<https://github.com/fchollet/deep-learning-with-python-notebooks>  
C:\Users\is95217\PycharmProjects\source\keras\deep-learning-with-python-notebooks-master

# \*\* Part 1 Fundamentals of DL

# 1 What is Deep Learning

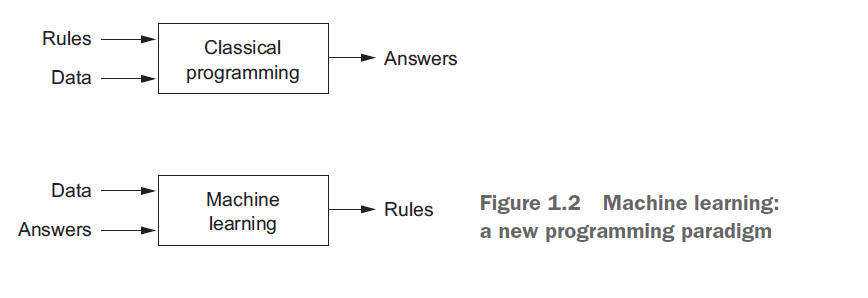
## AI, ML & DL

### AI

The effort to automate intellectual tasks normally performed by humans

### Machine Learning

A ML system is trained rather than explicitly programmed. It finds statistical structure of the examples given.

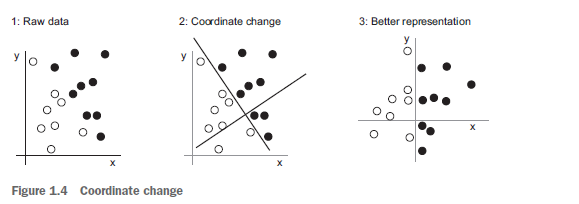


### Learning representations from data

To do machine learning :

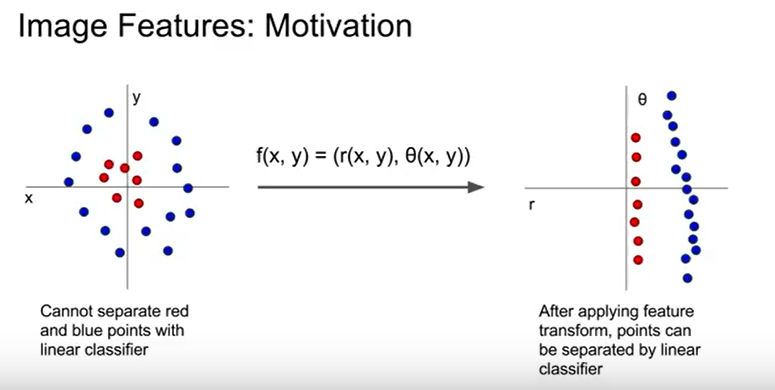
* Input data points
* Examples of expected output
* A way to measure whether the algorithm doing a good job

Central problem of ML is to learn useful represantations of the input data. Representations that get us closer to the expected output.



In fig3 Black points are x>0 and white points are x<0 ; this solves the classification problem

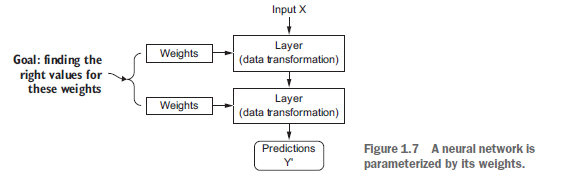
All machine-learning algorithms consist of automatically finding such transformations that turn data into more-useful representations for a given task.



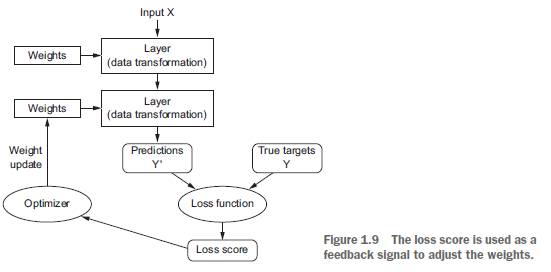
From: Stanford Lecture

### Deep Learning

* Successive layers of represantation. Layered representations learning, hierarchical rep learning
* What is representation?
  + Different way to look at data, to represent or encode data



Initially the weights of the network are assigned random values; its output will be far from the ideal first;



the weights are adjusted a little in the correct direction, and the loss score decreases.

## Before DL

### Probabilistic Modeling

Probabilistic modeling is the application of the principles of statistics to data analysis

Naive Bayes is a type of machine-learning classifier based on applying Bayes’ theoremwhile assuming that the features in the input data are all independent

### Early NN

* Multiple people independently rediscovered backpropogation , a way to train chains of parametric operations using gradient-descent optimization.
* In 1989 LeNet for handwritten digit classifiying.

### Kernel Methods

* A group of classification algorithms such as SVM(1990) It aims to find good decision boundaries between two sets.
* Trying to maximize the distance between hyperplane and the closest data points from each class.
* But SVMs proved hard to scale to large datasets and didn’t provide good results for

perceptual problems such as image classification

### Decision Trees

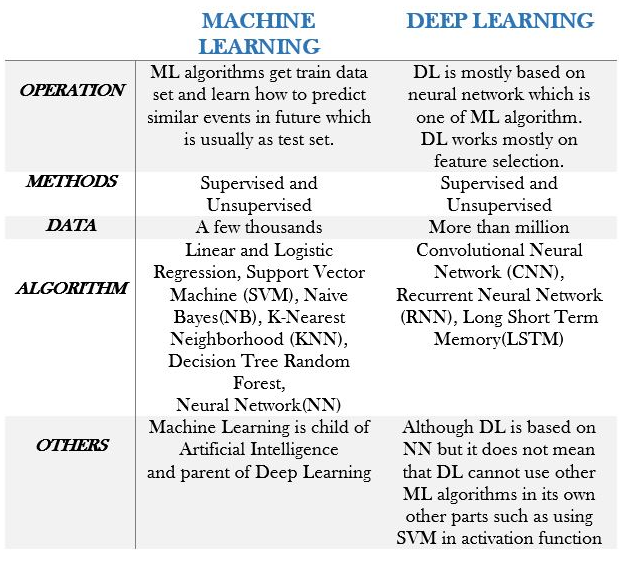
* Flowchart-like structures
* Random forest algorithm second best for any shallow machine learning task.
* Gradient boosting a way a way to improve any machine-learning model by iteratively training new models that specialize in addressing the weak points of the previous models
* Kaggle dominated by two approaches ;
  + Gradient boosting
  + Deep Learning

### Back to NN

* 2012 Imagenet computer vision accuracy from 74.3% to 83.6%, then to 96.4%
* It has completely replaced SVMs and decision trees in a wide range of applications.

### What makes DL different

* Better performance on many problems
* Problem solving much easier; because it automates feature engineering; you learn all features in one pass rather than having to engineering them yourself.



## Why DL? Why now?

### Hardware

In 2007 NVIDIA launched CUDA ( a API for GPU) . GPUs started to replace CPUs in highly paralellizable applications.

Deep neural networks, consisting mostly of many small matrix multiplications, are also highly parallelizable; and around 2011 some researchers began to write CUDA implementations of NN.

### Data

If deep learning is the steam engine of this revolution, then data is its coal:

the game changer has been the rise of the internet, making it feasible to collect and distribute very large datasets for machine learning

### Algorithms

In 2000s we were missing reliable way to train NN. As a result NN were still shallow with 1-2 layers.

This changed around 2009-2010

* Better activation functions for neural layers
* Better weight-initialization schemes, starting with layer-wise pretraining, which was quickly abandoned
* Better optimization schemes, such as RMSProp and Adam

### Others

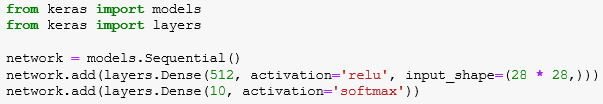
Investment increase, easy to use with Keras, Democratization of DL

# Mathematical building of NN

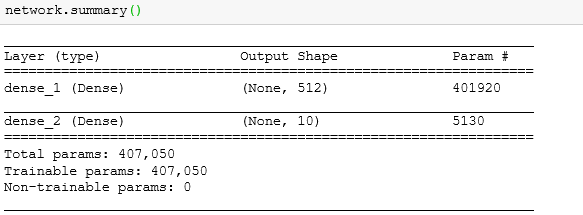
## First program

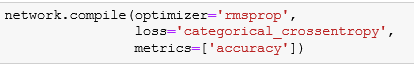
MNIST dataset in Keras

<https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/2.1-a-first-look-at-a-neural-network.ipynb>

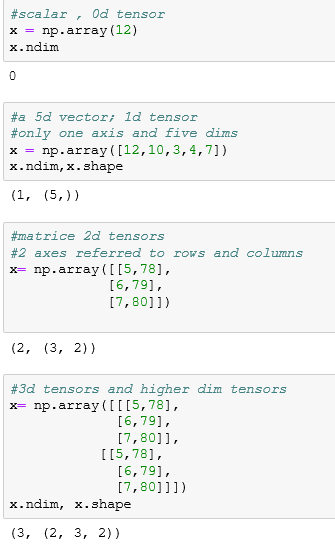


* 28\*28 image to vector 784 as input
* 784 ->(relu)-> 512 ->softmax-> 10
* 784+1 \* 512 = 401920 parameters(weights) to train in layer 1
* 512+1 \* 10 = 5130 params for layer 2

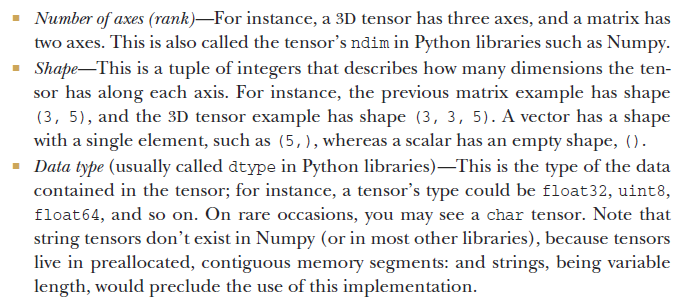




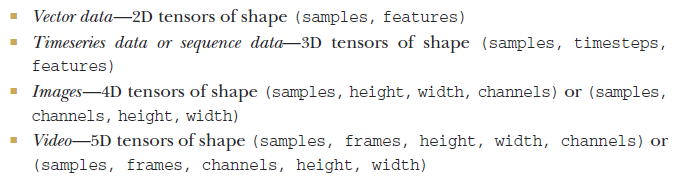
## Data representations for NN(tensor)



### Key attributes of tensors



### Real-world examples of data tensors



### Vector data(2D Tensor)

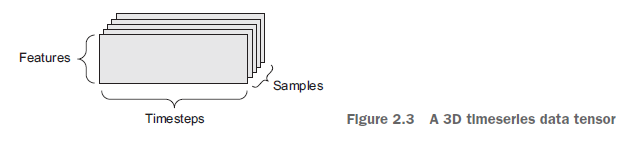
* Dataset of people, each person has age, zip code and income

2d tensor of shape(10000,3)

* Dataset of text documents where each doc has how many times each word appears in it

2d tensor of shape(500,20000)

### Timeseries data(3D Tensor)



* Dataset of stock prices, where every minute has current price,highest price and lowest price.
  + So each sample has 3D vector.
  + Entire day info is in 390,3 (there is 390 mins a work day)
  + 250 days of work so data will be stored (250,390,3) 3D tensor
* A dateset of 1m tweets can be stored (1m,280,128) where 128 is unique chars, 280 max char

### Image data(4D tensor)

A batch of 128 gray scale images is stored in a tensor of shape (128,256,256,1)

### Video data(5D tensor)

A 60 second 144x256 video sampled at 4 frames per second would have 60\*4=240 frames.

Tensor of shape ( 4,240,144,256,3)

## Tensor operations

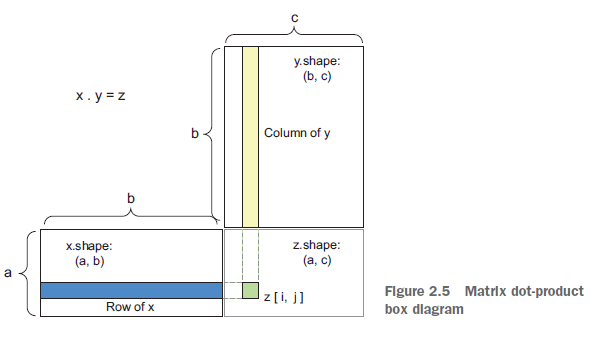
### Elementwise operations

The relu operation and addition are element-wise operations.

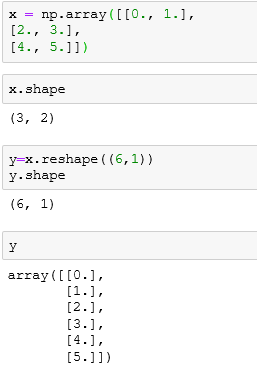
### Broadcasting

* What happens with addition when the shapes of the **two tensors being added differ?**
  + The smaller tensor will be broadcasted to match the shape of the larger tensor. Broadcasting consists of two steps:
    - Axes (called broadcast axes) are added to the smaller tensor to match the of ndim of the larger tensor.
    - The smaller tensor is repeated alongside these new axes to match the full shape of the larger tenso

### Tensor dot

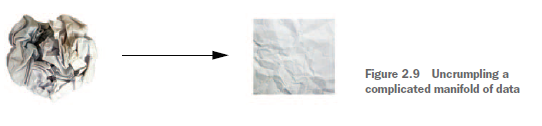


### Tensor reshaping



### Geometric interpretation of DL

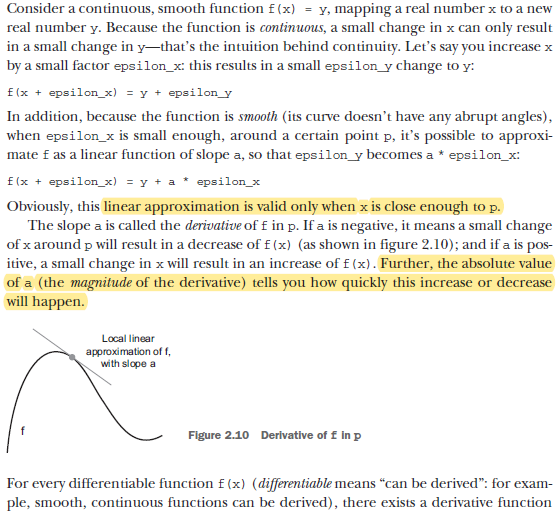
What a neural network (or any other machine-learning model) is meant to do is figure out a transformation of the paper ball that would uncrumple it, so as to make the two classes cleanly separable again



## Gradient\_based Optimization

* Each neural layer transforms its input data as follows
  + output = relu(dot(W, input) + b)
* W and b are tensors that are attributes of layer.(weights or trainable params)
* these weight matrices are filled with small random values (a step called randominitialization)
* What comes next is to gradually adjust these weights, based on a feedback signal. This gradual adjustment, also called training,

### What is derivative?



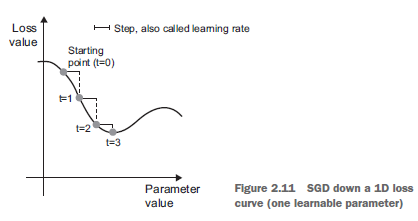
### Gradient- Derivative of tensor operation

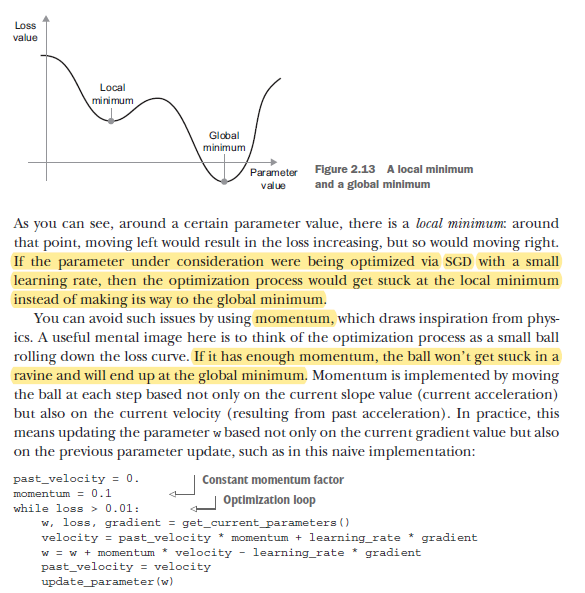
you can reduce f(W) by moving W in the opposite direction from the gradient:

for example, W1 = W0 - step \* gradient(f)(W0)

### Stochastistic Gradient Descent

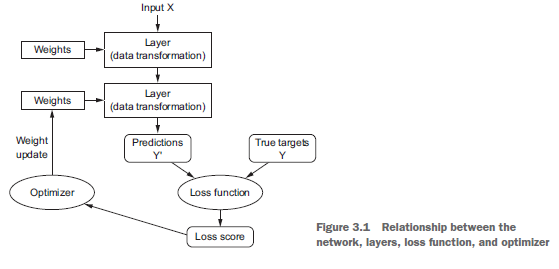
* The term stochastic refers to the fact that each batch of data is drawn at random
* combination of weight values that yields the smallest possible loss function





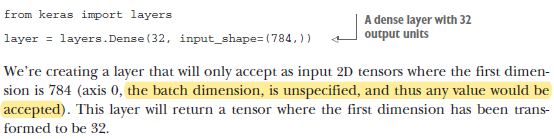
# Getting started with Neural Networks

## Anatomy of NN



### Layers

A layer is a data-processing module that takes as input one or more tensors and that outputs one or more tensors



The second layer didn’t receive an input shape argument—instead, it automatically inferred its input shape as being the output shape of the layer that came before

### Models

The most common instance is a linear stack of layers, mapping a single input to a single output.

Also there are Two-branch networks, Multihead networks, Inception blocks

### Loss functions and optimizers

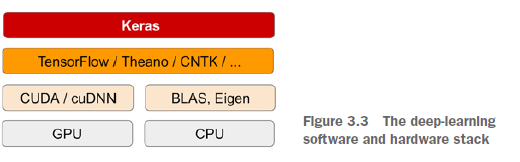
How to choose the correct loss?

* **Binary crossentropy** -> two class classification
* **Categorical crossentropy ->** many class classification problem
* **Meansquared error ->** regression problem
* **CTC ->** sequence learning problem

## 3.2 Keras

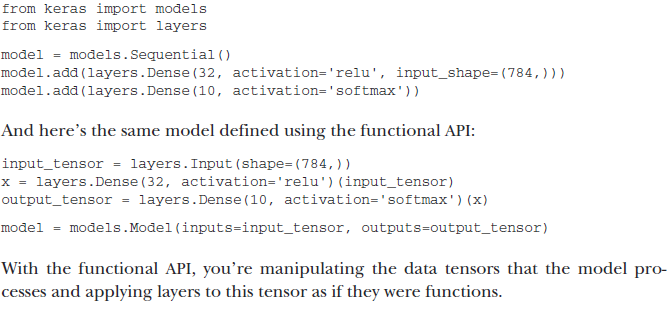
Model level library, providing high level building blocks for developing Deep Learning Models.

It doesnot handle low level operations such as tensor manipulation. Instead, it relies on a specialized, well-optimized tensor library to do so, serving as the backend engine of Keras



### Two ways to define a model

* Sequential class – only for linear stack of layers
* Functional API – for directed acyclic graphs of layer which lets you build completely arbitrary architectures.



-> 3.3 and 3.4 skipped

## 3.5 Binary and Multiclass classification

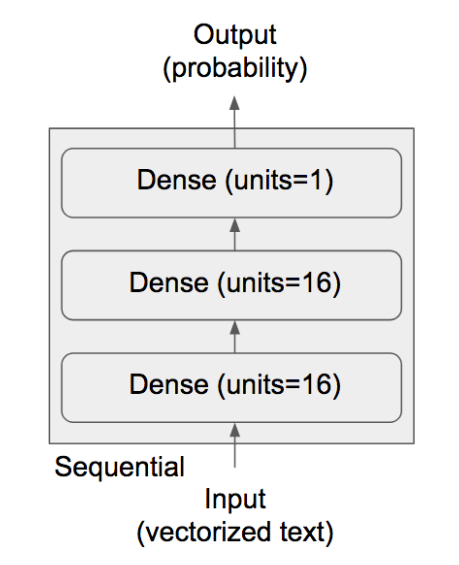
### Binary

IMDB review classification

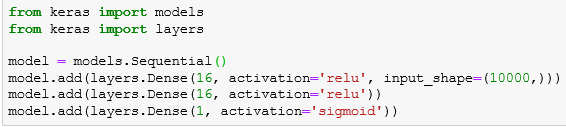
<http://localhost:8888/notebooks/source/keras/deep-learning-with-python-notebooks-master/3.5-classifying-movie-reviews.ipynb>

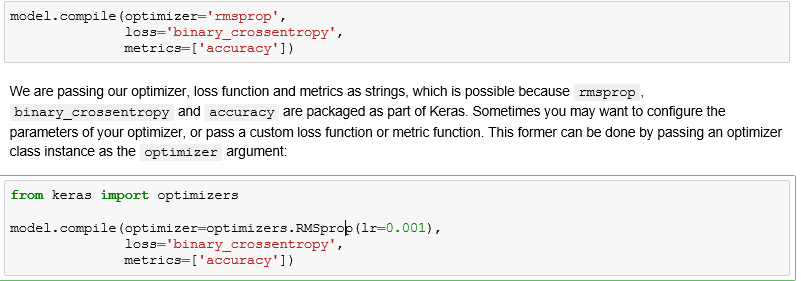
50K reviews from IMDB, 25K for training, 25k for testing. Each set has %50 positive and %50 negative

* Train\_data[0]= (218,) 🡺 To one hot vector x\_train[0]= (10000,)

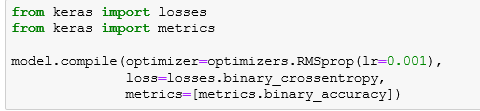


* Having 16 hidden units means that the weight matrix W will have shape (input\_dimension, 16),
* Having more hidden units (a higher-dimensional representation space) allows your network to learn more complex representations, but it makes your network more computationally expensive and may lead to learning unwanted patterns (patterns that will improve performance on the training data but not on the test data).



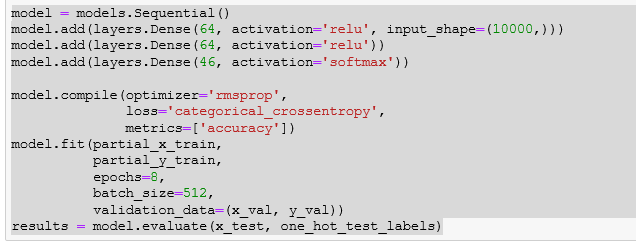


on a binary classification problem, the loss function you should use is binary\_crossentropy



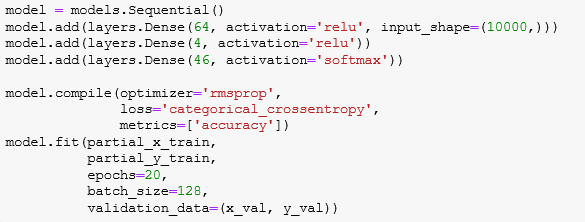
### Multiclass

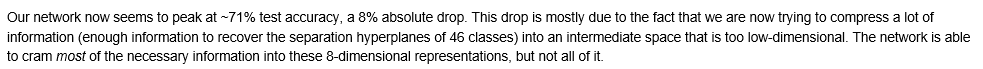
Newswire classification, softmax aand categorical\_crossentropy are to be used.



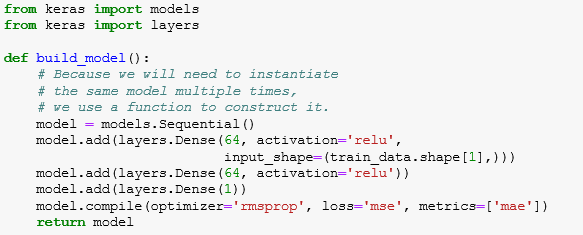
<http://localhost:8888/notebooks/source/keras/deep-learning-with-python-notebooks-master/3.6-classifying-newswires.ipynb>

With integer labels, we should use sparse\_categorical\_crossentropy:





## 3.6 Boston Housing Price



* Our network ends with a single unit, and no activation (i.e. it will be linear layer). This is a typical setup for scalar regression (i.e. regression where we are trying to predict a single continuous value). Applying an activation function would constrain the range that the output can take; for instance if we applied a sigmoid activation function to our last layer, the network could only learn to predict values between 0 and 1. Here, because the last layer is purely linear, the network is free to learn to predict values in any range.
* Note that we are compiling the network with the mse loss function -- Mean Squared Error, the square of the difference between the predictions and the targets, a widely used loss function for regression problems.
* We are also monitoring a new metric during training: mae. This stands for Mean Absolute Error. It is simply the absolute value of the difference between the predictions and the targets. For instance, a MAE of 0.5 on this problem would mean that our predictions are off by $500 on average.

# Fundementals of ML

## Four branches of ML

* Supervised
* Unsupervised
* Self-supervise
* Reinforcement

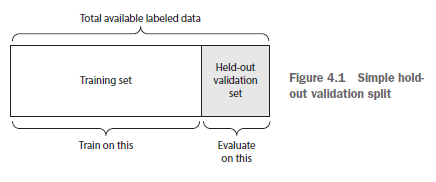
## Evaluating ML models

In machine learning, the goal is to achieve models that generalize

### Training, validation and test sets

* You may ask, why not have two sets: a training set and a test set?
  + The reason is that developing a model always involves tuning its configuration: for example, choosing the number of layers or the size of the layers (called the hyper parametersof the model, to distinguish them from the parameters, which are the net work’sweights).
  + You do this tuning by using as a feedback signal the performance of the model on the validation data. In essence, this tuning is a form of learning: a search for a good configuration in some parameter space. As a result, tuning the configuration of the model based on its performance on the validation set can quickly result in overfitting to the validation set, even though your model is never directly trained on it.
  + 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

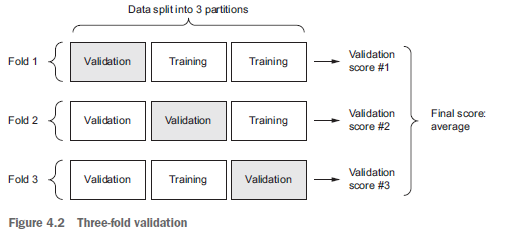
#### Simple hold-out validation



#### K-Fold Validation

you split your data into K partitions of equal size. For each partition I, train a model on the remaining K-1 partitions , evaluate it on partition i.

Your final score is then the avg of the K scores obtained



### Things to keep in mind

* **Data representativeness**

Randomly shuffle data into dev,test and train in classifying images of digit.

* **The arrow of time**

If you’re trying to predict the future given the past (for example, tomorrow’s weather, stock movements, and so on), you should not randomly shuffle your data before splitting it, because doing so will create a temporal leak:

* **Redundancy in your 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, which is the worst thing you can do! Make sure your training set and validation set are disjoint.

## Data preprocessing, feature engineering and learning

How do you prepare the input data and targets before feeding them into a neural network?

### Data Preprocessing

#### Vectorization

All inputs and targets in a neural network must be tensors of floating-point data (or, in specific cases, tensors of integers). Whatever data you need to process—sound, images, text—you must first turn into tensors, a step called data vectorization

#### Value normalization

it isn’t safe to feed data that has large values into a neural network ;

Doing so can trigger large gradient updates that will prevent the network from converging

**Take small values**—Typically, most values should be in the 0–1 range.

**Be homogenous**—That is, all features should take values in roughly the same range.

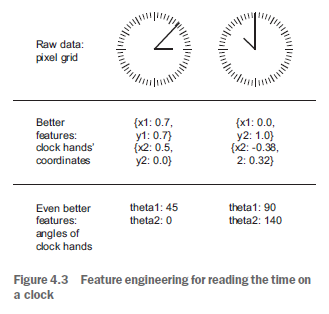
* Normalize each feature independently to have a mean of 0.
* Normalize each feature independently to have a standard deviation of 1.

#### Handling Missing Values

* With neural networks, it’s safe to input missing values as 0, 0 means missing data and will start ignoring the value.
* If there is missing values in test set, you should create artificially sample data for training.

### Feature Engineering

* Feature engineering is the process of using your own knowledge about the data and about the machine-learning algorithm at hand (in this case, a neural network) to make the algorithm work better by applying hardcoded (nonlearned) transformations to the data before it goes into the model.



* making a problem easier by expressing it in a simpler way.

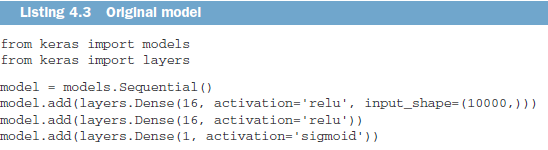
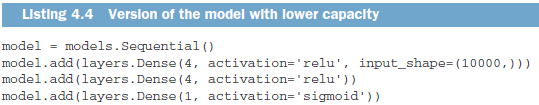
## Overfitting and Underfitting

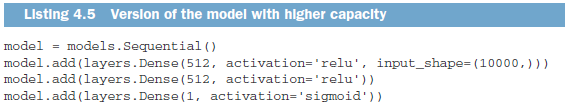
* Overfitting happens in every machine-learning problem
* The fundamental issue in machine learning is the tension between optimization and generalization.
* **Optimization** refers to the process of adjusting a model **to get the best performance** possible on the training data (the learning in machine learning), whereas **generalization** refers to how well the **trained model performs on data it has never seen before.**
* The goal of the game is to get good generalization, of course, but you don’t control generalization; you can only adjust the model based on its training data.
* the best solution is to get more training data
* When that isn’t possible, the next-best solution is to modulate the quantity of information that your model is allowed to store or to add constraints on what information it’s allowed to store
* The processing of fighting overfitting this way is called regularization

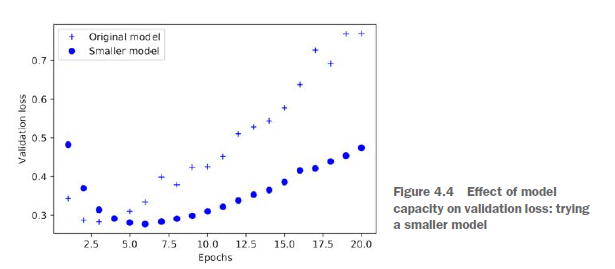
### Reducing the network’s size

to prevent overfitting:

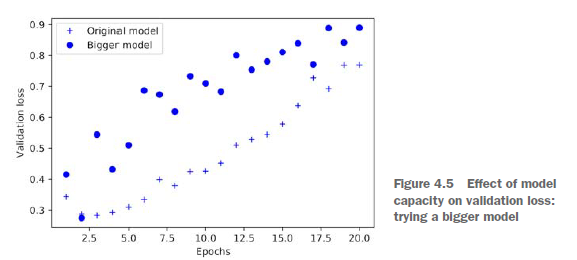
* reduce the size of the model
* the number of learnable parameters in the model (which is determined by the number of layers and the number of units per layer)
* Unfortunately, there is no magical formula to determine the right number of layersor the right size for each layer, size is to start with 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

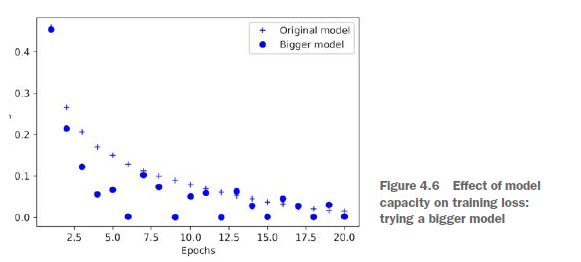




The smaller network starts overfitting later than the reference network



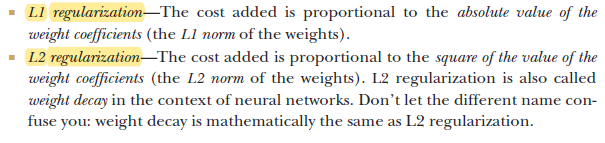
The more capacity the network has, the more quickly it can model the training data (resulting in a low training loss), but the more susceptible it is to overfitting (resulting in a large differencebetween the training and validation loss).

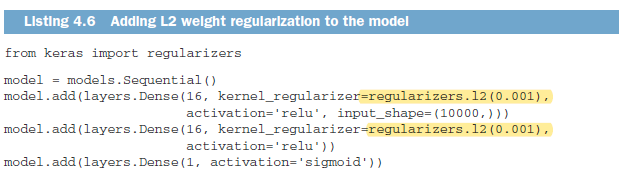


### Adding weight regularization

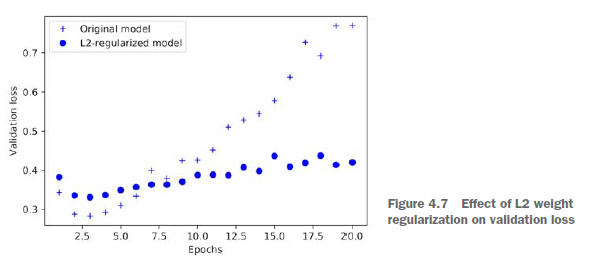
* Occam’s razor: the explanation most likely to be correct is the simplest one; the one that makes fewer assumptions.
* Simpler models are less likely to overfit han complex ones, simple models are with few parameters
* To mitigate overfitting is to put constraints on the complexity of a network by forcing its weights to take only small values(weight regularization)
* it’s done by adding to the loss function of the network a cost associated with having large

weights.





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

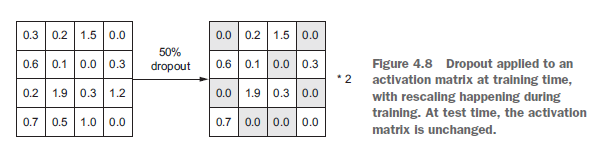


### Adding dropout

* (setting to zero) a number of output features of the layer during training

[0.2, 0.5, 1.3, 0.8, 1.1] 🡺 after dropout 🡺 [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.



* At test time, nounits are dropped out; At test time, we scale down the output by the dropout rate
* randomly removing a different subset of neurons on each example would prevent conspiracies

## Workflow of ML

### Defining Problem and assembling a dataset

* What will your input data be? What are you trying to predict?
* What type of problem are you facing? Is it binary classification? Multiclass classification? Scalar regression? Vector regression?
* You can’t move to the next stage until you know what your inputs and outputs are
  + You hypothesize that your outputs can be predicted given your inputs
  + You hypothesize that your available data is sufficiently informative to learn the

relationship between inputs and outputs.

* For nonstationary problems(model changes over time) retrain constantly or gather data at a time scale where the problem is stationary. Example: a recommendation engine for clothing is to be trained not only with a month data, but a few years data. Because it has to be suffice for seasonal change behaviour of customers.

### Choosing a measure of success

* For balanced-classification problems, where every class is equally likely, accuracy and area under the receiver operating characteristic curve (ROC AUC) are common metrics.
* For class-imbalanced problems, you can use precision and recall
* For ranking problems or multilabel classification, you can use mean average precision.
* Kaggle ; they showcase a wide range of problems and evaluation metrics

### Deciding on an evaluation protocol

* **A hold-out validation set**—The way to go when you have plenty of data
* Doing **K-fold cross-validation**—The right choice when you have too few samples for hold-out validation to be reliable
* Doing **iterated K-fold** validation—For performing highly accurate model evaluation when little data is available

Just pick one of these. In most cases, the first will work well enough.

### Preparing your data

* your data should be formatted as tensors
* tensors should usually be scaled to small values: for example, in the [-1, 1] range or [0, 1] range.
* If different features take values in different ranges (heterogeneous data), then the data should be normalized.
* You may want to do some feature engineering, especially for small-data problems.

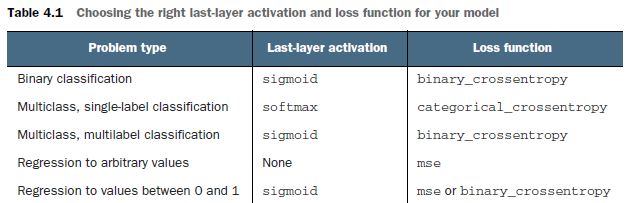
### Dev a model that does better than a baseline

* Aim is to achieve Statistical power.(develop a model capable of beating a dumb baseline)
* MNIST example accuracy gt 0.1, imdb example gt than 0.5 is OK.

it’s not always possible to achieve statistical power; in this case hypotheses are false and go back look to part 4.5.1 above where you design input output relation.

You need to make three key choices:

* Last layer activation
* Loss function
* Optimization configuration



### Scaling up: Developing a model that overfits

* Once you’ve obtained a model that has statistical power, the question becomes, is your model sufficiently powerful?
* Remember that the universal tension in machine learning is between optimization and generalization
* the ideal model is one that stands right at the border between underfitting and overfitting between undercapacity and overcapacity.

To figure out how big a model you’ll need, you must develop a model that overfits.This is fairly easy:

* Add layers.
* Make the layers bigger.
* Train for more epochs.

When validation data begins to degrade, you’ve achieved overfitting. The next stage is to start regularizing and tuning the model

### Regularizing model & tuning your hyperparameters

This step will take the most time: you’ll repeatedly modify your model, train it, evaluate on your validation data (not the test data, at this point), modify it again, and repeat, until the model is as good as it can get.

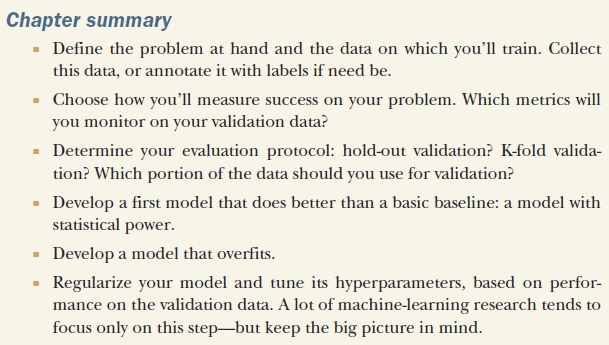
You should try:

* Add dropout.
* Try different architectures: add or remove layers.
* Add L1 and /or L2 regularization
* Try different hyperparameters
* Optionally, iterate on feature engineering: add new features, or remove featuresthat don’t seem to be informative

Be mindful of the following: every time you use feedback from your validation process to tune your model, you leak information about the validation process into the model

Once you’ve developed a satisfactory model configuration, you can train your finalproduction model **on all the available data** (training and validation) and evaluate it one last time on the test set

### Summary

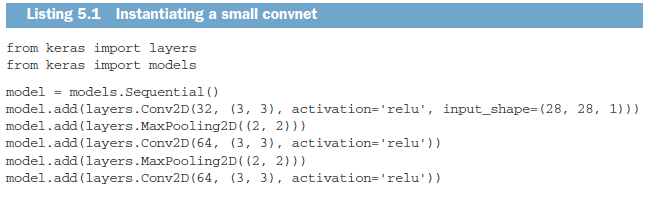


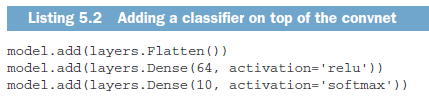
# \*\* Part 2 Deep Learning in practice

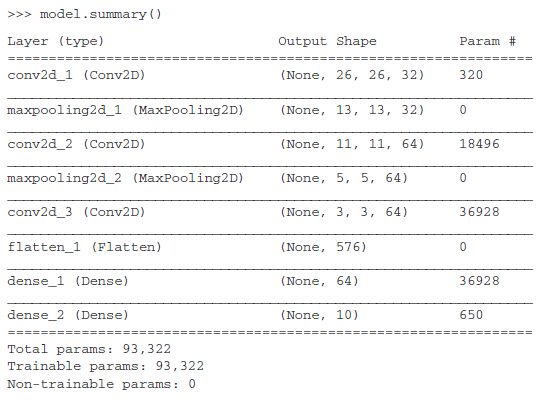
# Deep learning for computer vision

## Intro to Convnets

Basic convnet is better than densely connected network (Mnist example)







MNIST example:

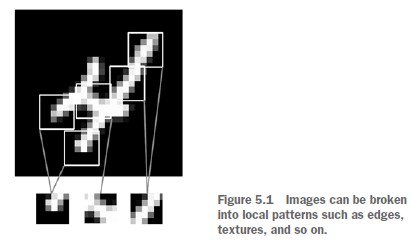
Densely connected network test accuracy : 97.8%

Convnet test accuracy :99.3%

### The convolution operation

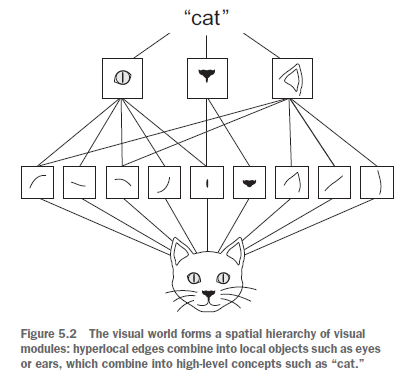
The fundamental difference between a densely connected layer and a convolution layer:

**Dense layers learn global patterns** in their input feature space,whereas **convolution layers learn local patterns**

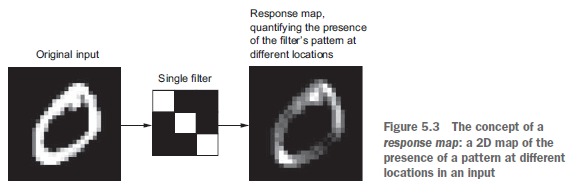


After learning a **certain pattern in the lower-right corner of a picture, a convnet can recognize it anywhere.**

A first convolution layer will learn small local patterns such as edges, a second convolution layer will learn larger patterns made of the features of the first layers, and so on



Convolutions operate over 3D tensors, called feature maps, with two spatial axes (height&width) as well as a depth axis (also called the channels axis).



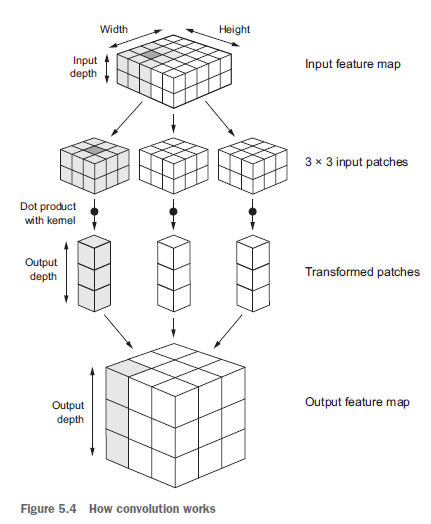
Convolutions are defined by two key parameters

* Size of the patches extracted from the inputs 3x3 or 5x5 typically
* Depth of the output feature map 32 and 64 in MNIST example

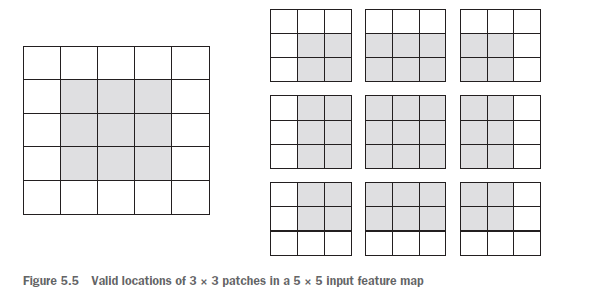
A convolution works by sliding these windows of size 3 × 3 or 5 × 5 over the 3D input feature map, stopping at every possible location, and extracting the 3D patch of surrounding features

Each such 3D patch is then transformed into a 1D vector of shape; All of

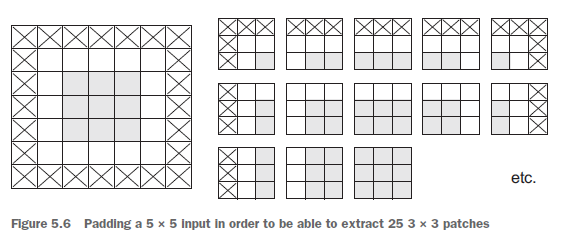
these vectors are then spatially reassembled into a 3D output map of shape



If you use 3x3 filter on 5x5 feature map, the output will be 3x3 ; shrink is inevitable



If you want to get an output feature map with the same spatial dimensions as the input, you can use padding

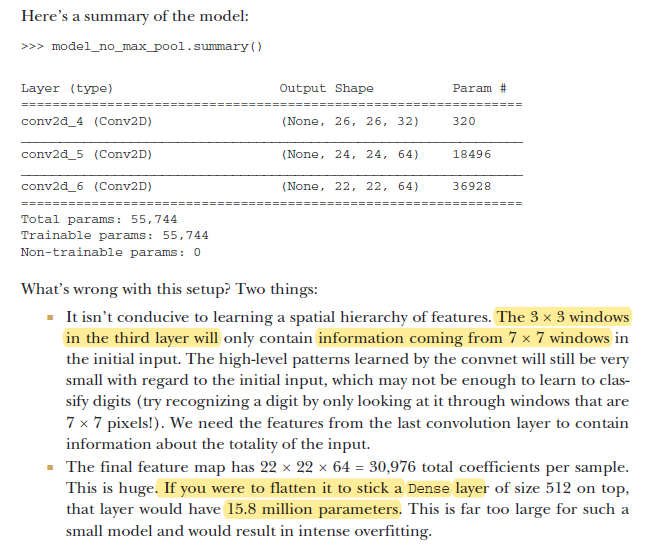


Using stride 2 means the width and height of the feature map are downsampled by a factor of 2,To downsample feature maps, instead of strides, we tend to use the max-pooling operation

### The max-pooling operation

Max pooling consists of extracting windows from the input feature maps and **outputtingthe max value of each channel**

instead of transforming local patches via a learned linear transformation (the convolutionkernel), they’re transformed via a hardcoded tensor operation. A big differencefrom convolution is that max pooling is usually done **with 2 × 2 windows** **and stride 2 in order to downsample the feature maps.**



But max pooling tends to work better than these alternative solutions,it’s more informative to look at the maximal presence of different features than at their average presence.

## Training a convnet from scratch on a small dataset

4000 pics of cats and dogs(2000 each) , 2000 for training 100 for val, 1000 for testing

* Convnet without any regularization accuracy: 71%
* With Data augmentation accuracy : 82%
* feature extraction with a pretrained network (90% -96%)
* With fine-tuning a pretrained network 97%

### The relevance of deep learning for small-data problems

It isn’t possible to train a convnet to solve a complex problem with just a few tens of samples;but a few hundred can potentially suffice if the model is small and well regularized and the task is simple

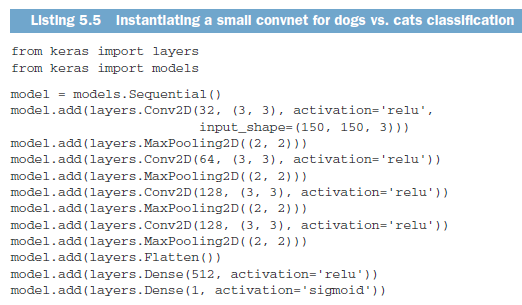
Training a convnet from scratch on a very small image dataset will still yield reasonable results despite a relative lack of data, without the need for any custom feature engineering

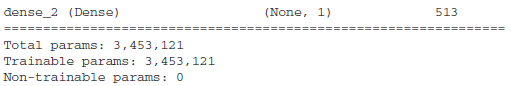
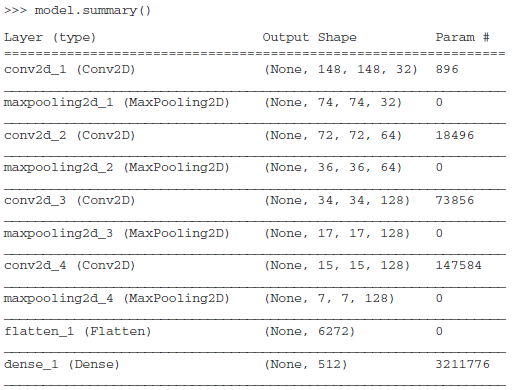
many pretrained models are now publicly available for download and can be used to bootstrap powerful vision models out of very little data.

### Downloading the data

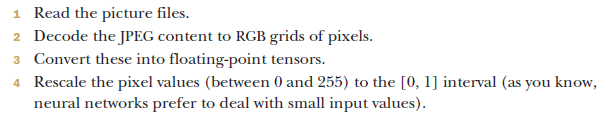
Kaggle.com/c/dogs-vs-cats/data

### Building your network

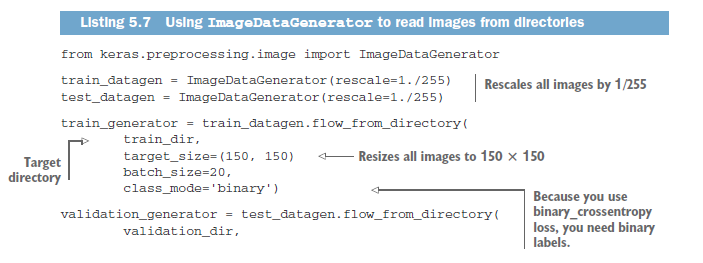




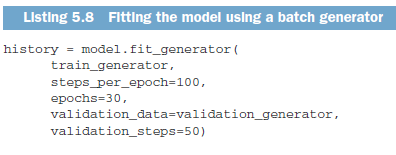
### Data preprocessing



In Keras, we use ImageDataGenerator which automatically turn image files into batches of preprocessed tensors.



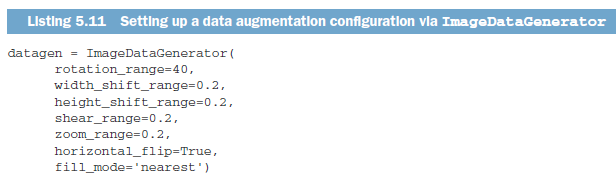
A Python generator is an object that acts as an iterator: it’s an object you can use with the for … in operator. Generators are built using the operator yield



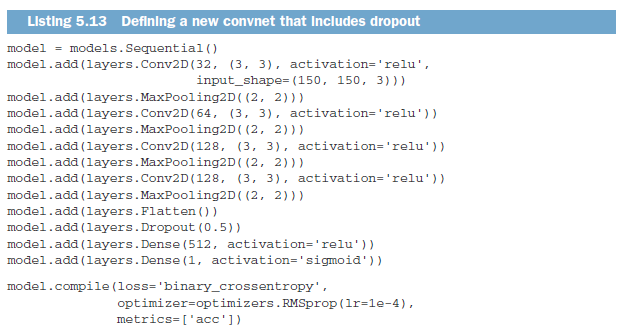
In this case, batches are 20 samples, so it will take 100 batches until you see your target of 2,000 samples

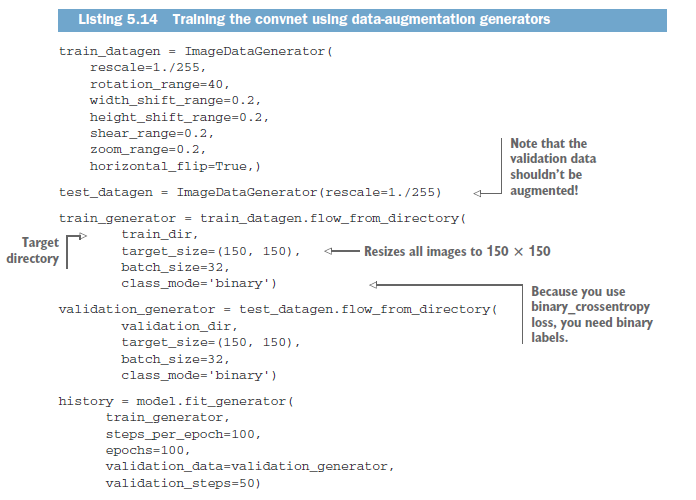
### Using data augmentation

Data augmentation takes the approach of generating more training data from existing training samples,



To further fight overfitting, you’ll also add a Dropout layer to your model, right before the densely connected classifier





Thanks to data augmentation and dropout, you’re no longer overfitting. Accuracy is now 82%.

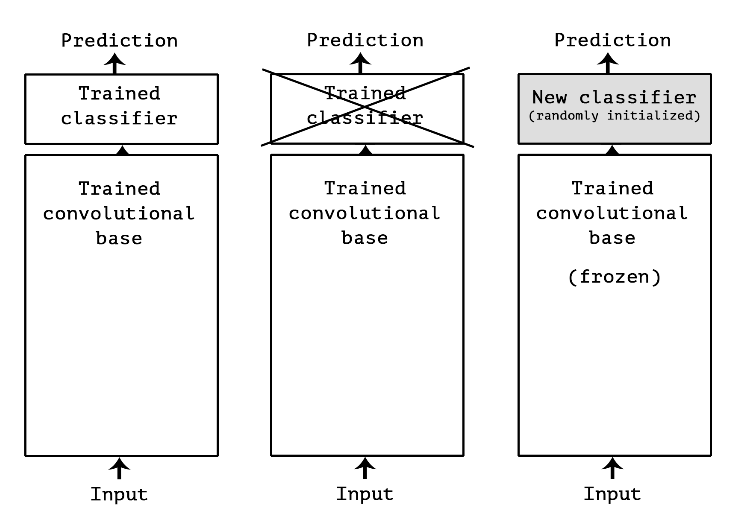
## Using a pretrained convnet

* A pretrained network is a saved network that was previously trained on a large dataset. pretrained network can effectively act as a generic model of the visual world, and hence its features can prove useful for many different computer vision problems.
* In this case, let’s consider a large convnet trained on the ImageNet dataset(1.4 million labeled images and 1,000 different classes). ImageNet contains many animal classes, including different species of cats and dogs, and you can thus expect to perform well on the dogs-versus-cats classification problem

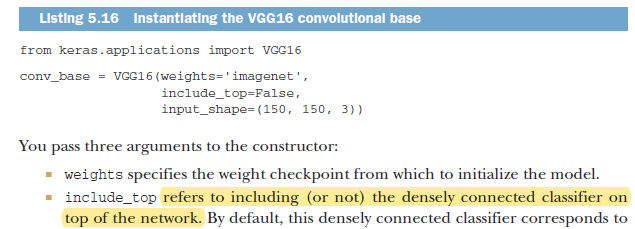
You’ll use the VGG16 architecture

### Feature extraction

* Feature extraction consists of using the representations learned by a previous network to extract interesting features from new samples. These features are then run through a new classifier, which is trained from scratch.



* Convnets used for image classification comprise two parts: they start with a series of pooling and convolution layers(conv base), and they end with a densely-connected classifier.
* Why only reuse the convolutional base? Could we reuse the densely-connected classifier as well?
  + the representations learned by the convolutional base are likely to be more generic and therefore more reusable: the feature maps of a **convnet are presence maps of generic concepts over a picture**
  + On the other end, the **representations learned by the classifier** will necessarily be **very specific to the set of classes** that the model was trained on -- they will only contain information about the presence probability of this or that class in the entire picture
  + representations found in **densely-connected layers** no longer contain any information about **where objects are located** in the input image
  + For problems where object location matters, densely-connected features would be largely useless



VGG16 takes input of 150x150x3 shape and produces output of 4x4x512 . That’s the feature on top of which you’ll stick a densely connected classifier

At this point, there are two ways you could proceed:

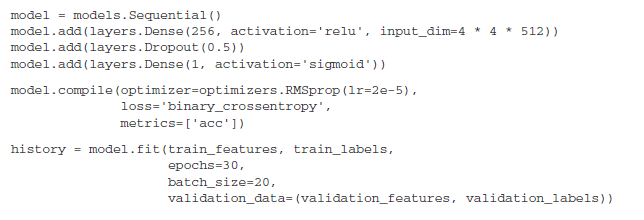
**1 FAST FEATURE EXTRACTION WITHOUT DATA AUGMENTATION**

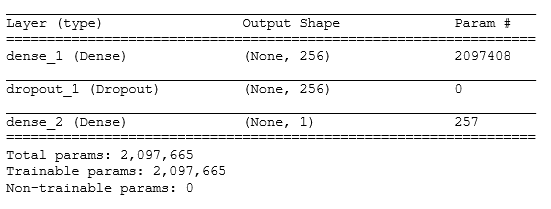
Running the convolutional base over your dataset, recording its output to a Numpy array on disk, and then **using this data as input to a standalone, densely connected** classifier similar to those you saw in part 1 of this book. it only requires running the convolutional base once for every input image, convolutional base is by far the most expensive part of the pipeline.

Input 150x150x3 🡺 **conv\_base(vgg16)** 🡺 output samplesx4x4x512

Running the convolutional base over our dataset, recording its output to a Numpy array on disk

This output will be feed to a densely connected classifer (samples, 8192)

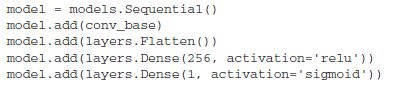


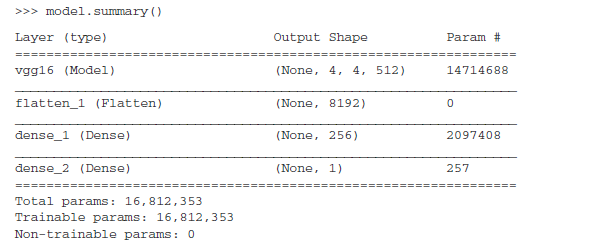


* batches are 20-sample large, so it will take 100 batches until we see our target of 2000 samples
* Training is very fast, because you only have to deal with two Dense layers
* You reach a validation accuracy of about 90%

**2 FAST FEATURE EXTRACTION WITH DATA AUGMENTATION**

* much slower and more expensive
* Extending the model you have(conv\_base) by adding dense layers on top, and running the whole thing end to end on the input data; this will allow to use augmentation, because every input image goes through the convolutional base every time it’s seen by the model.



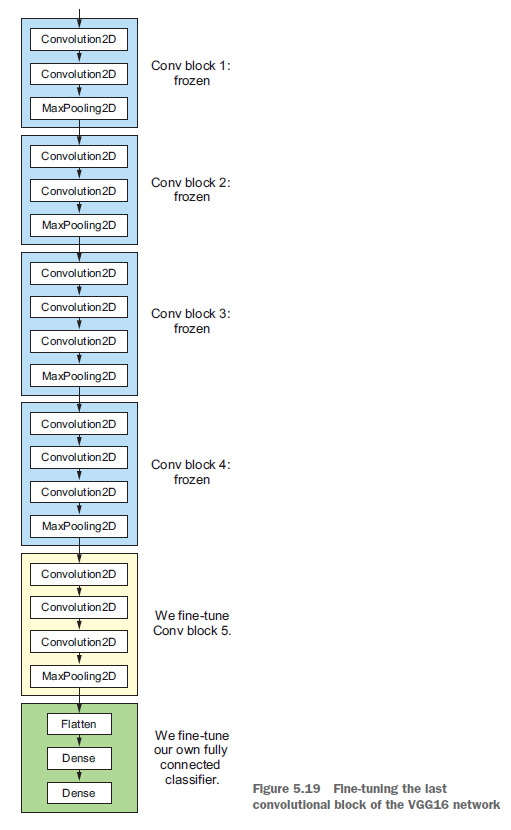


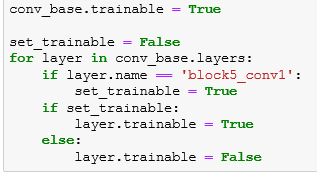
Freezing a layer or set of layers means preventing their weights from being updated during training

conv\_base.trainable = False

### Fine\_tuning

unfreezing a few of the top layers of a frozen model base used for feature extraction, jointly training both the newly added part of the model (in this case, the fully connected classifier) and these top layers. This is called "fine-tuning" because it slightly adjusts the more abstract representations of the model being reused, in order to make them more relevant for the problem at hand.





Why not fine-tune more layers? Why not fine-tune the entire convolutional base?

* Earlier layers in the convolutional base encode more-generic, reusable features, whereas layers higher up encode more-specialized features.
* The more parameters you’re training, the more you’re at risk of overfitting. The conv\_base has 15 million parameters, so it would be risky to attempt to train it on your small dataset.

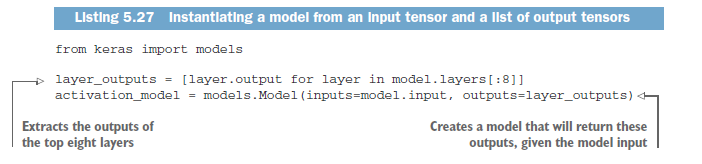
You’re seeing a nice 1% absolute improvement in accuracy, from about 96% to above 97%.

### Wrapping up

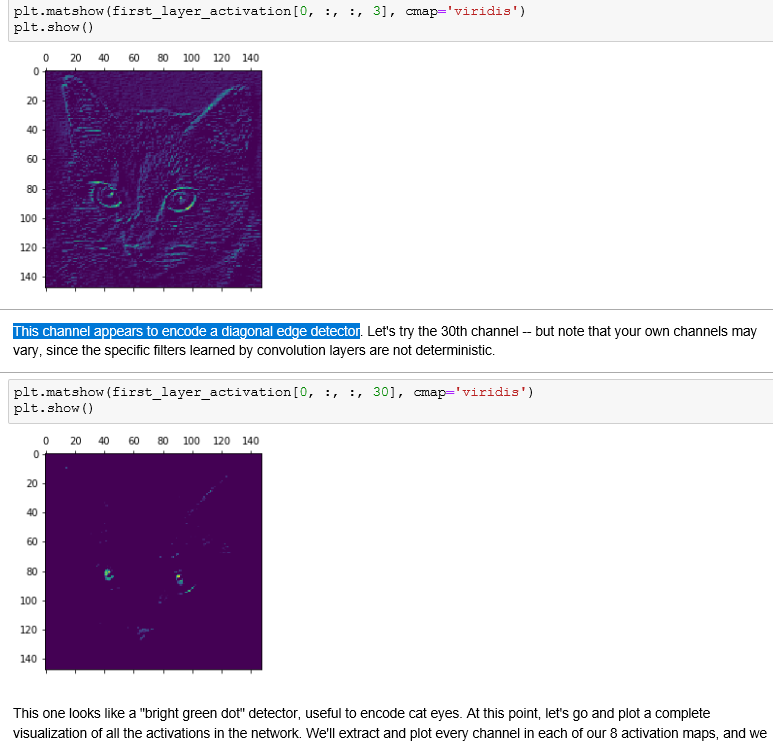
* Convnets are the best type of machine-learning models for computer-vision
* On a small dataset, overfitting will be the main issue. Data augmentation is a powerful way to fight overfitting when you’re working with image data.
* It’s easy to reuse an existing convnet on a new dataset via feature extraction. This is a valuable technique for working with small image datasets.
* As a complement to feature extraction, you can use fine-tuning, which adapts to a new problem some of the representations previously learned by an existing model. This pushes performance a bit further.

## Visualizing what convnets learn

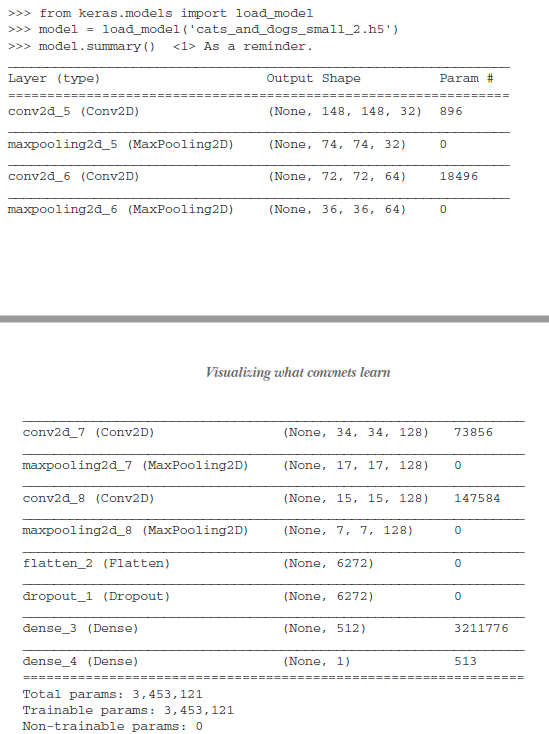
<http://localhost:8888/notebooks/source/deep-learning-with-python-notebooks/5.4-visualizing-what-convnets-learn.ipynb>



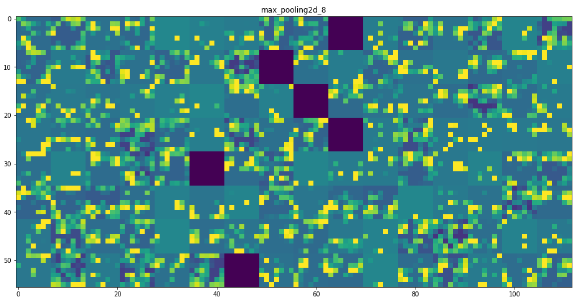
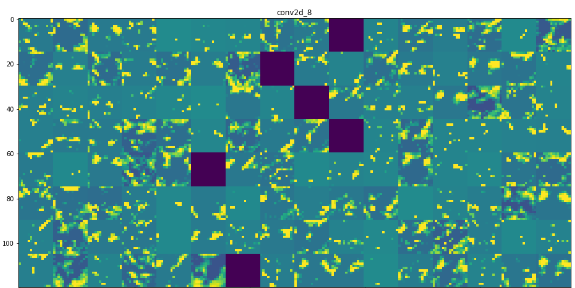
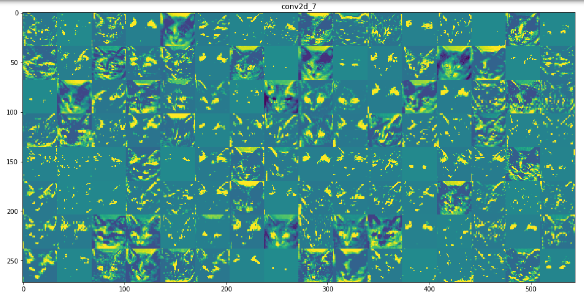
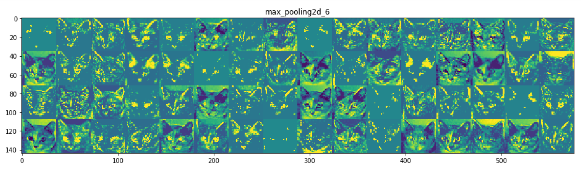
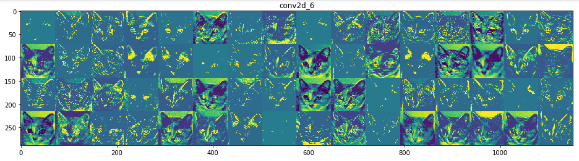
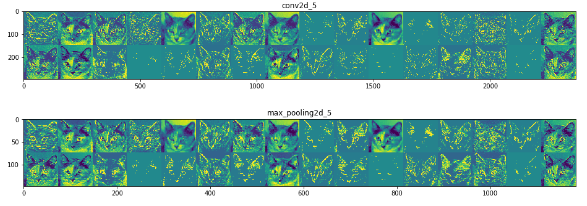
What sets the Model class apart is that it allows for models with multiple outputs, unlike Sequential.



### Cat&Dog model



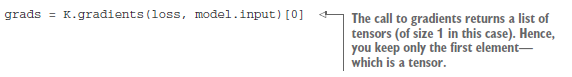
### Visualizing intermediate activations

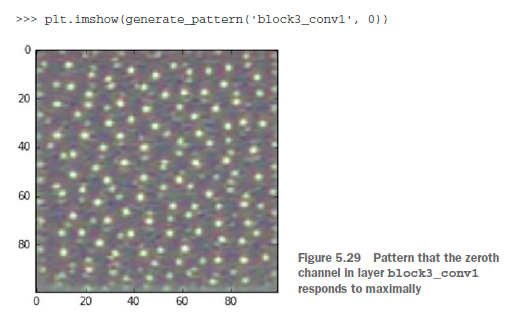


* **The first layer acts as a collection of various edge detectors**. At that stage, the activations are still retaining almost all of the information present in the initial picture
* As we go higher-up, the activations become increasingly abstract and less visually interpretable. They start encoding higher-level concepts such as "cat ear" or "cat eye". Higher-up presentations carry increasingly less information about the visual contents of the image, and increasingly more information related to the class of the image.
* The sparsity of the activations is increasing with the depth of the layer: in the first layer, all filters are activated by the input image, but in the following layers more and more **filters are blank. This means that the pattern encoded by the filter isn't found in the input image.**
* the features extracted by a layer get **increasingly abstract with the depth of the layer.** **The activations of layers higher-up carry less and less information about the specific input being seen, and more and more information about the target**
* What sets the Model class apart is that it allows for models with multiple outputs, unlike Sequential.

### Visualizing convnet filters

* applying gradient descent to the value of the input image of a convnet so as **to maximize the response of a specific filter**, starting from a blank input image
* you’ll build a loss function that maximizes the value of a given filter in a given convolution layer, and then you’ll use stochastic gradient descent to adjust the values of the input image so as to maximize this activation value.





* Filters from first layer encode simple directional edges and colors
* The filters from block2\_conv1 encode simple textures made from combinationsof edges and colors.
* The filters in higher layers begin to resemble textures found in natural images:feathers, eyes, leaves, and so on

# Deep learning for text and sequences

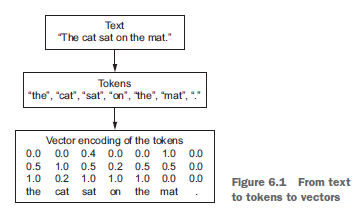
### Working with text data

Vectorizing text is the process of transforming text into numeric tensors. This can be done in multiple ways:

* Segment text into words and transform each word into a vector
* Segment text into characters and transform each char into a vector
* Extract n-grams of words or characters, and transform each n-gram into a vector.
  + N-grams are overlapping groups of multiple consecutive words or characters

The different units into which you can break down text (words, characters or n-grams) are called **tokens**. and breaking text into such tokens is called **tokenization**

* one-hot encoding of token
* token embedding



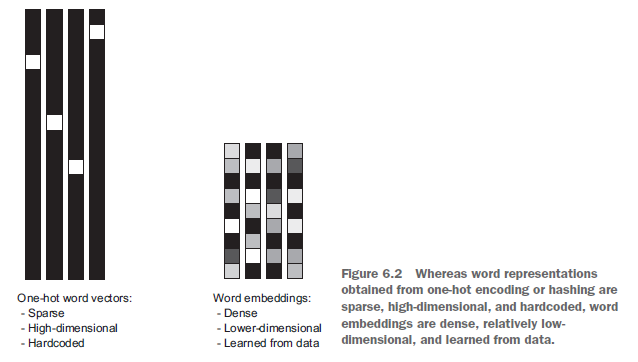
* Word n-grams are groups of N (or fewer) consecutive words that you can extract from a sentence.
* Example:
  + 2-grams:
    - {"The", "The cat", "cat", "cat sat", "sat","sat on", "on", "on the", "the", "the mat", "mat"}
  + 3-grams:
    - {"The", "The cat", "cat", "cat sat", "The cat sat","sat", "sat on", "on", "cat sat on", "on the", "the","sat on the", "the mat", "mat", "on the mat"}
* The term bag here refers to the fact that you’re dealing with a set of tokens rather than a list or sequence: the tokens have no specific order.
* Because bag-of-words isn’t an order-preserving tokenization method, it tends to be used in shallow language-processing models rather than in deep-learning models

#### One-hot encoding of words and characters

* Where **large sparse vectors were used to represent each word** or to score each word within a **vector to represent an entire vocabulary**.
* basic way to turn a token into a vector, the vector is all zeros except for the i th entry, which is 1.
* <http://localhost:8888/notebooks/source/DL%20Chollet/6.1-one-hot-encoding-of-words-or-characters.ipynb>  
  <https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/>
* the vectors obtained through one-hot encoding are **binary, sparse** (mostly made of zeros), and very **high-dimensional** (same dimensionality as the number of words in the vocabulary

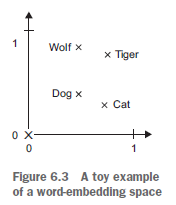
#### Using word embeddings

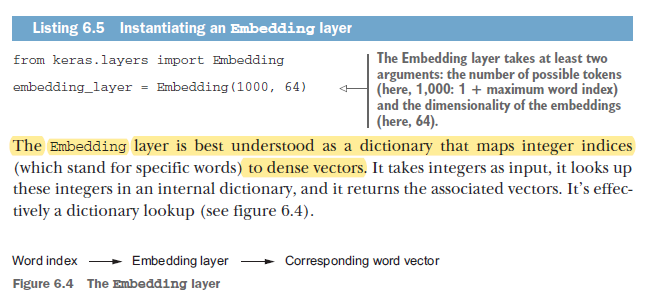
* Dense word vectors, also called word embeddings. Low dimensional
* It is an improvement over more the traditional bag-of-word model encoding schemes where large sparse vectors were used to represent each word or to score each word within a vector to represent an entire vocabulary.
* Word embeddings are learned from data.The position of a word within the vector space is learned from text and is based on the words that surround the word when it is used.
* The position of a word in the learned vector space is referred to as its embedding.
* 1,024-dimensional when dealing with very large vocabularies, whereas one hot is 20K dimensional
* word embeddings pack more information into far fewer dimensions.



* There are two ways to obtain word embeddings:
* **you start with random word vectors** and **then learn word vectors in the same way you learn the weights of a neural network.**
* Load into your model word embeddings that were **precomputed** using a different machine-learning task than the one you’re trying to solve. These are called **pretrained word embeddings.**

##### LEARNING EMBEDDING WORD EMBEDDINGS WITH THE LAYER

* Associate a dense vector with a word is to choose **the vector at random**, resulting embedding space has no structure: for example: **accurate and exact may end up completely different embeddings.**
* Geometric relationships between word vectors should reflect the semantic relationships between these words
* With the vector representations we chose here, some semantic relationships between these words can be encoded as geometric transformations
* 
* In real-world word-embedding spaces, common examples of meaningful geometric transformations are “gender” vectors and “plural” vectors
* **Keras Embedding Layer**



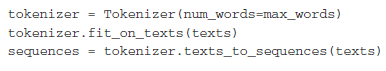
* All sequences in a batch must have the same length; if not will be padded with zeros.
* This layer returns a floating-point tensor of shape 3D (**samples, sequence\_\_length, embedding\_dimensionality)**
* When you instantiate an **Embedding layer**, **its weights** (its internal dictionary of token vectors) **are initially random**, just as with any other layer
* During training, these word vectors are gradually adjusted via backpropagation,
* **The Embedding layer is best understood as a dictionary mapping integer indices** (which stand for specific words) **to dense vectors.** It takes as input integers, it looks up these integers into an internal dictionary, and it returns the associated vectors. It's effectively a dictionary lookup**.**

##### USING PRETRAINED WORD EMBEDDINGS

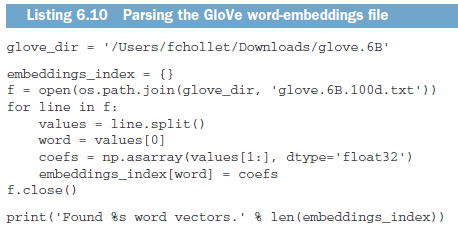
* you can load embedding vectors from a precomputed embedding space
* Such word embeddings are generally computed using word-occurrence statistics
* Two popular examples of methods of word embeddings from text include
  + Word2Vec.
  + GloVe. : factorizing a matrix of word co-occurrence statistics.

#### Putting it all together: from raw text to word embeddings

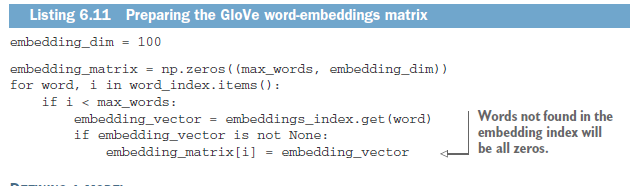
* Download the raw IMDB dataset
* Tokenize the data

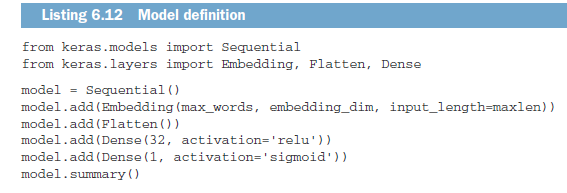


* Download the precomputed embeddings from 2014 English wiki(Glove.6B.zip)

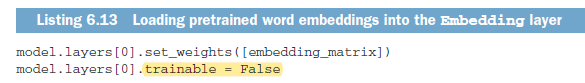


* Next, you’ll build an embedding matrix that you can load into an Embedding layer.





* pretrained parts shouldn’t be updated during training, to avoid forgetting what they already know



You can also train the same model without loading the pretrained word embeddings and without freezing the embedding layer. generally more powerful than pretrained word embeddings when lots of data is available

#### Wrapping up

### Understanding recurrent neural networks

#### A recurrent layer

#### LSTM and GRU

* The Long Short-Term Memory network, or LSTM network, is a recurrent neural network that is trained using Backpropagation Through Time and overcomes the vanishing gradient problem
* Instead of neurons, LSTM networks have memory blocks that are connected through layers.
* it saves information for later,thus preventing older signals from gradually vanishing during processing
* Conceptually, the carry dataflow is a way to modulate the next output and the next state
* There are three types of gates within a unit:
  + Forget Gate: conditionally decides what information to throw away from the block.
  + Input Gate: conditionally decides which values from the input to update the memory state.
  + Output Gate: conditionally decides what to output based on input and the memory of the block.
* The LSTM network expects the input data (X) to be provided with a specific array structure in the form of: **[samples, time steps, features].**

#### An example of LSTM

#### Wrapping up

### Advanced use of recurrent neural networks

### Sequence processing with convnets

### Summary

# Advanced deep-learning best practices

# Generative deep learning

# Conclusions