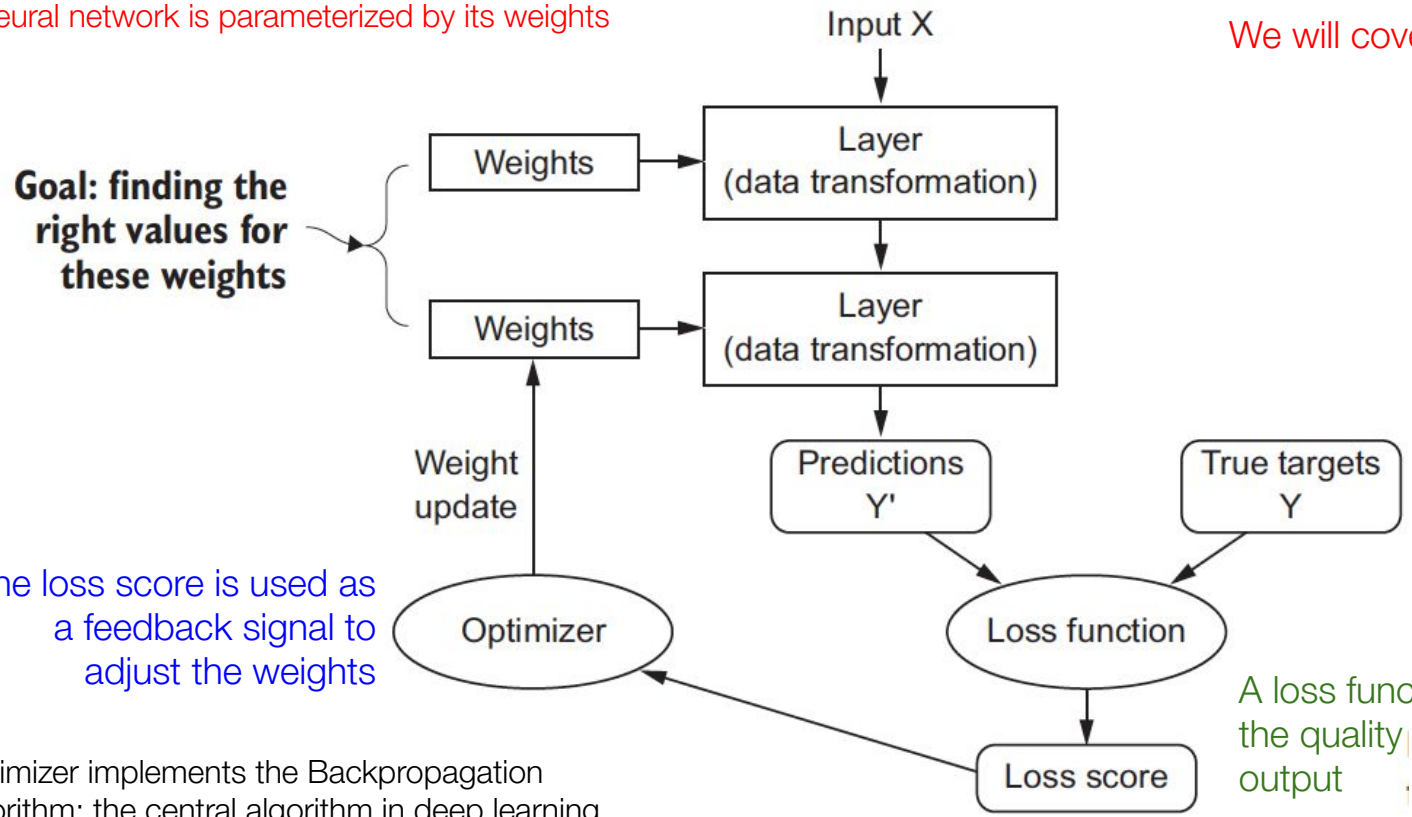


A loss function measures the quality of the network's output

1.1.5 How Does Machine Learning (or DL) Work?

A neural network is parameterized by its weights

We will cover backpropagation later!



The loss score is used as a feedback signal to adjust the weights

A loss function measures the quality of the network's output

Optimizer implements the Backpropagation algorithm: the central algorithm in deep learning

1.3 Why Deep Learning? Why Now?

- The two key ideas of deep learning for computer vision—convolutional neural networks and backpropagation—were already well understood in 1989
- The Long Short-Term Memory (LSTM) algorithm, which is fundamental to deep learning for time series, was developed in 1997 and has barely changed since
- So why did deep learning only take off after 2012? What changed in these two decades?

1.3 Why Deep Learning? Why Now?

- The two key ideas of deep learning for computer vision—convolutional neural networks and backpropagation—were already well understood in 1989
- The Long Short-Term Memory (LSTM) algorithm, which is fundamental to deep learning for time series, was developed in 1997 and has barely changed since
- So why did deep learning only take off after 2012? What changed in these two decades?
- In general, three technical forces are driving advances in machine learning:
  - Hardware (GPUs)
  - Datasets and benchmarks
    - ImageNet dataset - 1.4 million images hand annotated with 1,000 image categories
    - Kaggle competitions
  - Algorithmic advances
    - Better activation functions and better weight-initialization schemes
    - Better optimization schemes, such as RMSProp and Adam

http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

2.2 Data Representations for Neural Networks

- *Tensors* are multidimensional Numpy arrays
- A tensor is a container for data—almost always numerical data
  - it's a container for numbers
- All current machine-learning systems use tensors as their basic data structure
- Matrices are 2D tensors
  - Tensors are a generalization of matrices to an arbitrary number of dimensions

Matrix vs Tensor

- A matrix is a grid of n × m (say, 3 × 3) numbers surrounded by brackets
  - We can add and subtract matrices of the same size, multiply one matrix with another as long as the sizes are compatible ((n × m) × (m × p) = n × p), and multiply an entire matrix by a constant
  - A vector is a matrix with just one row or column

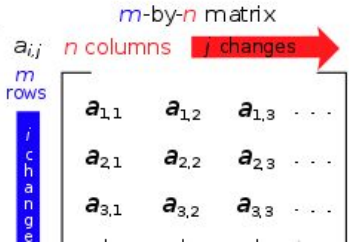
Matrix (mathematics)

From Wikipedia, the free encyclopedia

For other uses, see Matrix.

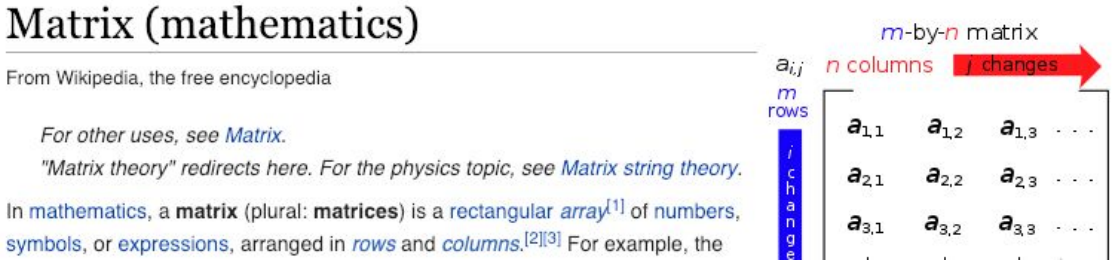
"Matrix theory" redirects here. For the physics topic, see Matrix string theory.

In mathematics, a **matrix** (plural: **matrices**) is a rectangular *array*[1] of numbers, symbols, or expressions, arranged in *rows* and *columns*.[2][3] For example, the



Matrix vs Tensor

- A matrix is a grid of n × m (say, 3 × 3) numbers surrounded by brackets
  - We can add and subtract matrices of the same size, multiply one matrix with another as long as the sizes are compatible ((n × m) × (m × p) = n × p), and multiply an entire matrix by a constant
  - A vector is a matrix with just one row or column

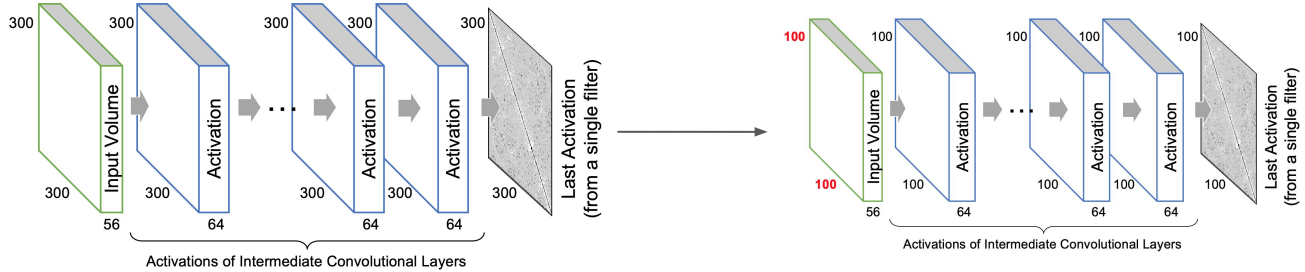


- A tensor is a generalized matrix
  - It could be a 1-D matrix (a vector is actually such a tensor)
  - A 3-D matrix (something like a cube of numbers)
  - Or even a 0-D matrix (a single number),
  - Or a higher dimensional structure that is harder to visualize

Matrix vs Tensor

It is not just that tensors are a generalization of matrices!

- A tensor is a mathematical entity that lives in a structure and interacts with other mathematical entities
  - If one transforms the other entities in the structure in a regular way, then the tensor must obey a related transformation rule

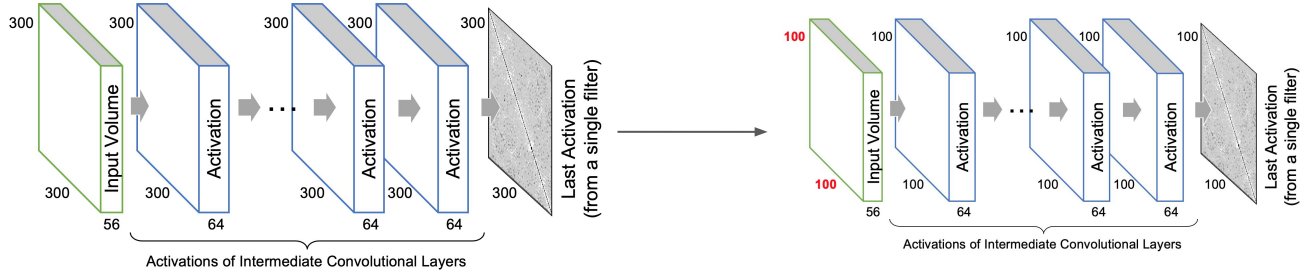


- This “dynamical” property of a tensor is the key that distinguishes it from a mere matrix
  - It’s a team player whose numerical values shift around along with those of its teammates when a transformation is introduced that affects all of them

Matrix vs Tensor

It is not just that tensors are a generalization of matrices!

- A tensor is a mathematical entity that lives in a structure and interacts with other mathematical entities
  - If one transforms the other entities in the structure in a regular way, then the tensor must obey a related transformation rule



2.2.5 A Tensor in “Tensorflow”

- A tf.Tensor has the following properties:
  - Number of axes (rank)
  - Data type (float32, int32, or string, for example)
  - Shape

2.2.5 A Tensor in “Tensorflow”

- A tf.Tensor has the following properties:
  - Number of axes (rank)
  - Data type (float32, int32, or string, for example)
  - Shape
- The **rank** of a tf.Tensor object is its number of dimensions
  - Synonyms for rank: **order** or **degree** or **n-dimension**
- The **shape** of a tensor
  - is the number of elements in each dimension

Rank	Shape	Dimension number	Example
0	[]	0-D	A 0-D tensor. A scalar.
1	[D0]	1-D	A 1-D tensor with shape [5].
2	[D0, D1]	2-D	A 2-D tensor with shape [3, 4].
3	[D0, D1, D2]	3-D	A 3-D tensor with shape [1, 4, 3].
n	[D0, D1, ... Dn-1]	n-D	A tensor with shape [D0, D1, ... Dn-1].

Rank	Math entity
0	Scalar (magnitude only)
1	Vector (magnitude and direction)
2	Matrix (table of numbers)
3	3-Tensor (cube of numbers)
n	n-Tensor (you get the idea)

2.2.5 A Tensor in “Tensorflow”

- A tf.Tensor has the following properties:
  - Number of axes (rank)
  - Data type (float32, int32, or string, for example)
  - Shape
- The **rank** of a tf.Tensor object is its number of dimensions
  - Synonyms for rank: **order** or **degree** or **n-dimension**
- The **shape** of a tensor
  - is the number of elements in each dimension

Rank	Shape	Dimension number	Example
0	[]	0-D	A 0-D tensor. A scalar.
1	[D0]	1-D	A 1-D tensor with shape [5].
2	[D0, D1]	2-D	A 2-D tensor with shape [3, 4].
3	[D0, D1, D2]	3-D	A 3-D tensor with shape [1, 4, 3].
n	[D0, D1, ... Dn-1]	n-D	A tensor with shape [D0, D1, ... Dn-1].

Rank
0
1
2
3
n

- Tensors have a **data type**
  - It is not possible to have a tf.Tensor with more than one data type

- `tf.float16` : 16-bit half-precision floating-point.
- `tf.float32` : 32-bit single-precision floating-point.
- `tf.float64` : 64-bit double-precision floating-point.
- `tf.bfloat16` : 16-bit truncated floating-point.
- `tf.complex64` : 64-bit single-precision complex.
- `tf.complex128` : 128-bit double-precision complex.
- `tf.int8` : 8-bit signed integer.
- `tf.uint8` : 8-bit unsigned integer.
- `tf.uint16` : 16-bit unsigned integer.
- `tf.uint32` : 32-bit unsigned integer.
- `tf.uint64` : 64-bit unsigned integer.
- `tf.int16` : 16-bit signed integer.
- `tf.int32` : 32-bit signed integer.
- `tf.int64` : 64-bit signed integer.
- `tf.bool` : Boolean.
- `tf.string` : String.
- `tf.qint8` : Quantized 8-bit signed integer.
- `tf.quint8` : Quantized 8-bit unsigned integer.
- `tf.qint16` : Quantized 16-bit signed integer.
- `tf.quint16` : Quantized 16-bit unsigned integer.
- `tf.qint32` : Quantized 32-bit signed integer.
- `tf.resource` : Handle to a mutable resource.
- `tf.variant` : Values of arbitrary types.

2.2.1 Scalars (0D tensors)

- A tensor that contains only one number is called a scalar (or scalar tensor, or 0-dimensional tensor, or 0D tensor)
  - In Numpy, a float32 or float64 number is a scalar tensor (or scalar array)
- You can display the number of axes of a Numpy tensor via the ndim attribute
  - a scalar tensor has 0 axes (ndim == 0)
  - The number of axes of a tensor is also called its rank

```
>>> import numpy as np
>>> x = np.array(12)
>>> x
array(12)
>>> x.ndim
0
```

2.2.2 Vectors (1D tensors)

- An array of numbers is called a vector, or 1D tensor
- A 1D tensor is said to have exactly one axis

```
>>> x = np.array([12, 3, 6, 14])
>>> x
array([12, 3, 6, 14])
>>> x.ndim
1
```

2.2.3 Matrics (2D tensors)

- An array of vectors is a matrix, or 2D tensor
- A matrix has two axes (often referred to rows and columns)
- You can visually interpret a matrix as a rectangular grid of numbers

```
>>> x = np.array([[5, 78, 2, 34, 0],
                  [6, 79, 3, 35, 1],
                  [7, 80, 4, 36, 2]])
>>> x.ndim
2
```

- Example:
  - Pima Indian Diabetes Dataset

2.2.4 3D Tensors and Higher-dimensional Tensors

- If you pack 2D matrices in a new array, you obtain a 3D tensor
  - you can visually interpret as a cube of numbers

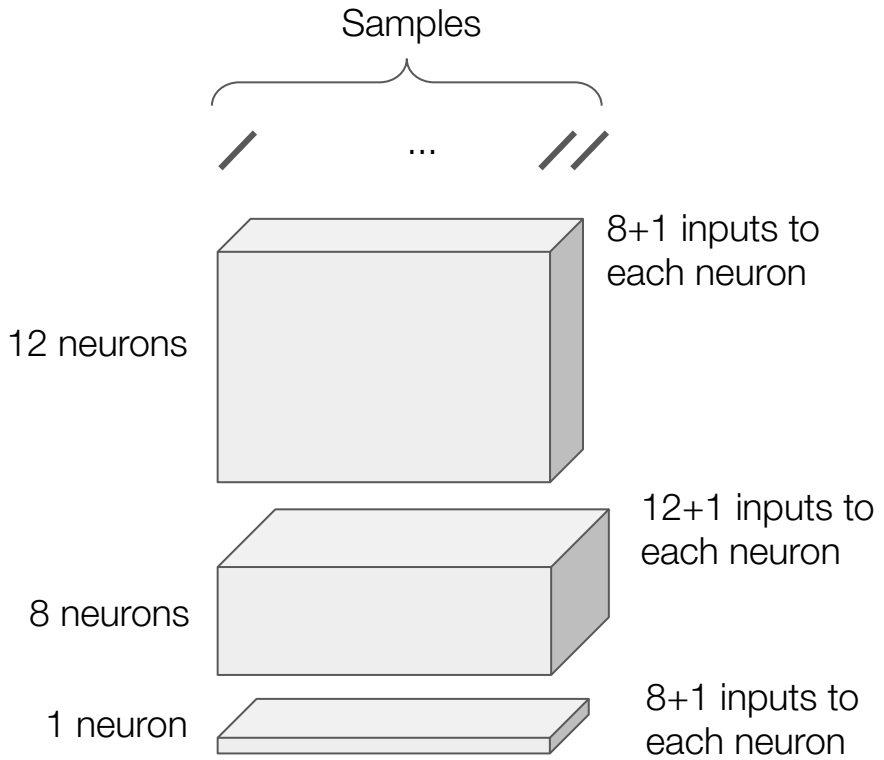
```
>>> x = np.array([[[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]]])
>>> x.ndim
3
```

- Examples:
  - Pima Indian Dataset over a period of time (time series data)
  - Input image volume (RGB)

Examples of 4D and 5D tensor data!

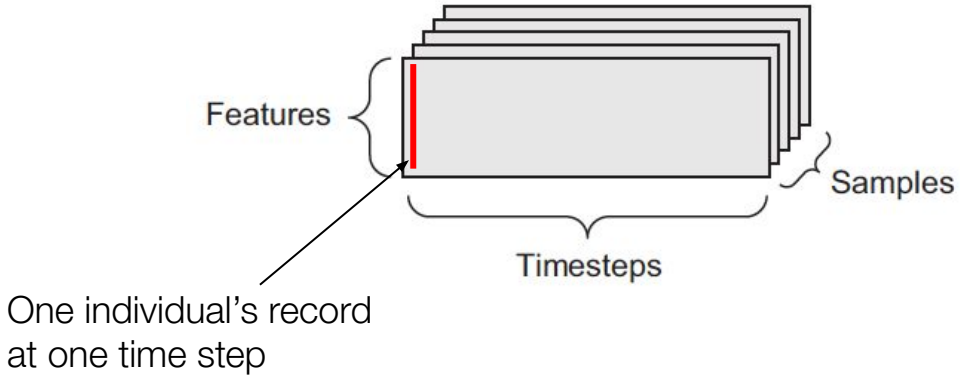
What happens to other tensors' shapes when input shape changes?

```
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
print(model.summary())
```



2.2.10 Time series Data or Sequence Data

- Whenever time matters in your data (or the notion of sequence order), it makes sense to store it in a 3D tensor with an explicit time axis
- Each sample can be encoded as a sequence of vectors (a 2D tensor), and thus a batch of data will be encoded as a 3D tensor
- The time axis is always the second axis (axis of index 1), by convention



2.2.10 Time series Data or Sequence Data

Suppose that we have the Pima Indian Diabetes dataset over last 4 years. What are some possible ways of structuring our inputs/outputs (and the corresponding NN models)?

THINK PAIR SHARE

Year 1

Year 2

Year 3

Year 4

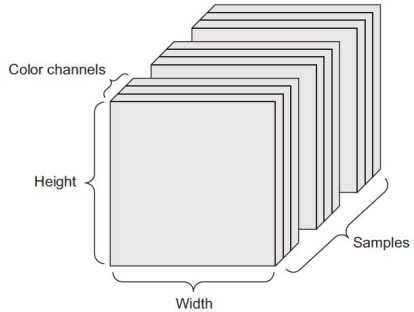
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

2.2.11 Image Data

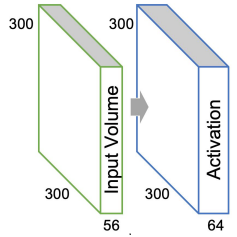
- Images typically have three dimensions: height, width, and color depth
  - Although grayscale images (like the MNIST digits) have only a single color channel and could thus be stored in 2D tensors, by convention image tensors are always 3D, with a one dimensional color channel for grayscale images
- A batch of 128 grayscale images of size 256×256 could thus be stored in a tensor of shape (128, 256, 256, 1), and a batch of 128 color images could be stored in a tensor of shape ?

2.2.11 Image Data

- Images typically have three dimensions: height, width, and color depth
  - Although grayscale images (like the MNIST digits) have only a single color channel and could thus be stored in 2D tensors, by convention image tensors are always 3D, with a one dimensional color channel for grayscale images
- A batch of 128 grayscale images of size 256 × 256 could thus be stored in a tensor of shape (128, 256, 256, 1), and a batch of 128 color images could be stored in a tensor of shape ?
- There are two conventions for shapes of images tensors: the channels-last convention (used by TensorFlow) and the channels-first convention (Theano).



Is this channels-first or channels last?



2.2.12 Video Data

- Video data is one of the few types of real-world data for which you'll need 5D tensors
- A video can be understood as a sequence of frames, each frame being a color image
  - Because each frame can be stored in a 3D tensor (height, width, color_depth), a sequence of frames can be stored in a 4D tensor (frames, height, width, color_depth)
  - Thus a batch of different videos can be stored in a 5D tensor of shape (samples, frames, height, width, color_depth)

2.2.12 Video Data

- Video data is one of the few types of real-world data for which you'll need 5D tensors
- A video can be understood as a sequence of frames, each frame being a color image
  - Because each frame can be stored in a 3D tensor (height, width, color_depth), a sequence of frames can be stored in a 4D tensor (frames, height, width, color_depth)
  - Thus a batch of different videos can be stored in a 5D tensor of shape (samples, frames, height, width, color_depth)
- A 60-second, 144 × 256 YouTube video clip sampled at 4 frames per second would have 240 frames
  - A batch of four such video clips would be stored in a tensor of shape (4, 240, 144, 256, 3)
  - That's a total of 106,168,320 values!
  - If the dtype of the tensor was float32, then each value would be stored in 32 bits, so the tensor would represent 405 MB
  - (The data is usually compressed using MPEG format)

2.3 The Gears of Neural Networks: Tensor Operations

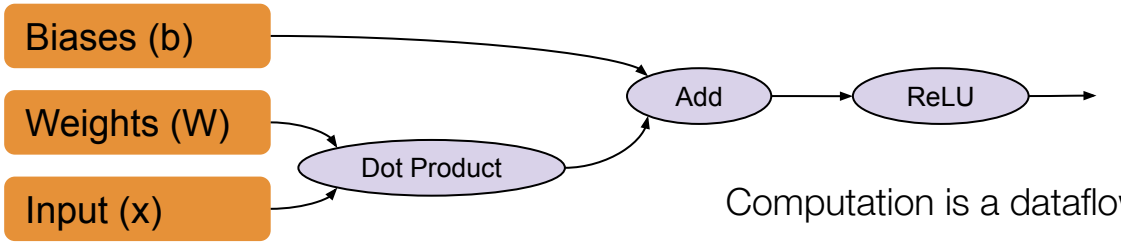
- Building our network by stacking Dense layers on top of each other:

```
keras.layers.Dense(512, activation='relu')
```

- This layer can be interpreted as a function, which takes as input a 2D tensor and returns another 2D tensor (where W is a 2D tensor and b is a vector)

```
output = relu(dot(W, input) + b)
```

- We have three tensor operations here:
  - a dot product (dot) between the input tensor and a tensor named W
  - an addition (+) between the resulting 2D tensor and a vector b
  - a relu operation - relu(x) is max(x, 0)



Computation is a dataflow graph

2.3.1 Element-wise Operations

- Naive implementation of an element-wise relu operation:

```
def naive_relu(x):
    assert len(x.shape) == 2    <-- x is a 2D Numpy tensor.

    x = x.copy()                <-- Avoid overwriting the input tensor.
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            x[i, j] = max(x[i, j], 0)
    return x
```

- With Numpy arrays, these operations are available as well-optimized built-in Numpy functions
  - These delegate the heavy lifting to a Basic Linear Algebra Subprograms (BLAS) / cuBLAS
  - BLAS are low-level, highly parallel, efficient tensor-manipulation routines that are typically implemented in Fortran or C

```
import numpy as np

z = x + y           <-- Element-wise addition

z = np.maximum(z, 0.)           <-- Element-wise relu
```

2.3.2 Broadcasting

- With broadcasting, we can apply two-tensor element-wise operations if one tensor has shape (a, b, … n, n + 1, … m) and the other has shape (n, n + 1, … m)

```
import numpy as np

x = np.random.random((64, 3, 32, 10))
y = np.random.random((32, 10))

z = np.maximum(x, y)
```

x is a random tensor with shape (64, 3, 32, 10).

y is a random tensor with shape (32, 10).

The output z has shape (64, 3, 32, 10) like x.

2.3.3 Tensor Dot

```
import numpy as np

z = np.dot(x, y)
```

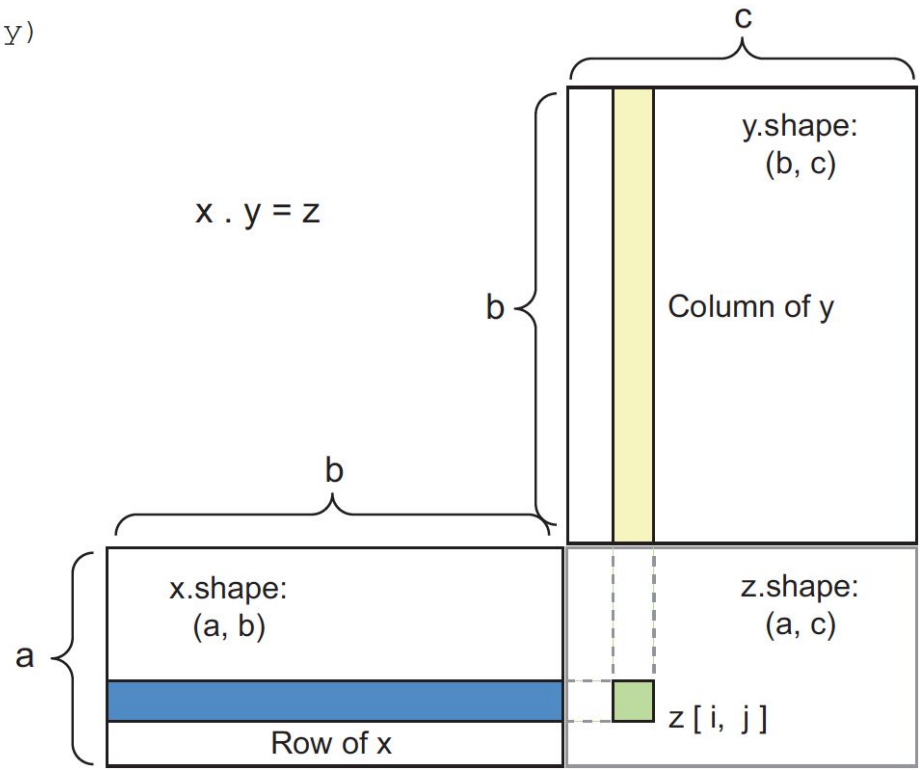


Figure 2.5 Matrix dot-product box diagram

2.3.4 Tensor Reshaping

- Reshaping a tensor is rearranging its rows and columns to match a target shape
- The reshaped tensor has the same total number of coefficients as the initial tensor

```
train_images = train_images.reshape(( 60000, 28, 28, 1 ))
```

```
>>> x = np.array([[0., 1.],
                  [2., 3.],
                  [4., 5.]])
>>> print(x.shape)
(3, 2)
```

```
>>> x = x.reshape((6, 1))
>>> x
array([[ 0.],
       [ 1.],
       [ 2.],
       [ 3.],
       [ 4.],
       [ 5.]])
```

```
>>> x = x.reshape((2, 3))
>>> x
array([[ 0.,  1.,  2.],
       [ 3.,  4.,  5.]])
```

```
>>> x = np.zeros((300, 20))
>>> x = np.transpose(x)
>>> print(x.shape)
(20, 300)
```

Creates an all-zeros matrix of shape (300, 20)

2.3.5 Geometric Interpretation of Tensor Operations

- The ‘contents of the tensors manipulated by tensor operations’ can be interpreted as ‘coordinates of points in some geometric space’
  - This implies that all tensor operations have a geometric interpretation
- If `A = [0.5, 1]` is a vector, we can picture the vector as an arrow linking the origin to the point

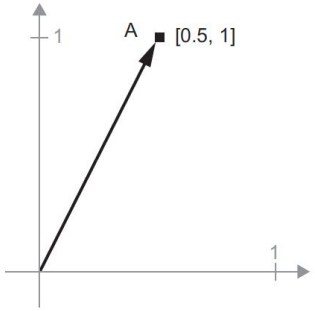


Figure 2.7 A point in a 2D space pictured as an arrow

- We add a new point, `B = [1, 0.25]` to A
  - the result is a new location, a vector representing the sum of the previous two vectors

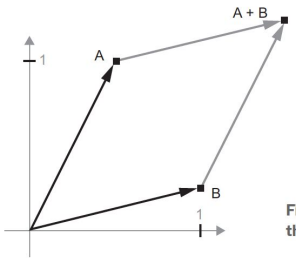
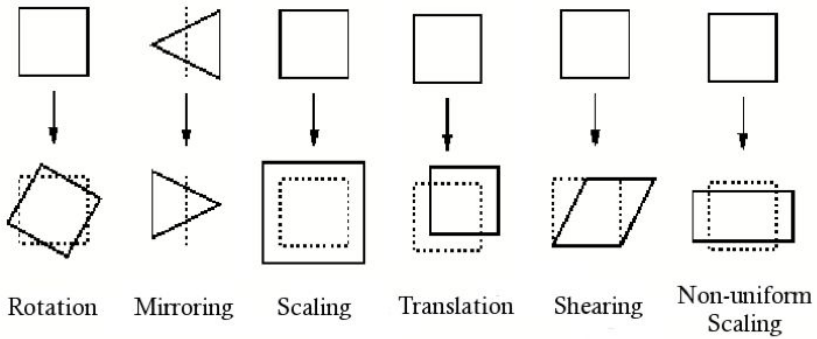


Figure 2.8 Geometric interpretation of the sum of two vectors

2.3.5 Geometric Interpretation of Tensor Operations

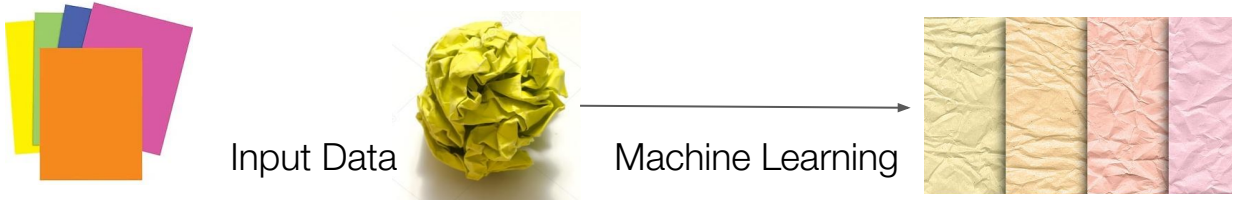
- This means that elementary geometric operations such as affine transformations, (rotations, scaling, etc.) can be expressed as tensor operations



- For instance, a rotation of a 2D vector by an angle `theta` can be achieved via a dot product with a 2 × 2 matrix `R = [u, v]`, where `u` and `v` are both vectors of the plane: `u = [cos(theta), sin(theta)]` and `v = [-sin(theta), cos(theta)]`

2.3.6 A Geometric Interpretation of Deep Learning

- Neural networks consist entirely of chains of tensor operations and that all of these tensor operations are just geometric transformations of the input data
  - We can interpret a neural network as a very complex geometric transformation in a high-dimensional space, implemented via a long series of simple steps
- Imagine two sheets of colored paper: one red and one blue
  - Put one on top of the other & crumple them together into a small ball
  - The crumpled paper ball is your input data, and each sheet of paper is a class of data in a classification problem.
- What a neural network (or any other machine-learning model) is meant to do is figure out a transformation of the paper ball that would uncrumple it, so as to make the two classes cleanly separable again



2.4 The Engine of NNs: Gradient-based Optimization

- A simple neural network:

```
output = relu(dot(W, input) + b)
```

- Initially, these weight matrices are filled with small random values (a step called random initialization)
- Training is the gradual adjustment of the weights

2.4 The Engine of NNs: Gradient-based Optimization

- A simple neural network:

```
output = relu(dot(W, input) + b)
```

- Initially, these weight matrices are filled with small random values (a step called random initialization)
- Training is the gradual adjustment of the weights
- Training loop:
  1. Draw a batch of training samples `x` and corresponding targets `y`
  2. Run the network on `x` to obtain predictions `y_pred` (forward pass)
  3. Compute the loss of the network on the batch, a measure of the mismatch between `y_pred` and `y`
  4. Compute the gradient of the loss with regard to the network's parameters (backward pass)
  5. Move the parameters a little in the opposite direction from the gradient—for example `W -= step * gradient`—thus reducing the loss on the batch a bit

3.2 Keras

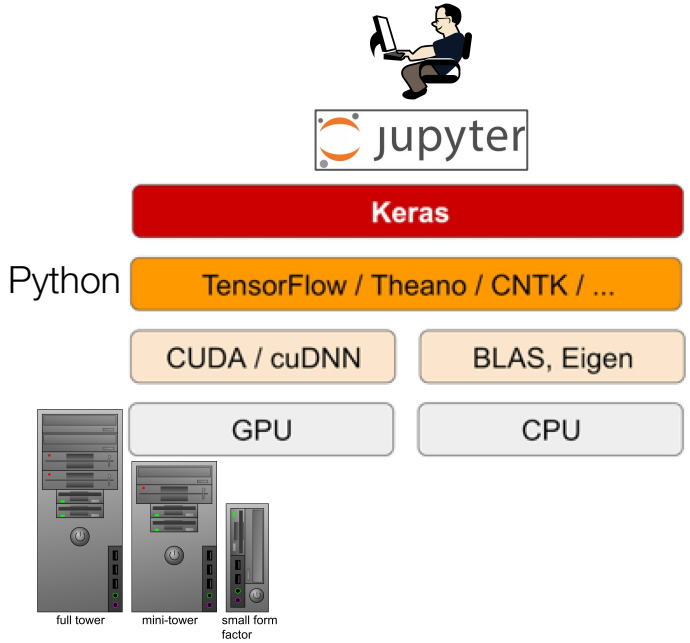
- Keras is a deep-learning framework for Python that provides a convenient way to define and train almost any kind of deep-learning model
- Keras was initially developed for researchers, with the aim of enabling fast experimentation

3.2 Keras

- Keras is a deep-learning framework for Python that provides a convenient way to define and train almost any kind of deep-learning model
- Keras was initially developed for researchers, with the aim of enabling fast experimentation
- 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 almost any deep-learning model
  - It also supports multi-GPU training
- Keras can be freely used in commercial projects (permissive MIT license)

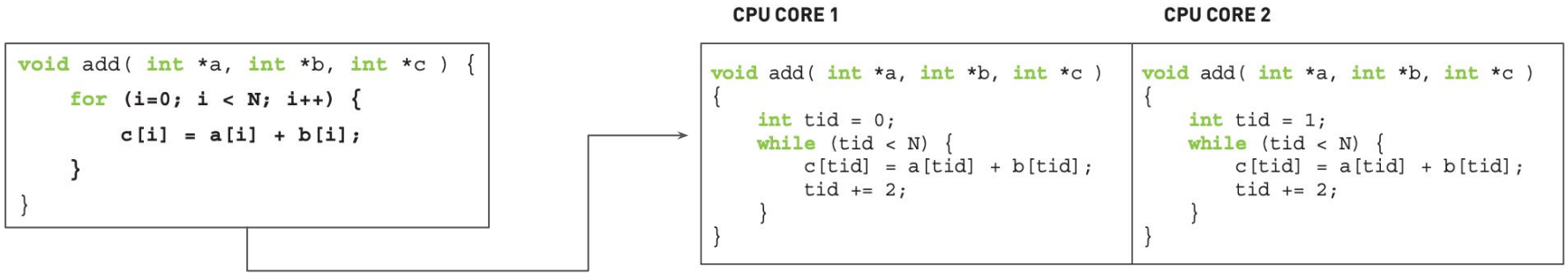
3.2.1 Keras, TensorFlow, & Jupyter Notebooks

- Keras doesn’t handle low-level operations such as tensor manipulation and differentiation
- Theano is developed by the MILA lab at Université de Montréal
- TensorFlow is developed by Google
- CNTK is developed by Microsoft
- Jupyter is developed by Project Jupyter
  - a nonprofit organization created to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages"



Cuda

- CUDA is a parallel computing platform and application programming interface (API) model created by Nvidia
  - C/C++ programmers use 'CUDA C/C++', compiled with **nvcc**, Nvidia's C/C++ compiler
- It allows software developers and software engineers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing
- The cuBLAS library is an implementation of BLAS (Basic Linear Algebra Subprograms) on top of the NVIDIA CUDA runtime
  - It allows the user to access the computational resources of NVIDIA Graphics Processing Unit (GPU)



Tensorflow vs Keras

In [3]:
```
# Training Data
train_X = numpy.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,2.167,
                         7.042,10.791,5.313,7.997,5.654,9.27,3.1])
train_Y = numpy.asarray([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.221,
                         2.827,3.465,1.65,2.904,2.42,2.94,1.3])
n_samples = train_X.shape[0]
```

In [4]:
```
# tf Graph Input
X = tf.placeholder("float")
Y = tf.placeholder("float")

# Set model weights
W = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")
```

In [5]:
```
# Construct a linear model
pred = tf.add(tf.multiply(X, W), b)
```

In [6]:
```
# Mean squared error
cost = tf.reduce_sum(tf.pow(pred-Y, 2))/(2*n_samples)
# Gradient descent
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

In [8]:
```
# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()
```

In [9]:
```
# Start training
with tf.Session() as sess:
    sess.run(init)

    # Fit all training data
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y: y})

        #Display logs per epoch step
        if (epoch+1) % display_step == 0:
            c = sess.run(cost, feed_dict={X: train_X, Y:train_Y})
            print "Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(c), \
                "W=", sess.run(W), "b=", sess.run(b)

    print "Optimization Finished!"
    training_cost = sess.run(cost, feed_dict={X: train_X, Y: train_Y})
    print "Training cost=", training_cost, "W=", sess.run(W), "b=", sess.run(b), '\n'
```

3.3.2 Getting Keras Running

- Options:
  - Google Colab
  - Microsoft Azure
  - Install everything locally
    - For frequent/heavy usage and large datasets

- Local Installation
  - Ubuntu version for your hardware (problems with windows
  - CUDA driver version, CUDA version & CuDNN version
  - Python version & libraries' versions
  - Anaconda & Jupyter Notebook
  - Set up Tensorboard correctly

Install all of these in your Laptop & Show it to me. (Windows is fine)

+1 bonus point

- It took me about a week to setup my first deep learning hardware platform!!

3.4 Classifying Movie Reviews

Classify movie reviews as positive or negative, based on the text content of the reviews

★ 10/10

Breathtaking!
angelo-29 6 June 2000

Once again, Director Ridley Scott proves to b
recreation of ancient Rome is splendid, with
and he is able to show with all brightness th
political problems and military discipline, and
still have a place in our world today. But the

★ 7/10

Victory of production over a lousy script
CherryBlossomBoy 22 October 2006

There is an unwritten rule in movie making that if you have an excellent script it doesn't matter how you film it. If you have a lousy script - then everything else matters, the direction, the cinematography, the acting, the music... "Gladiator" is an epitome of the rule. The case where direction, music, acting and art direction ned a crappy story.

tching it in German overdubbing. As I don't speak German there was me from visuals and music. And it's a new experience on another that later. First the crappy part. The story.

whether you care about history or not, this is a horrible script. It's quite unacceptable) liberties with the period it purports to portray, it's

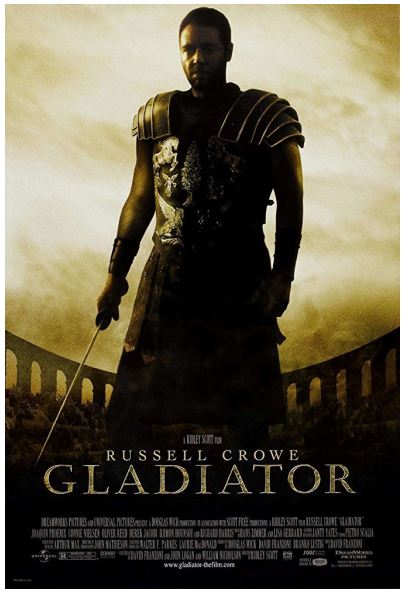
★ 6/10

Pretentious and Mediocre
screenman 6 January 2008

Evaluating opinion of this movie by the 'Hated It' selection, I lost count of the number of contributors who gave it just one star. I thought they were being unnecessarily 'Roman' in their ruthlessness, but the result does show how disappointed so many viewers were and the extent to which this movie has polarised opinion.

I diplomatically place myself somewhere in the middle.

For me, scepticism set-in at the outset. That much-vaunted battle scene didn't sit right at all. In combat, the greatest Roman strength lay in the legion. The scrupulously drilled and disciplined foot-soldiers worked in unison, advancing upon enemies in their orderly units and maniples.Their Shields interlocked to create a mobile wall that could form-up on all flanks and even provide overhead protection. In between, they used their short,



3.4.1 The IMDB Dataset

- A set of 50,000 highly polarized reviews from the Internet Movie Database
  - Split into 25,000 reviews for training and 25,000 reviews for testing
  - Each set consisting of 50% negative and 50% positive reviews

```
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

- The argument `num_words=10000` means you'll only keep the top 10,000 most frequently occurring words in the training data (Rare words will be discarded)

```
train_data[0]
```

```
[1,
 14,
 22,
 16,
 43,
 530,
 973,
 1622,
```

```
train_labels[0]
```

```
1
```

What is the shape of the four numpy matrices?

3.4.1 The IMDB Dataset

```
# word_index is a dictionary mapping words to an integer index
word_index = imdb.get_word_index()
# We reverse it, mapping integer indices to words
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
# We decode the review; note that our indices were offset by 3
# because 0, 1 and 2 are reserved indices for "padding", "start of sequence", and "unknown".
decoded_review = ' '.join([reverse_word_index.get(i - 3, '?') for i in train_data[0]])
```

```
decoded_review
```

```
"? this film was just brilliant casting location scenery story direction everyone's really suited
the part they played and you could just imagine being there robert ? is an amazing actor and now t
he same being director ? father came from the same scottish island as myself so i loved the fact t
here was a real connection with this film the witty remarks throughout the film were great it was
just brilliant so much that i bought the film as soon as it was released for ? and would recommend
it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and
you know what they say if you cry at a film it must have been good and this definitely was also ?
to the two little boy's that played the ? of norman and paul they were just brilliant children are
often left out of the ? list i think because the stars that play them all grown up are such a big
profile for the whole film but these children are amazing and should be praised for what they have
done don't you think the whole story was so lovely because it was true and was someone's life afte
r all that was shared with us all"
```

How will you design your NN?

THINK PAIR SHARE

IMDB Dataset

					Label
Review 1	11	23	...	89	1
Review 2	67	78	...	546	0
Review 3	134	234	...	56	1

Here's what we did for the Pima Indian Dataset!

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

3.4.2 Preparing Data: One-hot Encoding

- One-hot encode your lists to turn them into vectors of 0s and 1s
- This would mean - turning the sequence [3, 5] into a 10,000 dimensional vector that would be all 0s except for indices 3 and 5, which would be 1s
  - Then you could use as the first layer in your network a Dense layer, capable of handling floating-point vector data

Label Encoding

Review #				Label
1	terrible	movie	eeh	0
2	terribly	fascinating	enjoyed	1
3	hell	good	movie	1
4	damn	good	good	1

One-hot Encoding

Review #	terrible	hell	damn	movie	fascinating	good	eeh	enjoyed	direction	Label
1	1	0	0	1	0	0	1	0	0	0
2	1	0	0	0	1	0	0	1	0	1
3	0	1	0	1	0	1	0	0	0	1
4	0	0	1	0	0	1	0	0	0	1

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results
```

Creates an all-zero matrix of shape (len(sequences), dimension)

Sets specific indices of results[i] to 1s

```
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

```
>>> x_train[0]
array([ 0.,  1.,  1., ...,  0.,  0.,  0.])
```

3.4.3 Building Your Network

- The input data is vectors, and the labels are scalars (1s and 0s)
  - this is the easiest setup you'll ever encounter
- A type of network that performs well on such a problem is a simple stack of fully connected (Dense) layers with relu activations:
  - `Dense(16, activation='relu')`
- Each such `Dense()` layer with relu activation is equivalent to:
  - `output = relu(dot(W, input) + b)`

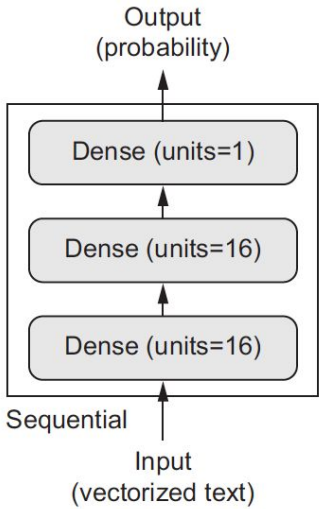
3.4.3 Building Your Network

- The input data is vectors, and the labels are scalars (1s and 0s)
  - this is the easiest setup you'll ever encounter
- A type of network that performs well on such a problem is a simple stack of fully connected (Dense) layers with relu activations:
  - `Dense(16, activation='relu')`
- Each such `Dense()` layer with relu activation is equivalent to:
  - `output = relu(dot(W, input) + b)`
- Having more hidden units (a higher-dimensional representation space) allows your network to learn more-complex representations
- But
  - it makes the network more computationally expensive
  - may lead to learning unwanted patterns (patterns that will improve performance on the training data but not on the test data)

3.4.3 Building Your Network

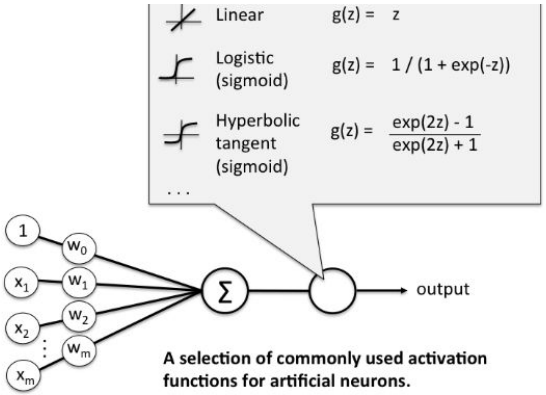
- There are two key architecture decisions to be made about such a stack of Dense layers:
  - How many layers to use?
  - How many hidden units to choose for each layer?

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```



Activation Functions

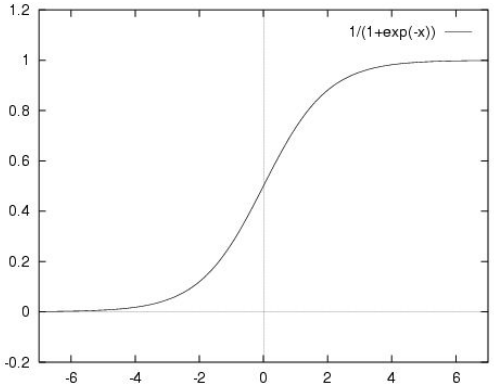
- No activation function = A simple linear function
  - Easy to solve
  - A linear function is just a polynomial of one degree
  - Limited in complexity
  - Have less power to learn complex functional mappings from data
- A Neural Network without Activation function = Linear regression model
  - No matter how many layers you have



A selection of commonly used activation functions for artificial neurons.

Sigmoid Activation function

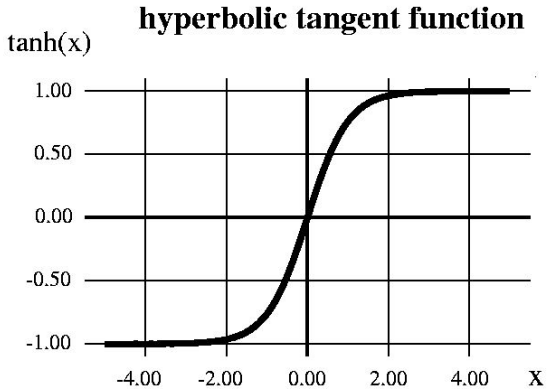
- `f(x) = 1 / 1 + exp(-x)`
  - Range is 0 to 1
  - S — shaped curve
  - Easy to understand and apply
- Problems:
  - Vanishing gradient problem
  - Its output isn't zero centered (0 < output < 1)
    - makes the gradient updates go too far in different directions
    - makes optimization harder
  - Slow convergence



Hyperbolic Tangent function

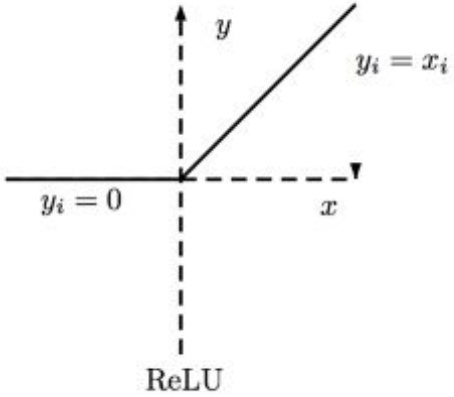
- `f(x) = 1 — exp(-2x) / 1 + exp(-2x)`
  - Range is -1 to 1
  - Output is zero centered
  - Optimization is easier
    - In practice, always preferred over Sigmoid

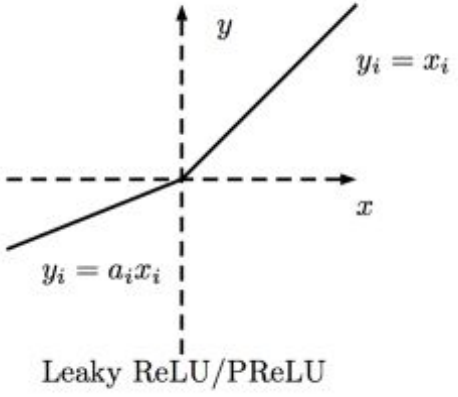
- Still, it suffers from vanishing gradient problem

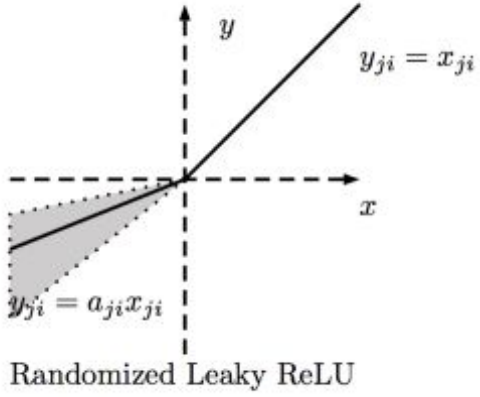


Rectified Linear units

- `R(x) = max(0,x)`
  - It has become very popular in the past few years





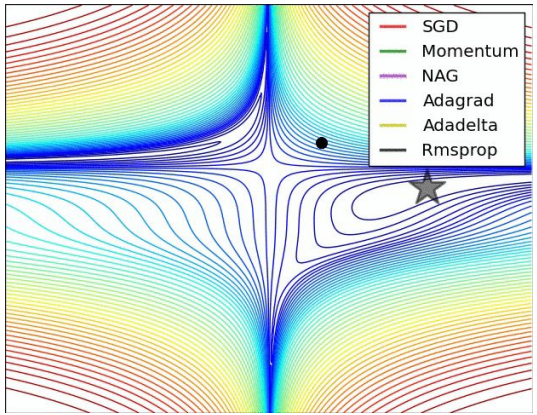


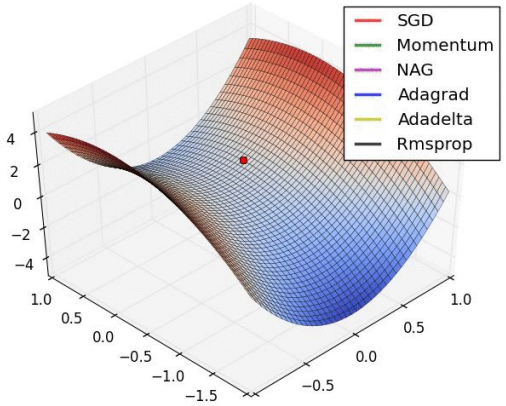
Optimizers

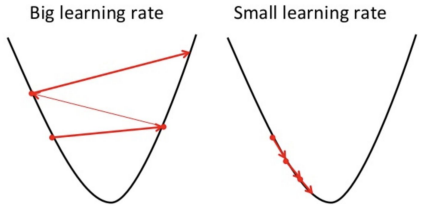
- Options (https://keras.io/optimizers/)
  - Stochastic gradient descent (sgd), RMSProp, Adagrad, Adadelta, Adam, Adamax, Nadam
- How to decide?

http://ruder.io/optimizing-gradient-descent/

  - Sparse input data is sparse - use adaptive learning-rate methods
  - Try all once with the default learning rates
  - Trend is to use SGD without momentum and a simple learning rate annealing schedule







Tip (when developing methods on new datasets): Debug by observing the trends of the training loss and validation loss.

Loss Functions

- Options (https://keras.io/losses/#available-loss-functions)
  - Mean squared error
  - Mean absolute error
  - Cross entropy

MAE and MSE

- Mean Absolute Error (L1 loss)

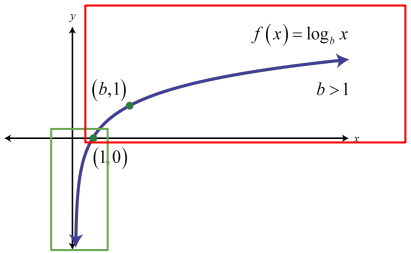
```
def L1(yHat, y):
    return np.sum(np.absolute(yHat - y))
```

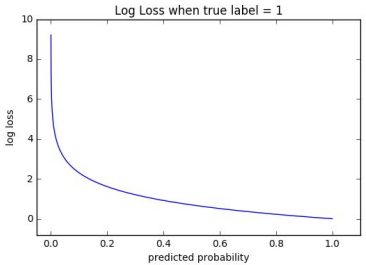
- Mean Squared Error (L2 loss)

```
def MSE(yHat, y):
    return np.sum((yHat - y)**2) / y.size
```

Cross Entropy (or Log Loss)

- Measures the performance of a classification model whose output is a ‘probability’ value between 0 and 1 (natural log)
- Log loss penalizes both types of errors, but especially those predictions that are confident and wrong!
- Loss increases as the predicted probability diverges from actual labels (0 or 1)
  - Predicting a probability of .012 when the actual observation label is 1 would result in a high loss value
  - Similarly predicting 0.97 when the true label is 0 results in high loss
- Negative output when the input is < 1
  - So, add a minus for a positive loss



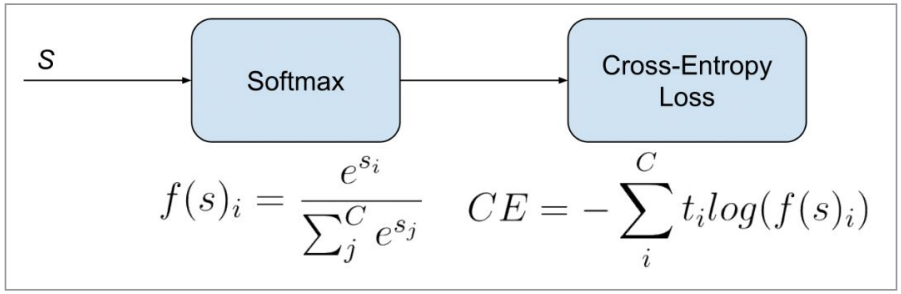


$$-(y \log(p) + (1 - y) \log(1 - p))$$

```
def binary_cross_entropy(pred, y):
      if y == 1:
            return -log(y)
      else:
            return -log(1-pred)
```

Categorical Cross-Entropy Loss (or Softmax Loss)

- Categorical Cross-Entropy loss is the most commonly used loss
- It is a Softmax activation plus a Cross-Entropy loss



- Example:

True Label: Rabbit

Prediction: Dog = 0.1, Cat = 0.4, Rabbit = 0.8, Squirrel = 0.2

```
CE Loss  =  - (0 * ln(0.1) + 0 * ln(0.4) + 1 * ln(0.8) + 0 * ln(0.2))
         =  - (0 + 0 + (-0.916) + 0)
         =  0.916
```

3.4.4 Validating Your Approach (a Validation Set)

- During training, in order to monitor the accuracy of the model on data it has never seen before
  - create a validation set by setting apart 10,000 samples from the original training data (50k)

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
```

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

```
history = model.fit(partial_x_train, partial_y_train, nb_epoch=20, batch_size=512, validation_data
=(x_val, y_val))
```

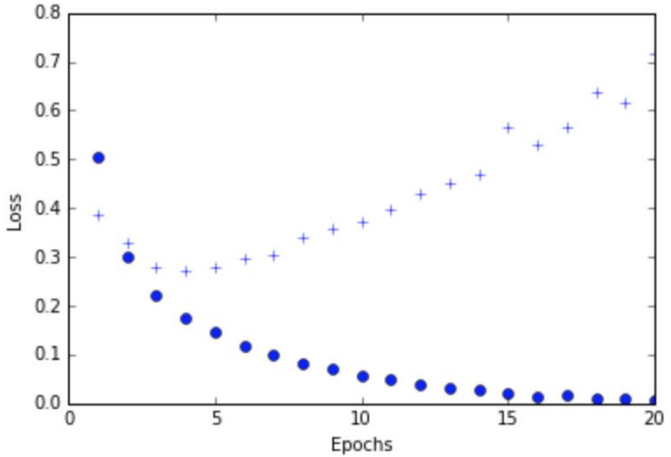
```
history_dict = history.history
history_dict.keys()
```

```
dict_keys(['val_loss', 'loss', 'val_binary_accuracy', 'binary_accuracy'])
```

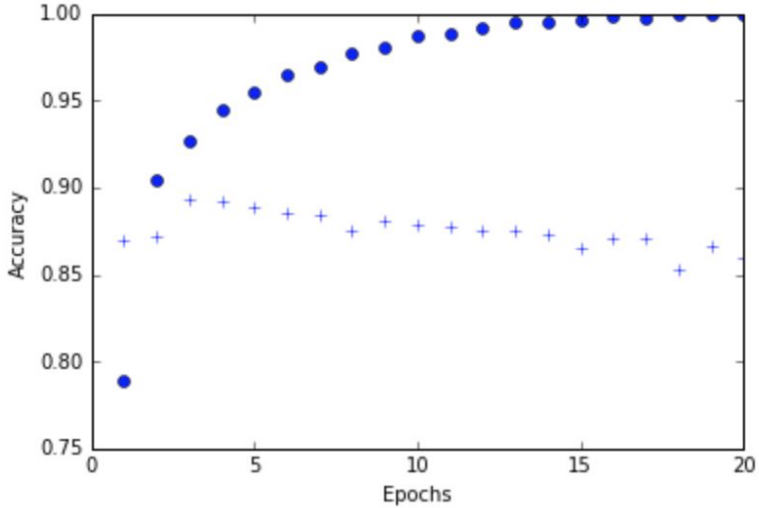
Why history? 

3.4.4 Validating Your Approach

```
import matplotlib.pyplot as plt
%matplotlib inline

loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values) + 1)

plt.plot(epochs, loss_values, 'bo')
plt.plot(epochs, val_loss_values, 'b+')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```




```
plt.clf()

acc_values = history_dict['binary_accuracy']
val_acc_values = history_dict['val_binary_accuracy']
plt.plot(epochs, acc_values, 'bo')
plt.plot(epochs, val_acc_values, 'b+')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.show()
```



We will talk about overfitting and underfitting later! But what can we do right now to obtain a more "general" model?

THINK PAIR SHARE

3.4.4 Validating Your Approach

```
model = Sequential()

model.add(Dense(16, activation='relu', input_dim=10000))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, nb_epoch=4, batch_size=512)

results = model.evaluate(x_test, y_test)
```

```
results
```

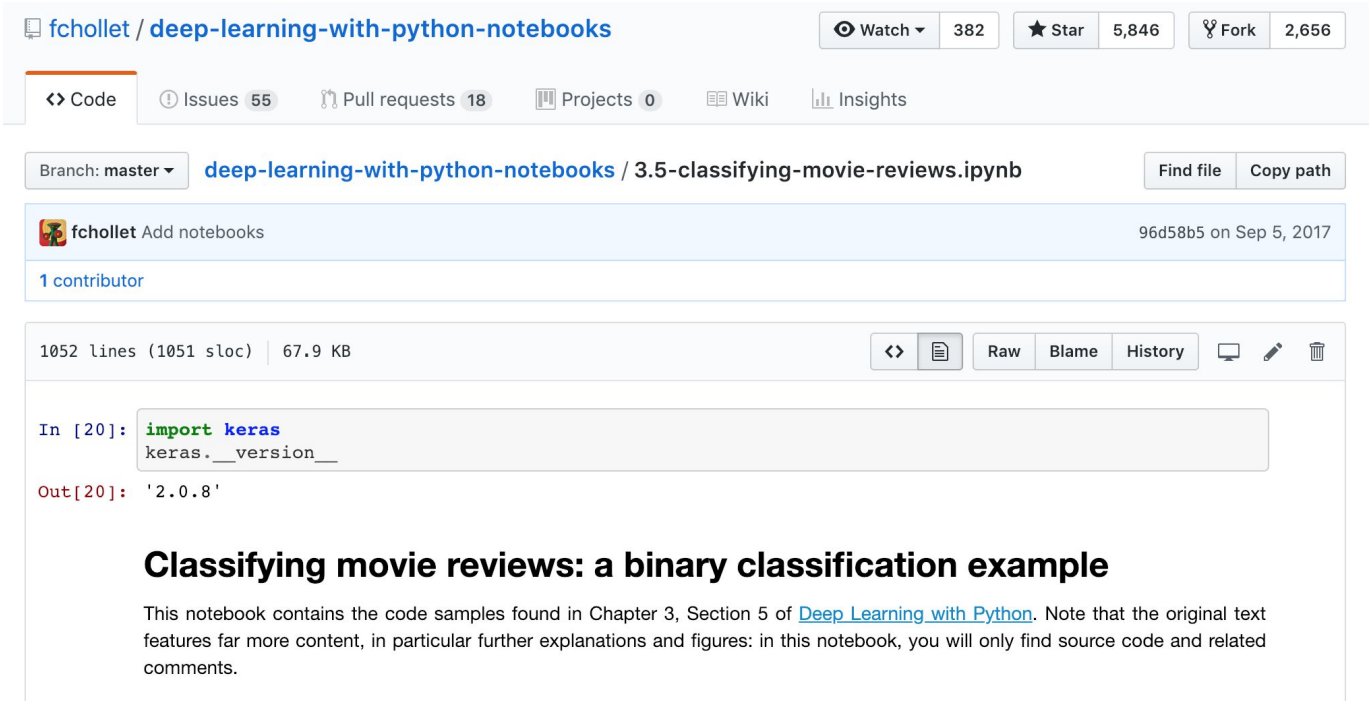
```
[0.2918469704437256, 0.88532]
```

```
model.predict(x_test)
```

```
array([[ 0.93180436],
       [ 0.80974817],
       [ 0.99873918],
       ...,
       [ 0.57409382],
       [ 0.00224084],
       [ 0.76358223]], dtype=float32)
```

1. What was the size of x_train/y_train? What did we use it for?
2. What was the size of x_test/y_test? What did we use it for?
3. What was the size of partial_x_train/partial_y_train? What did we use it for?
4. What was the size of x_val/y_val? What did we use it for?

+1 bonus point



https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/3.5-classifying-movie-reviews.ipynb

3.6 Predicting House Prices: a Regression Example

- Regression is predicting a continuous value instead of a discrete label
- Regression and the algorithm logistic regression are different
  - logistic regression isn't a regression algorithm—it's a classification algorithm

3.6 Predicting House Prices: a Regression Example

- Regression is predicting a continuous value instead of a discrete label
- Regression and the algorithm logistic regression are different
  - logistic regression isn't a regression algorithm—it's a classification algorithm

- **Problem:** Predict the median price of homes in a given Boston suburb in the Mid-1970s given data points about the suburb at the time
  - such as the crime rate, the local property tax rate, and so on

3.6 Predicting House Prices: a Regression Example

- Regression is predicting a continuous value instead of a discrete label
- Regression and the algorithm logistic regression are different
  - logistic regression isn't a regression algorithm—it's a classification algorithm

- **Problem:** Predict the median price of homes in a given Boston suburb in the Mid-1970s given data points about the suburb at the time
  - such as the crime rate, the local property tax rate, and so on

- It has relatively few data points: only 506, split between 404 training samples and 102 test samples
- Each feature in the input data (for example, the crime rate) has a different scale
  - Some values are proportions, which take values between 0 and 1; others take values between 1 and 12, others between 0 and 100, and so on

3.6.2 Preparing the Data

```
from keras.datasets import boston_housing

(train_data, train_targets), (test_data, test_targets) =  boston_housing.load_data()
```

```
train_data.shape
```

```
(404, 13)
```

```
test_data.shape
```

```
(102, 13)
```

```
train_targets
```

```
array([ 15.2,  42.3,  50. ,  21.1,  17.7,  18.5,  11.3,  15.6,  15.6,
        14.4,  12.1,  17.9,  23.1,  19.9,  15.7,   8.8,  50. ,  22.5,
        24.1,  27.5,  10.9,  30.8,  32.9,  24. ,  18.5,  13.3,  22.9,
        34.7,  16.6,  17.5,  22.3,  16.1,  14.9,  23.1,  34.9,  25. ,
```

```
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std

test_data -= mean
test_data /= std
```

Sam's well trained model predicts "NaN" for some inputs. Upon investigating he find some of this test_data have inputs as "NaN". What would be a possible cause? What is the solution?

Normalization / Standardization / Feature Scaling

- Standardization (most widely used)
  - Works well for populations that are normally distributed
  - Output can be -ve (bad for methods that expect positive inputs, e.g. RBMs)

$$x' = \frac{x - \bar{x}}{\sigma}$$

- Rescaling (min-max normalization)
  - Very fast

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- Mean normalization

$$x' = \frac{x - \text{average}(x)}{\max(x) - \min(x)}$$

- Why data normalization or standardization?
  - In stochastic gradient descent, feature scaling can sometimes improve the convergence speed of the algorithm
  - In support vector machines, it can reduce the time to find support vectors

Validation vs Testing

- In machine learning, what should be done first - test or validate?
- What do we do validation for? What do we do test for?
- What is wrong with doing only of of the two?

val·i·date
/ˈvaləˌdāt/

verb

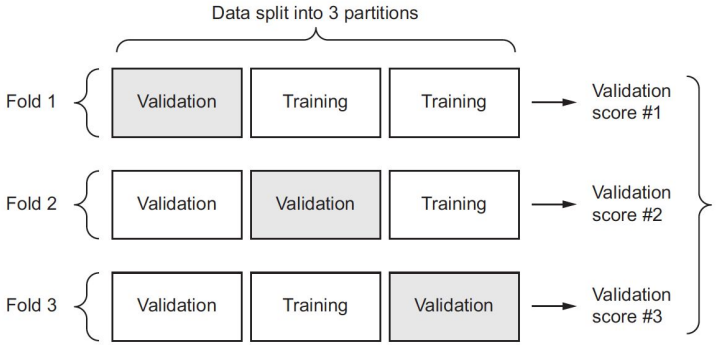
check or prove the validity or accuracy of (something).
"these estimates have been validated by periodic surveys"

test[1]
/test/

noun

1. a procedure intended to establish the quality, performance, or reliability of something, especially before it is taken into widespread use.
"no sparking was visible during the tests"
synonyms: trial, experiment, test case, case study, pilot study, trial run, tryout, dry run; More

3.6.4 Validating Your Approach Using K-fold Validation



Prepares the validation data: data from partition #k

Prepares the training data: data from all other partitions

```
for i in range(k):
    print('processing fold #', i)
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]

    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
         train_targets[(i + 1) * num_val_samples:]],
        axis=0)

    model = build_model()
    model.fit(partial_train_data, partial_train_targets,
              epochs=num_epochs, batch_size=1, verbose=0)
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
    all_scores.append(val_mae)
```

Builds the Keras model (already compiled)

Trains the model (in silent mode, verbose = 0)

Evaluates the model on the validation data

How do you sell such a model/s to a client?

https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/3.7-predicting-house-prices.ipynb

4.1 Four Branches of Machine Learning

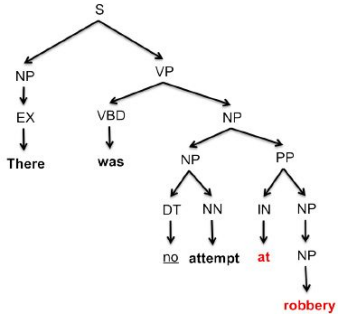
- Machine learning is a vast field with a complex subfield taxonomy
- Machine-learning algorithms generally fall into four broad categories:
  a) Supervised learning
  b) Unsupervised learning
  c) Self-supervised learning
  d) Reinforcement learning

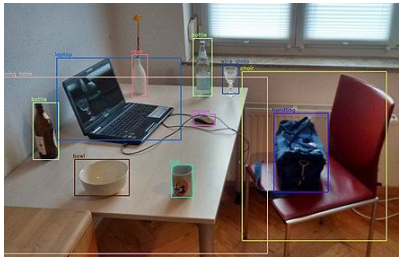
4.1.1 Supervised Learning

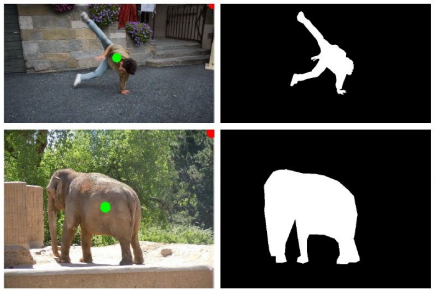
- It consists of learning to map input data to known targets (also called annotations), given a set of examples (often annotated by humans)
- Some ‘uncommon’ examples:
  - *Sequence generation*—Given a picture, predict a caption describing it
  - *Syntax tree prediction*—Given a sentence, predict its decomposition into a syntax tree
  - *Object detection*—Given a picture, draw a bounding box around certain objects inside the picture
  - *Image segmentation*—Given a picture, draw a pixel-level mask on a specific object

A female tennis player in action on the court.

A group of young men playing a game of soccer







https://machinelearningmastery.com/how-to-caption-photos-with-deep-learning/

4.1.2 Unsupervised Learning

- Finding interesting transformations of the input data without the help of any targets
  - For the purposes of (a) data visualization, (b) data compression, (c) data denoising, or (d) to better understand the correlations present in the data at hand
- Unsupervised learning is the bread and butter of data analytics
  - It's often a necessary step in better understanding a dataset before attempting to solve a supervised-learning problem (basic NN training is slower when two features are highly correlated)
- "Dimensionality reduction" and "clustering" are well-known categories of unsupervised learning, e.g. Principal component analysis

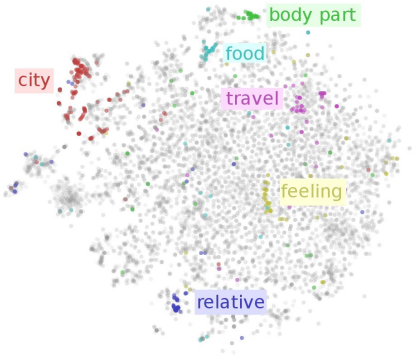


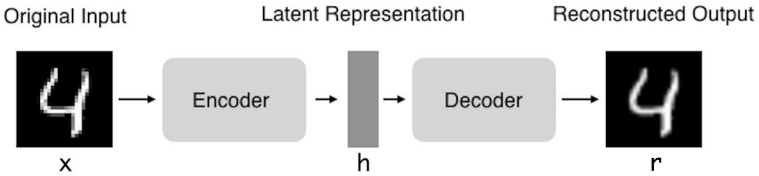
Image Restoration - https://en.wikipedia.org/wiki/Image_restoration

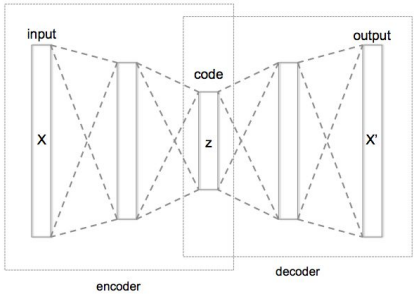
PCA - https://www.youtube.com/watch?v=HMOI_lkzW08

Correlation vs Causation - https://towardsdatascience.com/why-correlation-does-not-imply-causation-5b99790df07e

4.1.3 Self-supervised Learning

- Self-supervised learning is supervised learning without human-annotated label
  - supervised learning without any humans in the loop
- There are still labels involved
  - they're generated from the input data, typically using a heuristic algorithm
- "autoencoders" are an instance of self-supervised learning





- Other examples:
  - Trying to predict the next frame in a video, given past frames
  - Trying to predict next word in a text, given previous words

4.1.3 Self-supervised Learning

- The distinction between supervised, self-supervised, and unsupervised learning can be blurry sometimes
  - These categories are more of a continuum without solid borders
  - Self-supervised learning can be reinterpreted as either supervised or unsupervised learning, depending on whether you pay attention to the learning mechanism or to the context of its application

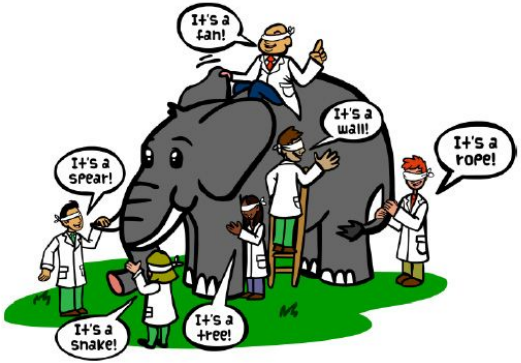
4.1.4 Reinforcement Learning

- Reinforcement learning is mostly a research area and hasn't yet had significant practical successes beyond games
- In reinforcement learning, an agent receives information about its environment and learns to choose actions that will maximize some reward
    - For instance, a neural network that "looks" at a videogame screen and outputs game actions in order to maximize its score can be trained via reinforcement learning

4.2 Evaluating Machine-learning Models

- In machine learning, the goal is to achieve models that “generalize”
  - that perform well on never-before-seen data—and overfitting is the central obstacle

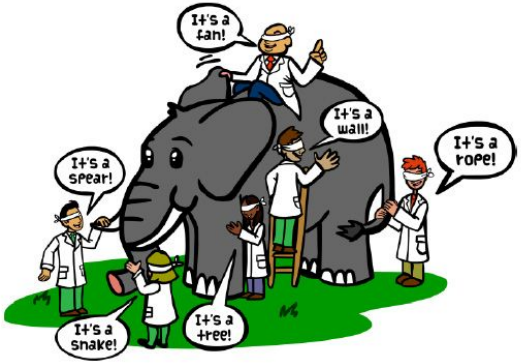
https://www.youtube.com/watch?v=bJVBQefNXIw



4.2 Evaluating Machine-learning Models

- In machine learning, the goal is to achieve models that “generalize”
  - that perform well on never-before-seen data—and overfitting is the central obstacle

    https://www.youtube.com/watch?v=bJVBQefNXIw

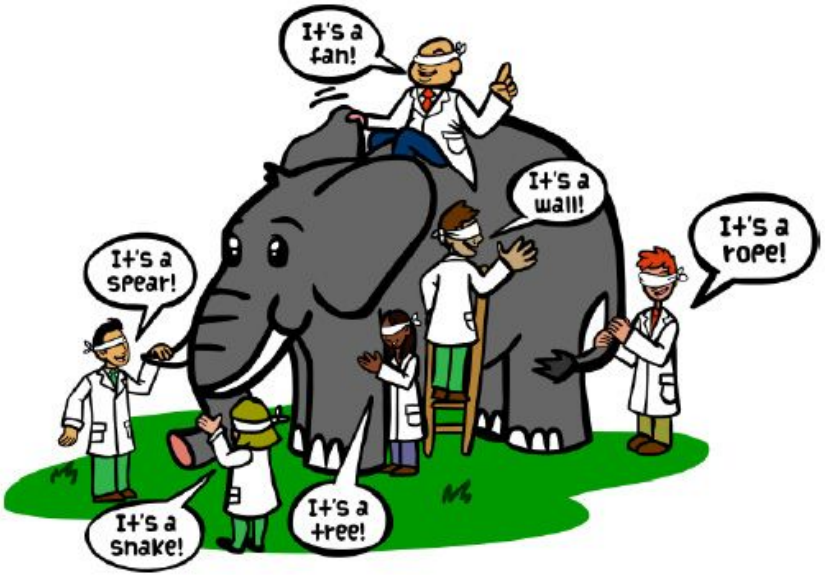


- You can only control that which you can observe
  - It’s crucial to be able to reliably measure the generalization power of your model
- It is important to learn how not to overfit and to maximize generalization
- Evaluating a machine learning model is “measuring generalization”

4.2 Evaluating Machine-learning Models

Optimization: It's a "spear + fan + wall + **rope** + tree + snake"

Generalization: It's an elephant (one side is snake like and the other side rope like, etc.)



4.2.1 Training, Validation, and Test sets

- For model evaluation - split the available data into three sets:
  - `training`, `validation`, and `test`
- You train on the training data and evaluate your model on the validation data
- Once your model is ready for prime time, you test it one final time on the test data
- Why not have two sets: a training set and a test set?
  - Hyperparameters - number of layers, number of neurons, activations, etc.
  - Parameters - network's weights

4.2.1 Training, Validation, and Test sets

- Why not have two sets: a training set and a test set?
- The key problem issue is the phenomenon of "**information leak**"
- Every time you tune a hyperparameter of your model based on the model's performance on the validation set, some information about the validation data leaks into the model
  - If you do this only once, for one parameter, then very few bits of information will leak, and your validation set will remain reliable to evaluate the model
  - But if you repeat this many times—running one experiment, evaluating on the validation set, and modifying your model as a result—then you'll leak an increasingly significant amount of information about the validation set into the model
- At the end of the day, you'll end up with a model that performs artificially well on the validation data, because that's what you optimized it for
  - We care about performance on completely new data, not the validation data, so you need to use a completely different, never-before-seen dataset to evaluate the model: the test dataset.
  -

4.2.1 Training, Validation, and Test sets

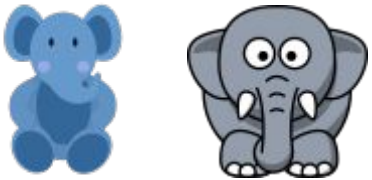
Explain the problem of information leak with reference to the “Blind men and the elephant” story!

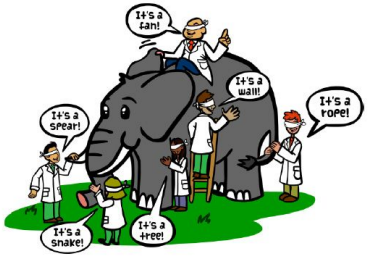
+1 bonus point

Training Data

Model

Test Data





4.2.1 Three Classic Evaluation Recipes

1. Simple Hold-out Validation
   - Set apart some fraction of your data as your test set
   - Train on the remaining data, and evaluate on the test set
   - In order to prevent information leaks
     - Don’t tune your model based on the test set
     - Reserve a separate validation set

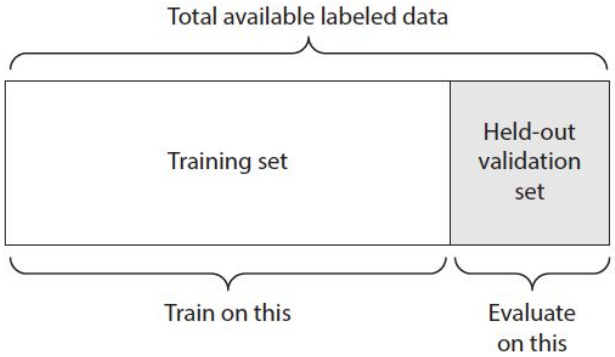


Figure 4.1 Simple hold-out validation split

4.2.1 Three Classic Evaluation Recipes

Listing 4.1 Hold-out validation

```
num_validation_samples = 10000

np.random.shuffle(data)

validation_data = data[:num_validation_samples]
data = data[num_validation_samples:]

training_data = data[:]

model = get_model()
model.train(training_data)
validation_score = model.evaluate(validation_data)

# At this point you can tune your model,
# retrain it, evaluate it, tune it again...

model = get_model()
model.train(np.concatenate([training_data,
                            validation_data]))
test_score = model.evaluate(test_data)
```

Shuffling the data is usually appropriate.

Defines the validation set

Defines the training set

Trains a model on the training data, and evaluates it on the validation data

Once you've tuned your hyperparameters, it's common to train your final model from scratch on all non-test data available.

4.2.1 Three Classic Evaluation Recipes

Limitation of the simple hold-out validation approach:

- If little data is available, then your validation and test sets may contain too few samples to be statistically representative of the data at hand

This is easy to recognize:

- If different random shuffling rounds of the data before splitting end up yields very different measures of model performance, then you have this issue

K-fold validation and iterated K-fold validation are two ways to address this!

4.2.1 Three Classic Evaluation Recipes

2. K-fold Validation

- Split your data into K partitions of equal size
- For each partition i, train a model on the remaining K – 1 partitions, and evaluate it on partition i
- Your final score is then the averages of the K scores obtained

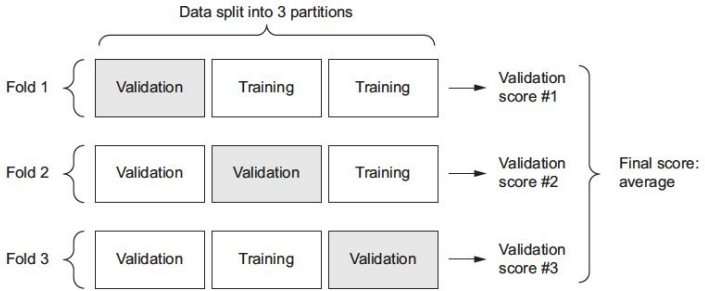


Figure 4.2 Three-fold validation

4.2.1 Three Classic Evaluation Recipes

2. K-fold Validation

- Split your data into K partitions of equal size
- For each partition i, train a model on the remaining K – 1 partitions, and evaluate it on partition i
- Your final score is then the averages of the K scores obtained

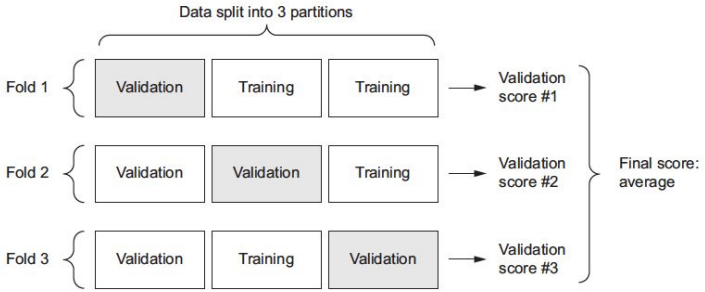


Figure 4.2 Three-fold validation

- When to use K-fold validation?
  - This method is helpful when the performance of your model shows significant variance based on your train-test split
  - Like hold-out validation, you can use a distinct validation set for model calibration

4.2.1 Three Classic Evaluation Recipes

3. Iterated K-fold validation with shuffling
   - Apply K-fold validation multiple times, shuffling the data every time before splitting it K ways
   - The final score is the average of the scores obtained at each run of K-fold validation
   - You end up training and evaluating P x K models (where P is the number of iterations you use)
     - can very expensive

4.2.1 Three Classic Evaluation Recipes

3. Iterated K-fold validation with shuffling
   - Apply K-fold validation multiple times, shuffling the data every time before splitting it K ways
   - The final score is the average of the scores obtained at each run of K-fold validation
   - You end up training and evaluating P x K models (where P is the number of iterations you use)
     - can very expensive

When is it appropriate?

- For situations in which you have relatively little data available and you need to evaluate your model as precisely as possible
- Winning teams in Kaggle competitions use K-fold validation with shuffling (for small data sets)

4.2.2 How to Choose an Evaluation Protocol?

1. Data representativeness
   - You want both your training set and test set to be representative of the data at hand
   - On the Pima Indian Diabetes dataset, what can happen if we don't shuffle the data?

4.2.2 How to Choose an Evaluation Protocol?

1. Data representativeness
   - You want both your training set and test set to be representative of the data at hand
   - On the Pima Indian Diabetes dataset, what can happen if we don't shuffle the data?
2. The arrow of time
   - If you're trying to predict the future given the past (for example, tomorrow's weather), you should not randomly shuffle your data before splitting it, because doing so will create a temporal leak:
     - Our model will effectively be trained on data from the future
     - In such situations, you should always make sure all data in your test set is **posterior** to the data in the training set

4.2.2 How to Choose an Evaluation Protocol?

1. Data representativeness
   - You want both your training set and test set to be representative of the data at hand
   - On the Pima Indian Diabetes dataset, what can happen if we don't shuffle the data?
2. The arrow of time
   - If you're trying to predict the future given the past (for example, tomorrow's weather), you should not randomly shuffle your data before splitting it, because doing so will create a temporal leak:
     - Our model will effectively be trained on data from the future
     - Make sure all data in your test set is **posterior** to the data in the training set (you cannot use future data in order to predict past events)
3. Redundancy in data
   - If some data points in your data appear twice (fairly common with real-world data), then shuffling the data and splitting it into a training set and a validation set will result in redundancy between the training and validation sets
   - In effect, you'll be testing on part of your training data!
   - Make sure your training set and validation set are disjoint

Temporal Leak Example

- Given the temperature of three previous time steps, we would like to predict the temperature at current time step.

Location	Feature 1	Feature 2	Feature 3	Label
A (train)	11 AM = 15	12 PM = 13	1 PM = 7	2 PM = ?
A (test)	9 AM = 10	10 AM = 14	11 AM = 15	12 PM = ?
A	10 AM = 14	11 AM = 15	12 PM = 13	1 PM = ?
B	11 AM = 10	12 PM = 9	1 PM = 7	2 PM = ?
B	9 AM = 15	10 AM = 11	11 AM = 10	12 PM = ?
B	10 AM = 11	11 AM = 10	12 PM = 9	1 PM = ?

How can information leak if we randomly split into training and test set?

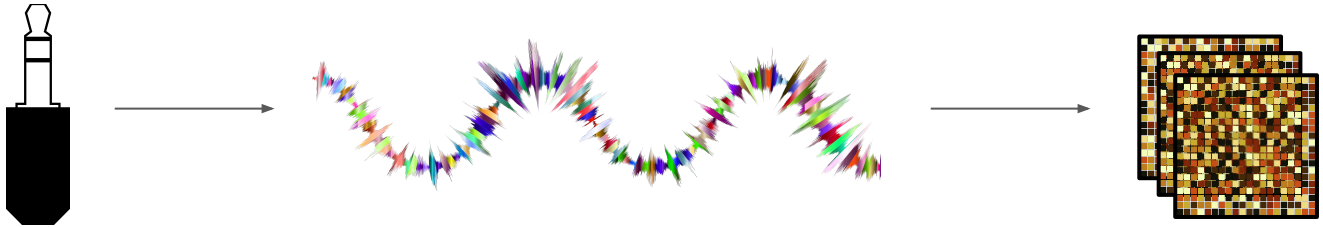
4.3 Data preprocessing, feature engineering, and feature learning

- How do you prepare the input data and targets before feeding them into a neural network?
  - For example, convert audio data to input features !
- Many data-preprocessing and feature-engineering techniques are domain specific (for example, specific to text data or image data)
  - But, there are some standard ones that are widely used across domains

4.3.1 Data Preprocessing for Neural Networks

Data Vectorization

- Turn the input data into tensors



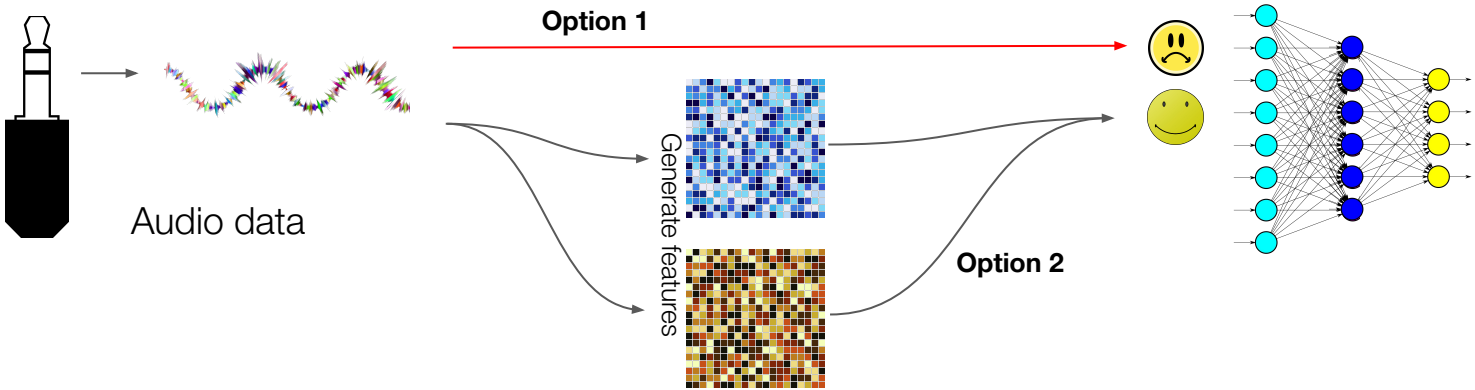
Value Normalization

- Heterogeneous data is unsafe for neural networks
  - for example, data where one feature is in the range 0–1 and another is in the range 100–200

```
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std

test_data -= mean
test_data /= std
```

4.3.2 Feature Engineering

- For a problem, you may know
  - (a) Some ‘things’ about the data that you believe the n/w cannot learn
  - (b) Some limitations of the n/w architecture that you design
- Feature engineering
  - The process of using your ‘own knowledge about the data’ and ‘about the machine-learning algorithm at hand’ to make the algorithm work better
- How do we do feature engineering?
  - hardcode (non-learned) transformations to the data before it goes into the model



4.3.2 Feature Engineering

- Problem: Build a deep learning model to “tell” time with clock image as input

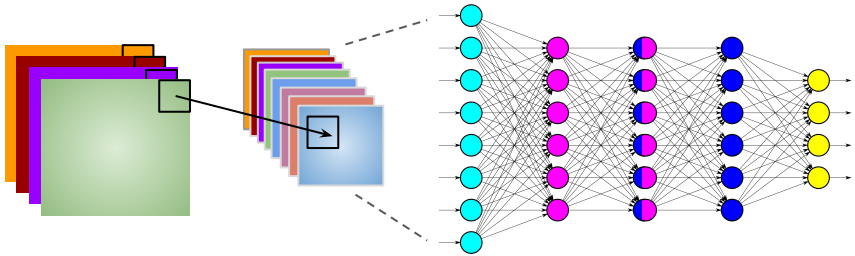
10:00 o’clock

4.3.2 Feature Engineering

- Problem: Build a deep learning model to “tell” time with clock image as input

10:00 o’clock

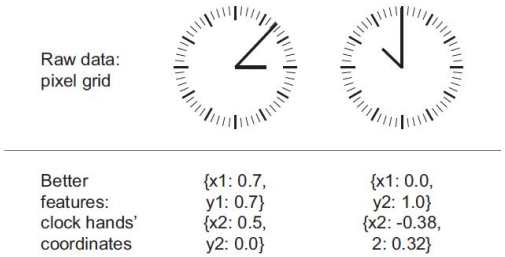
Will this work?



10:00 o’clock

4.3.2 Feature Engineering

- Feature engineering for reading the time on a clock
  - Write a Python script to follow the black pixels of the clock hands and output the (x, y) coordinates of the tip of each hand.



- Further:
  - Do a coordinate change, and express the (x, y) coordinates as polar coordinates with regard to the center of the image
  - Input = the angle theta of each clock hand
  - Is machine learning even required?
    - A dictionary lookup is enough to recover the approximate time of day.

4.3.2 Feature Engineering

- Feature engineering (usually) requires understanding the problem in depth
- Before deep learning, feature engineering used to be critical
  - Classical shallow algorithms don't have hypothesis spaces rich enough to learn useful features by themselves
  - The way you present the data to the algorithm was essential to its success
- MNIST Example:
  - Before CNNs became successful, hardcoded features were used:
    - the number of loops in a digit image
    - the height of each digit in an image
    - a histogram of pixel values

4.3.2 Feature Engineering

- Modern deep learning removes the need for most feature engineering
  - neural networks are capable of automatically extracting useful features from raw data
- But we still NEED feature engineering. Why? 

4.3.2 Feature Engineering

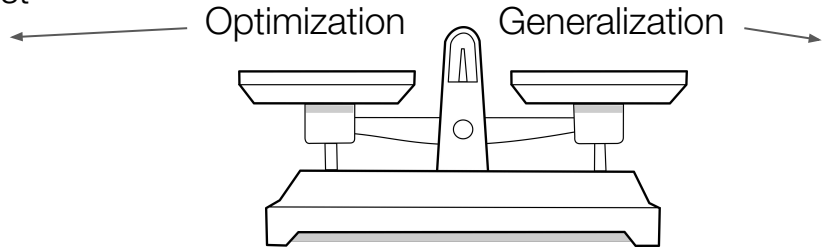
- Modern deep learning removes the need for most feature engineering
  - neural networks are capable of automatically extracting useful features from raw data
- But we still NEED feature engineering. Why? 

1. Good features let you solve a problem with far less data
   - Ability of deep learning models to learn features on their own relies on having lots of training data
   - If you have only a few samples, then the information value in their features becomes critical
2. Good features still allow you to solve problems more elegantly while using fewer resources
   - For instance, it would be ridiculous to solve the problem of reading a clock face using a convolutional neural network

4.4 Overfitting and Underfitting

- When we do the held-out validation data:
  - Performance on validation set peaks after a few epochs and then begins to degrade
- The model quickly starts to **overfit** the training data
- Learning how to deal with overfitting is essential to mastering machine learning
- The fundamental issue in machine learning is the tension between **optimization** and **generalization**
- The goal of the game is to get good generalization
  - but you don't control generalization; you can only adjust the model based on its training data

Adjusting a model to get the best performance possible on the training data!



How well the trained model performs on data it has never seen before!

4.4 Overfitting and Underfitting

Optimization: It has to look like a spear & fan & wall & **rope** & tree & snake & ...

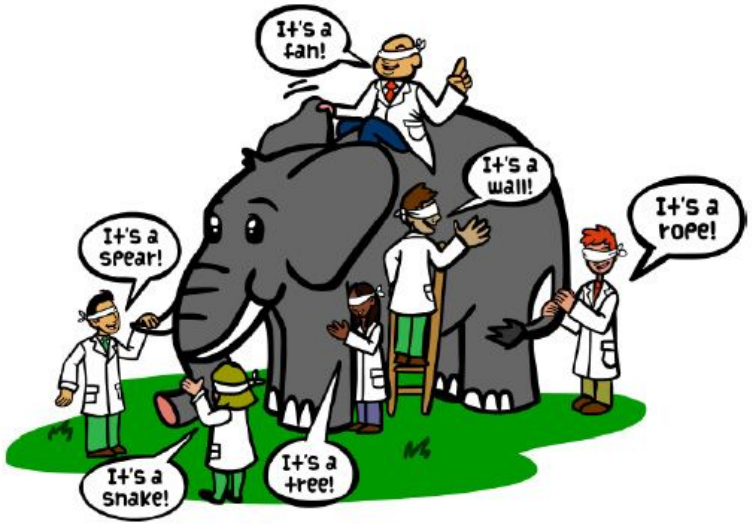
Generalization: “One side is snake like and the other side rope like” or “...”, etc.

To better optimize:

We need “more” blind people

To better generalize:

We need few but “smart” blind people

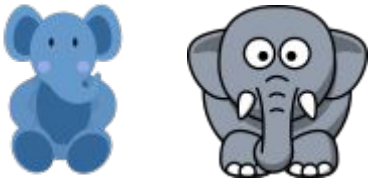


4.4 Overfitting and Underfitting

Training Data

Model

Test Data



What can we do to increase the model's ability to "generalize"?

- Increase the training data or decrease model's capacity?

4.4 Overfitting and Underfitting

We fight overfitting using "regularization" techniques:

1. Get more training dat
   - Check if augmentatic
2. Reducing network's s
3. Adding weight regula
4. Adding dropout layer

How to prevent overfitting.

Early stopping:

- Stop training as soon as the error on the validation set is higher than it was the last time it was checked.

Noise addition:

- Dropout: dropping out units (both hidden and visible) in a neural network.
- Add noise to data (e.g. denoising autoencoders): we train the network to reconstruct the input from a corrupted version of it.

Regularization penalties:

- Create weight penalties L1 and L2.

Dataset augmentation:

- Create fake data and add it to the training set.

4.4.1 Regularization by Reducing the Network's Size

- Model with more parameters = More memorization capacity
  - More capacity allows a model to easily learn a perfect dictionary-like mapping between 'training samples' and their 'targets'
    - a mapping 'without any generalization power'

4.4.1 Regularization by Reducing the Network's Size

- Model with more parameters = More memorization capacity
  - More capacity allows a model to easily learn a perfect dictionary-like mapping between 'training samples' and their 'targets'
    - a mapping 'without any generalization power'
- The challenge is generalization, not fitting
  - Deep learning models tend to be good at fitting to the training data

4.4.1 Regularization by Reducing the Network's Size

- Model with more parameters = More memorization capacity
  - More capacity allows a model to easily learn a perfect dictionary-like mapping between 'training samples' and their 'targets'
    - a mapping 'without any generalization power'
- The challenge is generalization, not fitting
  - Deep learning models tend to be good at fitting to the training data
- "Too much capacity vs not enough capacity"
  - If the network has limited memorization resources, it won't be able to learn this mapping as easily
  - The model will starve for memorization resources

4.4.1 Regularization by Reducing the Network's Size

- Model with more parameters = More memorization capacity
  - More capacity allows a model to easily learn a perfect dictionary-like mapping between 'training samples' and their 'targets'
    - a mapping 'without any generalization power'
- The challenge is generalization, not fitting
  - Deep learning models tend to be good at fitting to the training data
- "Too much capacity vs not enough capacity"
  - If the network has limited memorization resources, it won't be able to learn this mapping as easily
  - The model will starve for memorization resources
- So, what is the magical formula?
  - We must evaluate an array of different architectures (on your validation set, not on test set) in order to find the correct model size for your data
  - The general workflow to find an appropriate model size:
    - Start with relatively few layers and parameters, and increase the size of the layers or add new layers until you see diminishing returns with regard to validation loss

4.4.1 Regularization by Reducing the Network's Size

Listing 4.3 Original model

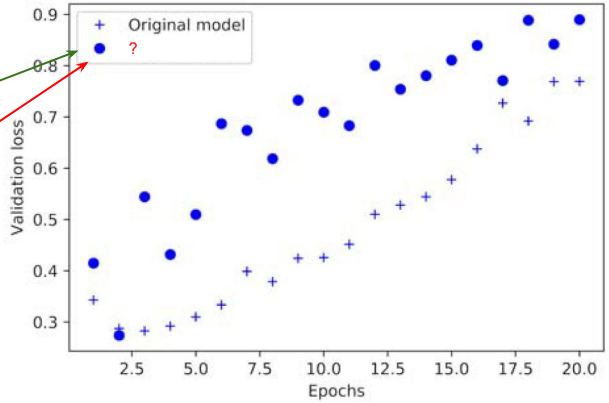
```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

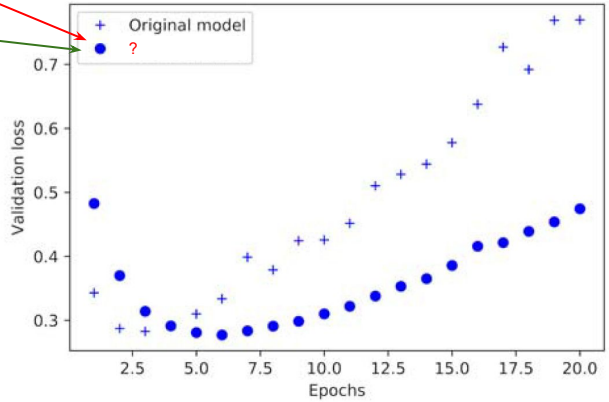
Listing 4.4 Version of the model with lower capacity

```
model = models.Sequential()
model.add(layers.Dense(4, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(4, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Listing 4.5 Version of the model with higher capacity

```
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```





Is red the correct mapping or green? Why?

Hints:

1) Which network nears zero loss more quickly?
2) Which network appears to have more 'capacity'?

4.4.2 Adding Weight Regularization

- Occam's razor: Given two explanations for something, the explanation most likely to be correct is the simplest one—the one that makes fewer assumptions
- Simpler models are less likely to overfit than complex ones
  - A simple model in this context is a model where the distribution of parameter values has less entropy (or a model with fewer parameters)
- How to reduce the complexity of a network (without changing the number of parameters?
  - Force its weights to take only small values
  - This will make the distribution of weight values more **regular**
- Weight regularization
  - Adding to the loss function of the network - "a cost associated with having large weights"
  - I.e. don't let the weight values grow

4.4.2 Adding Weight Regularization

Two ways to regularize weights: L_1 and L_2

L_1 regularization:

- The cost added is proportional to the absolute value of the weight coefficients

L_2 regularization

- The cost added is proportional to the square of the value of the weight coefficients
- L_2 regularization is also called weight decay in the context of neural networks

4.4.2 Adding Weight Regularization

Listing 4.6 Adding L2 weight regularization to the model

```
from keras import regularizers

model = models.Sequential()
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                       activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                       activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

`l2(0.001)` means every coefficient in the weight matrix of the layer will add `0.001 * weight_coefficient_value` to the total loss of the network.

4.4.2 Adding Weight Regularization

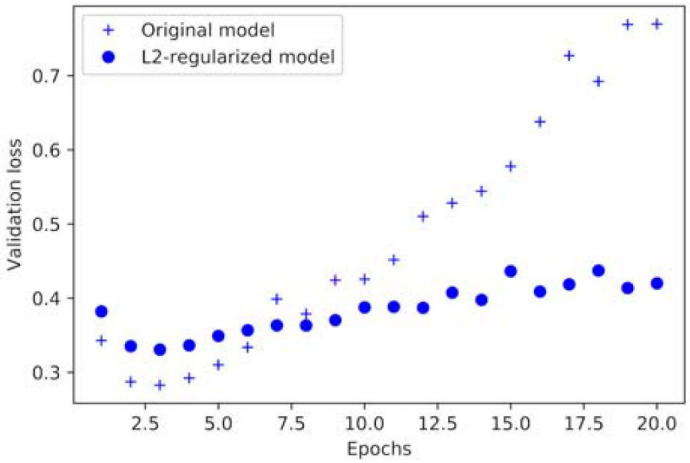


Figure 4.7 Effect of L2 weight regularization on validation loss

Listing 4.7 Different weight regularizers available in Keras

```
from keras import regularizers
regularizers.l1(0.001)
regularizers.l1_l2(l1=0.001, l2=0.001)
```

L1 regularization

Simultaneous L1 and L2 regularization

4.4.2 Adding Weight Regularization

DEMO

2019-Spring-DL / course-content / Module2 / **DL_L2_Regularization_IMDB_Movie_Reviews.ipynb**

badriadhikari Created using Colaboratory — 342af73 — 4 minutes ago

1 contributor

658 lines (658 sloc) | 172 KB — Raw | Blame | History

Open in Colab

Original tutorial at https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/3.5-classifying-movie-reviews.ipynb

In [1]:
```
import keras
from keras import models
from keras import layers
from keras import regularizers
import numpy as np
import matplotlib.pyplot as plt
from keras.callbacks import EarlyStopping
```

```
Using TensorFlow backend.
```

In [0]:
```
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

4.4.3 Adding Dropout

- Dropout is one of the most effective and most commonly used regularization
- Dropout, applied to a layer, consists of randomly dropping out (setting to zero) a number of output features of the layer during training
  - Let's say a given layer would normally return a vector `[0.2, 0.5, 1.3, 0.8, 1.1]` for a given input sample during training
  - After applying dropout, this vector will have a few zero entries distributed at random
    - For example, `[0, 0.5, 1.3, 0, 1.1]`

4.4.3 Adding Dropout

- Dropout is one of the most effective and most commonly used regularization
- Dropout, applied to a layer, consists of randomly dropping out (setting to zero) a number of output features of the layer during training
  - Let's say a given layer would normally return a vector `[0.2, 0.5, 1.3, 0.8, 1.1]` for a given input sample during training
  - After applying dropout, this vector will have a few zero entries distributed at random
    - For example, `[0, 0.5, 1.3, 0, 1.1]`
- The **dropout rate** is the fraction of the features that are zeroed out
  - It's usually set between 0.2 and 0.5

4.4.3 Adding Dropout

What about during testing?

- At test time, no units are dropped out (Strategy 1)
  - Instead, the layer's output values are scaled down by a factor equal to the dropout rate
  - `layer_output *= 0.5`
  - This is to balance for the fact that more units are active than at training time
- What will happen if we dropout during test?

4.4.3 Adding Dropout

What about during testing?

- At test time, no units are dropped out (Strategy 1)
  - Instead, the layer's output values are scaled down by a factor equal to the dropout rate
  - `layer_output *= 0.5`
  - This is to balance for the fact that more units are active than at training time
- What will happen if we dropout during test?

THINK PAIR SHARE

- Inverted dropout (Strategy 2)
  - `activation = activation / keep_probability` to scale up during the training

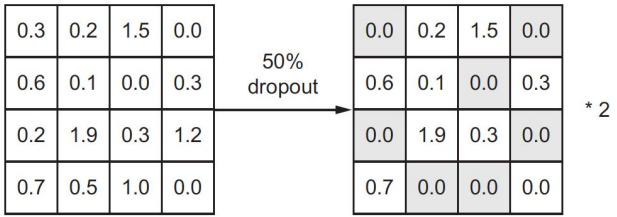


Figure 4.8 Dropout applied to an activation matrix at training time, with rescaling happening during training. At test time, the activation matrix is unchanged.

4.4.3 Adding Dropout

Why does dropout work?

- Hinton was inspired by a fraud-prevention mechanism used by banks
  - “I went to my bank. The tellers kept changing and I asked one of them why. He said he didn’t know but they got moved around a lot. I figured it must be because it would require cooperation between employees to successfully defraud the bank. This made me realize that randomly removing a different subset of neurons on each example would prevent conspiracies and thus reduce overfitting.”
- The core idea is that introducing noise in the output values of a layer can break up happenstance patterns that aren’t significant

4.4.3 Adding Dropout

Why does dropout work?

- Hinton was inspired by a fraud-prevention mechanism used by banks
  - “I went to my bank. The tellers kept changing and I asked one of them why. He said he didn’t know but they got moved around a lot. I figured it must be because it would require cooperation between employees to successfully defraud the bank. This made me realize that randomly removing a different subset of neurons on each example would prevent conspiracies and thus reduce overfitting.”
- The core idea is that introducing noise in the output values of a layer can break up happenstance patterns that aren’t significant

Listing 4.8 Adding dropout to the IMDB network

```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))
```

4.4.3 Adding Dropout

DEMO

2019-Spring-DL / course-content / Module2 / **DL_L2_Regularization_IMDB_Movie_Reviews.ipynb**

badriadhikari Created using Colaboratory — 342af73 — 4 minutes ago

1 contributor

658 lines (658 sloc) | 172 KB — Raw | Blame | History

Open in Colab

Original tutorial at https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/3.5-classifying-movie-reviews.ipynb

In [1]:
```
import keras
from keras import models
from keras import layers
from keras import regularizers
import numpy as np
import matplotlib.pyplot as plt
from keras.callbacks import EarlyStopping
```

```
Using TensorFlow backend.
```

In [0]:
```
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

4.5 The Universal Workflow of Machine Learning

Step 1: Define the problem and assemble a dataset

- Binary / multi-class classification or regression?

Step 2: Choose a measure of success

- Accuracy (for balanced), Precision / Recall (for unbalanced)

Step 3: Decide on an evaluation protocol

- Holdout / K-fold cross validation

Step 4: Prepare your data

- Normalization / standardization

Step 5: Develop a model that does better than a baseline

Step 6: Scale up: Develop a model that overfits

Step 7: Regularize your model and tune hyperparameters

4.5.5 Develop a Model That Does Better Than Baseline

- Can you achieve “statistical power”?
  - a small model that is capable of beating a dumb baseline
- Example 1:
  - In the MNIST digit-classification example, anything that achieves an accuracy greater than 0.1
- Example 2:
  - In the IMDB example, it’s anything with an accuracy greater than 0.5

4.5.5 Develop a Model That Does Better Than Baseline

- Can you achieve “statistical power”?
  - a small model that is capable of beating a dumb baseline
- Example 1:
  - In the MNIST digit-classification example, anything that achieves an accuracy greater than 0.1
- Example 2:
  - In the IMDB example, it’s anything with an accuracy greater than 0.5

- What can you do, if you cannot build a model with statistical power?
  - Check the last-layer’s activation - are you using sigmoid for regression?
  - Check the loss function - binary_crossentropy for classification and mse/mae for regression
  - Check optimizer - start with rmsprop with its default learning rate
  - Last resort: Use output as one of the input features

4.5.5 Develop a Model That Does Better Than Baseline

Table 4.1 Choosing the right last-layer activation and loss function for your model

Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	None	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

Example:
Classifying text into categories: One text could have multiple labels

4.5.6 Scale up: Develop a Model That Overfits

- So, we obtained a model that has statistical power, what next?
- Is our model sufficiently powerful to learn more patterns?
  - Does it have enough layers and parameters to properly model the problem at hand?

4.5.6 Scale up: Develop a Model That Overfits

- So, we obtained a model that has statistical power, what next?
- Is our model sufficiently powerful to learn more patterns?
  - Does it have enough layers and parameters to properly model the problem at hand?

- To figure out if you can overfit, you have to overfit
  - How to overfit?
    - Add layers, Make the layers bigger, and Train for more epochs
  - Monitor the training loss and validation loss (and accuracy/mae)
  - When you see that the model's performance on the validation data begins to degrade, you've achieved overfitting

4.5.7 Regularize Model & Tune Hyperparameters

- Repeatedly modify your model, train it, evaluate on your validation data
  - Repeat, until the model is as good as it can get
- What to change?
  - Add dropout
  - Try different architectures: add or remove layers
  - Add L1 and/or L2 regularization
  - Try different hyperparameters (such as the number of units per layer or the learning rate of the optimizer) to find the optimal configuration
  - [Optional] Add new features or remove features that don't seem to be informative

4.5.7 Regularize Model & Tune Hyperparameters

- Repeatedly modify your model, train it, evaluate on your validation data
  - Repeat, until the model is as good as it can get
- What to change?
  - Add dropout
  - Try different architectures: add or remove layers
  - Add L1 and/or L2 regularization
  - Try different hyperparameters (such as the number of units per layer or the learning rate of the optimizer) to find the optimal configuration
  - [Optional] Add new features or remove features that don't seem to be informative
- Once you've developed a satisfactory model configuration
  - Train your final production model on all the available data (training and validation) and evaluate it one last time on the test set