# Reading: [Artificial Intelligence](https://ezproxy.snhu.edu/login?url=https://doi.org/10.1017/9781108770422.026)

This reading from the Shapiro Library is a chapter from The Cambridge Handbook of Intelligence. It provides a brief overview of the history of artificial intelligence as well as describing different subfields of artificial intelligence. As you read, consider the following questions:

* What are the differences between engineering and psychological AI?
  + *Engineering AI* is concerned with how to design the smartest intelligent artifacts possible, regardless of whether the processes implemented reflect those found in people (or other animals). The vast majority of AI research on robotics and machine learning falls into this category. *Psychological AI*, in contrast, endeavors to design artifacts that think the way people do (or sometimes groups of people do). Much, but not all, research on cognitive systems belongs to this paradigm, though it is possible to design cognitive systems, such as Siri and Watson, that interact with humans but do not necessarily reason as people do.
* What is a production system?
* How does employing statistics affect logic?
* What is agent-based AI?

## **Summary**

Artificial intelligence (AI) is a scientific discipline that seeks to understand intelligence through the design and construction of intelligent machines. AI and cognitive science have a strong two-way relationship: Cognitive psychology often has inspired AI theories, and AI research has led to new theories of cognition that have been tested through psychological experimentation. While AI theories of cognition often are under-constrained, cognitive theories of AI tend to be over-constrained. Nevertheless, AI is useful for cognitive psychologists both as a source of new ideas and insights, and an experimental testbed. In this chapter, we describe some of the basic concepts and methods of AI by taking robot navigation in a city as an illustrative example. We also briefly discuss the history of AI, methods for assessing progress in AI, and some of AI’s potential impacts on society.

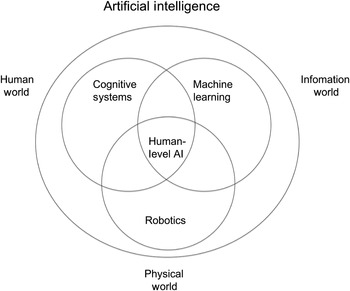
## 25 - Artificial Intelligence

two features unify all of AI as a discipline. First, AI is united in the core belief that intelligence is a kind of computation. Thus, although in principle the design of any intelligent artifact might be classified as an AI, in practice AI agents are almost always computers or computer programs, and AI laboratories typically are found in departments of computer science. Second, its main methodology is the exploration of the principles of intelligence by building computational artifacts.

Broadly speaking, AI includes three large subfields: *robotics, machine learning*, and *cognitive systems*.

* Robotics deals with embodied AI agents that interact with the physical world.
* Machine learning typically pertains to computer programs that can detect and exploit patterns in data.
* Cognitive systems, such as game-playing programs (for example, DeepBlue and AlphaGo) and conversational agents (for example, Siri and Watson) pertain to higher-level cognition, and interact with human and social worlds.

### Figure 25.1 The field of artificial intelligence includes the subfields of robotics, machine learning, and cognitive systems.



## AI and Cognitive Science

AI models of cognition differ from other models in psychology in that AI models always implement *information-processing* theories. That is, the theory describes intelligence in terms of the content, representation, access, use, and acquisition of information, as opposed to, say, a statistical model of the influences of age or nutrition on IQ in a population.

AI contributes to the understanding of intelligence in several ways

* First, although they can be under constrained, *AI programs demonstrate what kinds of data need to be collected*.
* Another thing AI is particularly good at is *exploring the benefits and limitations of various ways to represent and organize knowledge in memory*.
* Finally, once there is an AI program that resembles some part of human thinking to a researcher’s satisfaction, *it is possible to run experiments on the program that are either unethical or too expensive (in terms of time or money) to run on living beings*.

In both cognitive psychology and AI cognitive systems, researchers over the years have tried to build theories of intelligence of two different kinds at two different levels of abstraction: the symbolic and the sub-symbolic.

* Symbols represent conceptual abstractions of the world, such as *dog* or *justice*, and act as pointers both to the world and to one another; thus, symbolic processing pertains to conceptual information processing.
  + Symbolic cognitive systems may organize symbols into propositions that can be used in reasoning processes, such as deduction, induction, and various transformations.
* In contrast, sub-symbolic processing tends to deal with information at a finer grain, such as the pixels of an image, the weight of an association, and the probability of the truth of a proposition.
  + Connectionism is the dominant sub-symbolic modeling paradigm in cognitive science, and works by applying neural networks to psychological problems.

## Navigational Planning: An Illustrative Example

We want to illustrate a simple example of AI in some detail to help make this discussion more concrete. Let us suppose that Sunny, a cheerful AI agent, is about to start a new job in a new city. Sunny starts its car at its apartment and needs to navigate to an office building downtown. How might Sunny think and what might Sunny do, given that this is its first day in the city and it has never been to the office building? Our goals in this section are to explain some dimensions in designing AI agents as well as describe some issues in putting multiple capabilities into an AI agent.

### Action, Perception, and Cognition

To reach its office from its apartment, Sunny might use one (or more) of several possible strategies.

* Go a short distance in a random direction, see if it has reached its destination, then go another short distance in a random direction and check again.
  + takes a lot of time but is very efficient in terms of internal processing
  + This *perceive-act* internal computational processing, called *situated action* (or *reactive control*; Arkin, [1999](https://www-cambridge-org.ezproxy.snhu.edu/core/books/cambridge-handbook-of-intelligence/artificial-intelligence/B994B0D29512087BF53979CA9EABC9AB#REFe-r-25_005)), works by perceiving the immediate environment, acting based on those perceptions, and then repeating.
* Ask a long-time resident how to get to the office building and get directions from them
  + In contrast to the previous strategy, this strategy produces very efficient output behavior: Assuming that Honey’s directions are good, Sunny should reach its goal quite efficiently. However, this strategy of *asking* requires a society of intelligent agents (human or AI), each with different knowledge. It also requires a culture in which Sunny may in fact approach Honey for directions; Honey might in fact stop to help Sunny, and the two can communicate in a shared language; Sunny might trust Honey, a total stranger, enough to follow its directions in a new city; and so on. AI research on robot societies and human-robot interaction is in its early stages, and so here we will briefly mention only a small set of selected issues.
* utilize pregiven instructions from the factory, like if trying to go to downtown of a city, then go to the tallest buildings
  + This knowledge directly uses the goal (reaching downtown) to suggest a high-level action (move in the direction of the tallest buildings) and is heuristic in its nature since it may not correctly apply in all cities.
  + This strategy of *perceive-think-act* not only requires some knowledge but also must use more complex internal processing than the simpler perceive-act strategy of situated action.
* consult a map of the new city

### Reasoning, Learning, and Memory

* Find route using map for first day, then next day retrieve that information from saved case in memory to reuse.
  + This is called *case-based reasoning* (Kolodner, [1993](https://www-cambridge-org.ezproxy.snhu.edu/core/books/cambridge-handbook-of-intelligence/artificial-intelligence/B994B0D29512087BF53979CA9EABC9AB#REFe-r-25_022)). This approach views reasoning largely as a memory task, that is, as a task of retrieving and modifying almost correct solutions from memory to address the current problem. Related subdisciplines of cognitive science studying similar phenomena are exemplar-based reasoning, memory-based reasoning, instance-based reasoning, and analogical reasoning.
* Let us suppose, for example, that Sunny uses the strategy of situated action for action selection. It might, for example, use a table (called a *policy*) that specifies mappings from percepts of the world into actions on it. Then, from the feedback, or the reward, on a series of actions, Sunny can learn updates to the policy so that over time its action selection is closer to optimal. This is called *reinforcement learning* (Sutton & Barto, [1998](https://www-cambridge-org.ezproxy.snhu.edu/core/books/cambridge-handbook-of-intelligence/artificial-intelligence/B994B0D29512087BF53979CA9EABC9AB#REFe-r-25_065)). Note that if the series of actions results in success, then the reward will be positive; otherwise it is negative. Reinforcement learning is an especially useful learning strategy when the reward is delayed, that is, it comes after a sequence of actions rather than immediately after an action so that it is not clear what specific action in the sequence was responsible for the success or failure.
* Alternatively, suppose that Sunny employs the strategy of using production rules such as “If x then do y” to select actions. In this case, Sunny can use the learning strategy of *chunking* (Laird et al., [1987](https://www-cambridge-org.ezproxy.snhu.edu/core/books/cambridge-handbook-of-intelligence/artificial-intelligence/B994B0D29512087BF53979CA9EABC9AB#REFe-r-25_028)) to learn new rules from its experiences over time. Thus, just as AI has developed many reasoning strategies for action selection, it has developed many learning strategies for acquiring the knowledge needed by the reasoning strategies. Further, just like the reasoning strategies, the learning strategies too offer trade-offs among knowledge requirements, computational efficiency, and solution quality.

### Deliberation and Situated Action

the design of Sunny, our friendly robot, might contain a deliberative planner that generates plans to navigate from one location in a city to another.

### Deliberation and Reflection

Recent AI research on meta-reasoning is starting to design AI agents capable of self-adaptation (Cox & Raja, [2011](https://www-cambridge-org.ezproxy.snhu.edu/core/books/cambridge-handbook-of-intelligence/artificial-intelligence/B994B0D29512087BF53979CA9EABC9AB#REFe-r-25_011)). Such an AI agent might contain a specification of its own design. For example, the meta-reasoner in Sunny may have a specification of Sunny’s design, including its functions (e.g., its goals) and its mechanisms for achieving the functions (e.g., the method of map-based navigation planning). When Sunny generates a plan that fails on execution, Sunny’s meta-reasoner uses the specification of its design to diagnose and repair its reasoning process. If the feedback from the world on the failed plan pertains to an element of knowledge (e.g., at intersection A, I expected a road going directly toward downtown but when I reached there, I found no such road), then Sunny enters this new knowledge in its map of the city.

### AI Safety

It is important that we recognize that programming ethics into robots is not merely a programming issue, but an interdisciplinary problem requiring contribution from law, philosophy, psychology, sociology, and other fields.

### Putting It All Together

In this section, we took navigational planning as an example to illustrate how AI is putting together multiple capabilities ranging from perception, cognition, and action, to reasoning, learning, and memory, and on to reflection, deliberation, and situated action.

## A Very Brief History of AI

In the middle of the twentieth century, the scientific world experienced a shift in focus from descriptions of matter and energy to descriptions of information. One manifestation of information theory applied to real-world problems was in the field of *cybernetics* (Weiner, [1961](https://www-cambridge-org.ezproxy.snhu.edu/core/books/cambridge-handbook-of-intelligence/artificial-intelligence/B994B0D29512087BF53979CA9EABC9AB#REFe-r-25_071)), the study of communication and control in self-regulating analog systems. Cybernetics’ focus on analog signal contributed to its losing ground against discrete symbolic approaches common in AI. Not only did the symbolic approaches come to dominate AI research, but the symbol-processing approach came to dominate cognitive psychology as well.

The early exuberance of AI was tempered with the first “AI winter” that dominated the late 1960s and the 1970s, characterized by a decrease of optimism and funding, and caused by unfulfilled expectations. Early interest in associative processing was diminished by an influential book *Perceptrons* (Minsky & Papert, [1969](https://www-cambridge-org.ezproxy.snhu.edu/core/books/cambridge-handbook-of-intelligence/artificial-intelligence/B994B0D29512087BF53979CA9EABC9AB#REFe-r-25_040)) around the same time.

## Assessing Progress in AI

The task of measuring progress in AI is complex. In the past, tasks such as arithmetic and chess were considered to require intelligence. However, computers have been performing arithmetic calculations with great precision for more than seventy-five years and reliably beating human grand masters at chess for more than twenty-five (Hsu, Campbell, & Hoane, [1995](https://www-cambridge-org.ezproxy.snhu.edu/core/books/cambridge-handbook-of-intelligence/artificial-intelligence/B994B0D29512087BF53979CA9EABC9AB#REFe-r-25_020)).

Recently there have been proposals (Bringsjord & Schimanski, [2003](https://www-cambridge-org.ezproxy.snhu.edu/core/books/cambridge-handbook-of-intelligence/artificial-intelligence/B994B0D29512087BF53979CA9EABC9AB#REFe-r-25_009)) for using psychometrics tests of human intelligence, such as the Wechsler test ([1939](https://www-cambridge-org.ezproxy.snhu.edu/core/books/cambridge-handbook-of-intelligence/artificial-intelligence/B994B0D29512087BF53979CA9EABC9AB#REFe-r-25_070)) and the Raven’s test ([1962](https://www-cambridge-org.ezproxy.snhu.edu/core/books/cambridge-handbook-of-intelligence/artificial-intelligence/B994B0D29512087BF53979CA9EABC9AB#REFe-r-25_051)) to measure progress in AI. However, there already exist computer programs that approach human performance on various versions of the Raven’s test including the Standard, Color, and Advanced Raven’s test (e.g., Kunda, McGreggor, & Goel., [2013](https://www-cambridge-org.ezproxy.snhu.edu/core/books/cambridge-handbook-of-intelligence/artificial-intelligence/B994B0D29512087BF53979CA9EABC9AB#REFe-r-25_024)). Other recent proposals (Marcus, Rossi, & Veloso, [2016](https://www-cambridge-org.ezproxy.snhu.edu/core/books/cambridge-handbook-of-intelligence/artificial-intelligence/B994B0D29512087BF53979CA9EABC9AB#REFe-r-25_034)) for measuring AI have covered a wide range, from playing soccer and winning the FIFA world championship to scientific discovery and winning the Nobel Prize.

## AI and Society

In any case, almost everyone agrees that we are nowhere near having a superintelligent AI. The study of ethical AI behavior is a growing field of interest. A related issue is whether or not we will someday need to have ethical considerations for the AIs themselves, should they ever be able to suffer pain.

## Conclusions

In this chapter we have reviewed the history of AI and its major subfields, illustrated AI as a science and as a technology, examined its relationship to psychology, and discussed the problem of measuring the intelligence of AI agents. A somewhat surprising lesson from the history of AI is that it is relatively easy to make AI systems for some cognitive tasks that seem difficult for humans to solve (for example, mathematical, logical, and chess problems), and extraordinarily difficult to make computers solve some tasks that are apparently easy for humans to address (for example, seeing, walking, and talking). This apparent paradox has meant that repeated predictions about bold AI successes have gone unfulfilled.

# Reading: [Understanding Key Terms in AI](https://medium.com/datadriveninvestor/understanding-key-terms-in-ai-415baa8b37a1)

This article explains the different types of learning (reinforcement, machine, etc.) and algorithms that fall under the umbrella of AI. As you read, consider the following questions:

* What are the different types of learning associated with AI?
* How does machine learning work?

# Video: [What to Know about Data Science and Machine Learning in 2022 Peter Norvig](https://www.youtube.com/watch?v=KPDX47vf2pc)

In this video Peter Norvig, a director of research at Google, discusses some of the differences between traditional programming and machine learning and how AI is transforming how we live. As you watch, consider the following questions:

* How does AI programming differ from traditional programming?
* What are some challenges faced when using machine learning solutions?

# Video: [Machine Learning Zero to Hero (Google I/O'19) opens in new window](https://www.youtube.com/watch?v=VwVg9jCtqaU)

In this video, the concept of machine learning is explained from a programmer’s perspective. Different models such as neural networks are discussed, and the differences in the terminology between traditional coding and machine learning are explained. Examples of code are provided and discussed. As you watch, consider the following questions:

* How do you train a neural network?
  + Give answers and data and the machine develops the rules
  + Training phase
  + saved model
  + Deployment
    - web based
    - embedded devices
    - mobile
* How do you know a neural network is working correctly?
  + If the prediction of the machine matches the correct output
* What are convolutions and how do they help with images?
  + Look at every pixel in an image, look at its immediate neighbors and use a filter to get a new value for the pixels
  + Extracts features from an image to get rid of the “noise” in an image
  + Pooling compresses image by taking the largest value of a pool of pixels

# Reading: [An A.I. Glossary](https://www.nytimes.com/2018/10/18/business/an-ai-glossary.html)

This website provides a glossary of commonly used terms and their definitions in the field of artificial intelligence (AI). Consider bookmarking this page so that you can review terms that come up in your readings and assignments.

# Textbook: [Deep Learning with Keras](http://ezproxy.snhu.edu/login?url=https://ebookcentral.proquest.com/lib/snhu-ebooks/detail.action?docID=4850536), Chapter 1 (all) and Chapter 2 (pp. 59-70)

In Chapter 1 of this Shapiro Library textbook, you will be introduced to the basics of neural networks with a focus on perceptron, multilayer perceptron, and Keras as a model. This reading discusses problems for training the perceptron, and incorporating activation functions and back propagation as a solution. The required sections of Chapter 2 will introduce you to coding examples, architecture, and reporting. As you read, consider the following:

* What are some of the potential problems with the perceptron?
* What is a multilayer perceptron?
* What is an epoch?
* How do you modify hyperparameters in a neural network?

## Chapter 1: Neural Networks Foundations

Nets: Artificial neural networks represent a class of machine learning models loosely inspired by human brains

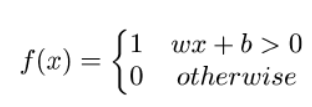
Each net is made up of several interconnected neurons, organized in layers, which exchange messages (they fire, in jargon) when certain conditions happen.

deep learning: a class of neural networks characterized by a significant number of layers of neurons, which are able to learn rather sophisticated models based on progressive levels of abstractions.

Deep has gone from 3-5 layers to 100-200 in the past few years

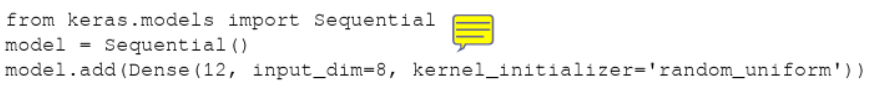
### Perceptron

The perceptron is a simple algorithm which, given an input vector x of m values often called input features or simply features, outputs either 1 (yes) or 0 (no). Mathematically, we define a function:



#### The first example of Keras code

The simplest mosel is called sequential

This code fragment defines a single layer with 12 artificial neurons, and it expects 9 input variables (ALSO KNOWN AS FEATURES)

Each neuron can be initialized with specific weights, Keras provides a few choices, the most common of which are listed as follows

* random\_uniform : Weights are initialized to uniformly random small values
* random\_normal : Weights are initialized according to a Guassian (think about a symmetric bell curve shape), with a zero mean and small standard deviation of 0.05
* zero : all weights are initialized to zero

A full list is available at https://keras.io/initializations/ .

### Multilayer perceptron

the first example of a network

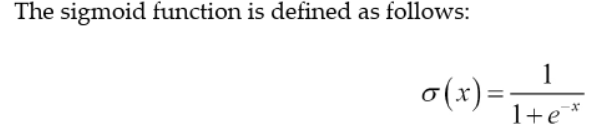
The net is dense, meaning that each neuron in a layer is connected to all neurons located in the previous layer and to all the neurons in the following layer

#### Problems in training the perceptron and a solution

You cannot progressively learn if you have a big jump in outputs

### Activation functions

#### sigmoid



it has a smooth and continuous change from 0-1

A sigmoid neuron can answer maybe instead of just yes or no

#### ReLU

rectified linear unit: f(x) = max(0,x)

0 for negative values, and grows linearly for positives

#### Activation functions

these are the functions that allow learning algorithms to adapt little by little

allows for a reduction in mistakes

Keras supports a number of activation functions, and a full list is available at https://keras.io/activations/ .

### A real example-recognizing handwritten digits

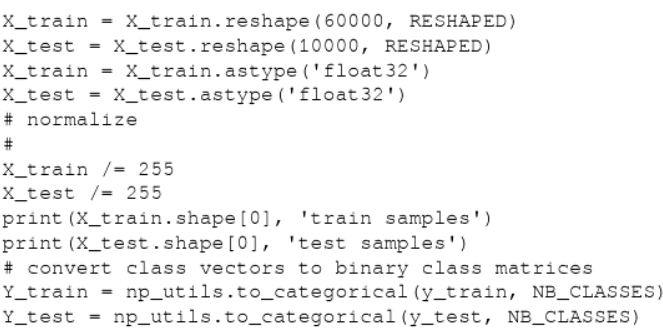
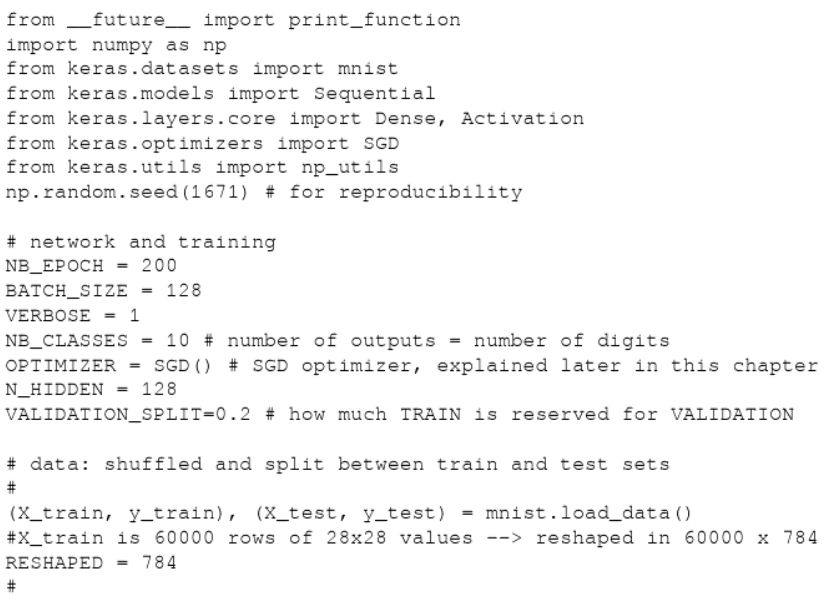
when a dataset has correct answers available, it is a form of supervised learning

#### One-hot encoding – OHE

In many applications, it is convenient to transform categorical (non-numerical) features into numerical variables. For instance, the categorical feature digit with the value d in [0-9] can be encoded into a binary vector with 10 positions, which always has 0 value, except the d -th position where a 1 is present. This type of representation is called one-hot encoding ( OHE ) and is very common in data mining when the learning algorithm is specialized for dealing with numerical functions.

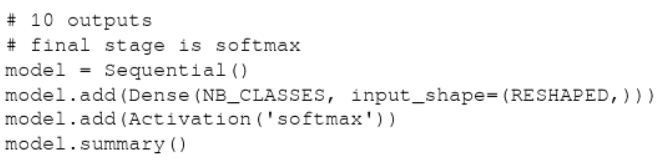
#### Defining a simple neural net in Keras

Keras provides suitable libraries to load the dataset and split it into training sets X\_train , used for fine-tuning our net, and tests set X\_test , used for assessing the performance. Data is converted into float32 for supporting GPU computation and normalized to [0, 1] . In addition, we load the true labels into Y\_train and Y\_test respectively and perform a onehot encoding on them. Let's see the code:



The input layer has a neuron for every single pixel. in a 28 x 28 grid there are 784 neurons

The final layer is a single neuron with activation function softmax, which is a generalization of the sigmoid function. Softmax squashes a k-dimensional vector of arbitrary real values into a k-dimensional vector of real values in the range (0-1). In our case, it aggregates 10 answers provided by the previous layer with 10 neurons:



Once we define the model, we have to compile it so that it can be executed by the Keras backend (either Theano or TensorFlow). There are a few choices to be made during compilation:

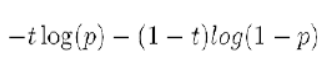
* We need to select the optimizer that is the specific algorithm used to update weights while we train our model
* We need to select the objective function that is used by the optimizer to navigate the space of weights (frequently, objective functions are called loss function , and the process of optimization is defined as a process of loss minimization )
* We need to evaluate the trained model

Some common choices for the objective function (a complete list of Keras objective functions is at https://keras.io/objectives/ ) are as follows:

* MSE : This is the mean squared error between the predictions and the true values. Mathematically, if is a vector of n predictions, and Y is the vector of n observed values, then they satisfy the following equation:



* + These objective functions average all the mistakes made for each prediction, and if the prediction is far from the true value, then this distance is made more evident by the squaring operation.
* Binary cross-entropy : This is the binary logarithmic loss. Suppose that our model predicts p while the target is t , then the binary cross-entropy is defined as follows:



* + This objective function is suitable for binary labels prediction.
* Categorical cross-entropy : This is the multiclass logarithmic loss. If the target is t i,j and the prediction is p i,j , then the categorical cross-entropy is this:



* + This objective function is suitable for multiclass labels predictions. It is also the default choice in association with softmax activation.

Some common choices for metrics (a complete list of Keras metrics is at https://keras.io /metrics/) are as follows:

* Accuracy: This is the proportion of correct predictions with respect to the targets
* Precision: This denotes how many selected items are relevant for a multilabel classification
* Recall: This denotes how many selected items are relevant for a multilabel classification

Metrics are similar to objective functions, with the only difference that they are not used for training a model but only for evaluating a model. Compiling a model in Keras is easy:

model.compile(loss='categorical\_crossentropy', optimizer=OPTIMIZER, metrics=['accuracy'])

Once the model is compiled, it can be then trained with the fit() function, which specifies a few parameters:

* epochs : This is the number of times the model is exposed to the training set. At each iteration, the optimizer tries to adjust the weights so that the objective function is minimized.
* batch\_size : This is the number of training instances observed before the optimizer performs a weight update.

Training a model in Keras is very simple. Suppose we want to iterate for NB\_EPOCH steps:

history = model.fit(X\_train, Y\_train,

batch\_size=BATCH\_SIZE, epochs=NB\_EPOCH,

verbose=VERBOSE, validation\_split=VALIDATION\_SPLIT)

We reserved part of the training set for validation. The key idea is that we reserve a part of the training data for measuring the performance on the validation while training. This is a good practice to follow for any machine learning task, which we will adopt in all our examples.

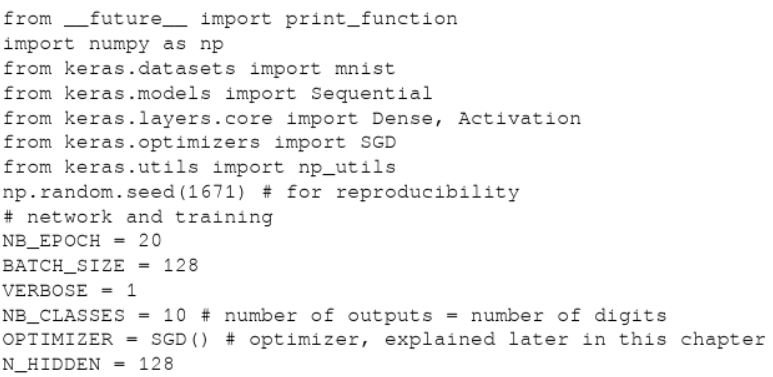
Once the model is trained, we can evaluate it on the test set that contains new unseen examples. In this way, we can get the minimal value reached by the objective function and best value reached by the evaluation metric.

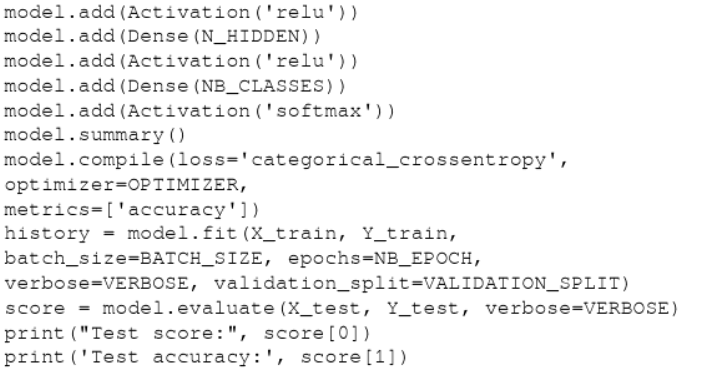
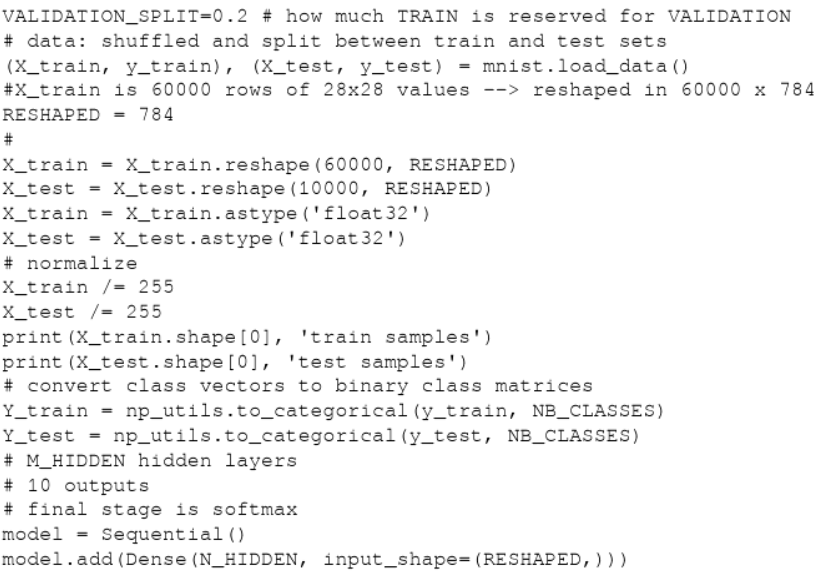
#### Running a simple Keras net and establishing a baseline

#### Improving the simple net in Keras with hidden layers

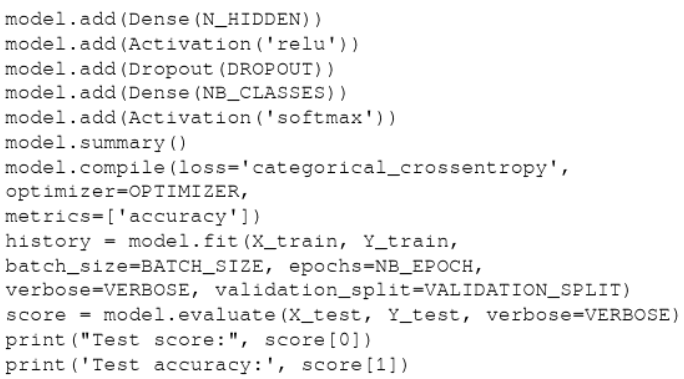
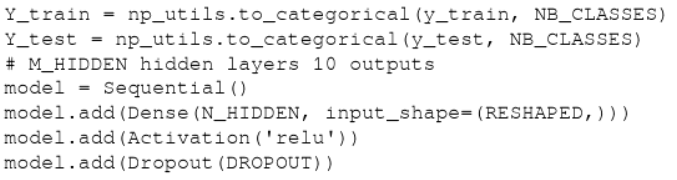
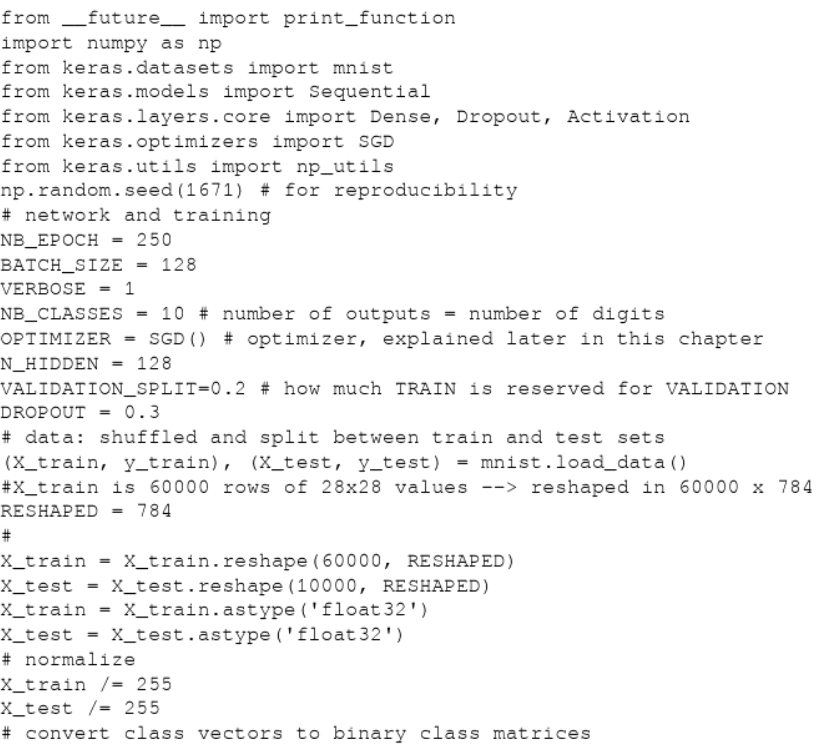
improve by adding additional layers to the network.

A layer is considered hidden if it is not directly connected to either the input or the output





#### Further improving the simple net in Keras with dropout



Training accuracy should always be above the test accuracy, otherwise the training is not long enough

random dropout improves the neurons because they can no longer rely on its neighbors, so since there is no dropout in testing all of the finer tuned neurons get to act

### Gradient descent

#### Testing different optimizers in Keras

You do not have to worry about doing the math, Keras uses its backend to compute that for you, but it involves using and finding the derivatives

### Stochastic gradient descent

Keras implements a fast variant of gradient descent known as stochastic gradient descent ( SGD ) and two more advanced optimization techniques known as RMSprop and Adam . RMSprop and Adam include the concept of momentum (a velocity component) in addition to the acceleration component that SGD has. This allows faster convergence at the cost of more computation. A full list of Keras-supported optimizers is at https://keras.io/optim izers/ . SGD was our default choice so far. So now let's try the other two. It is very simple, we just need to change few lines:

from keras.optimizers import RMSprop, Adam

...

OPTIMIZER = RMSprop() # optimizer,

or

Optimizer = Adam() # optimizer

#### Increasing the number of epochs

sometimes increasing the number of epochs does not result in better accuracy

#### Controlling the optimizer learning rate

changing the learning parameter for the optimizer can increase accuracy sometimes, the optimal value is around 0.001, which is default for Adam.

#### Increasing the number of internal hidden neurons

changing the number of hidden neurons can increase accuracy.

There is a cost/benefit for adding neurons, and while the run time always increases exponentially, the accuracy plateaus at a certain point.

#### Increasing the size of batch computation

there is an optimal size, for the example given the optimal batch size is 128

#### Adopting regularization for avoiding overfitting

Having a training model be an overly complex might not always be a good thing. It can drastically increase the execution time, or it could show positive results from the training because it has memorized it all, so it's good on training but not on validation.

learning should be more about generalization and less on memorization

There are three different types of regularizations used in machine learning:

* L1 regularization (also known as lasso ): The complexity of the model is expressed as the sum of the absolute values of the weights
* L2 regularization (also known as ridge ): The complexity of the model is expressed as the sum of the squares of the weights
* Elastic net regularization : The complexity of the model is captured by a combination of the two preceding techniques

the idea of regularization can be applied independently to the weights, the model, and to the activation

playing with the regularization can be a very good way of optimizing the performance of a model

### Hyperparameters tuning

There are a bunch of different options for fine tuning, and not all will work for every net. some things that can be fine-tuned are the number of hidden neurons, batch size, number of epochs, and many more according to the complexity of the net itself

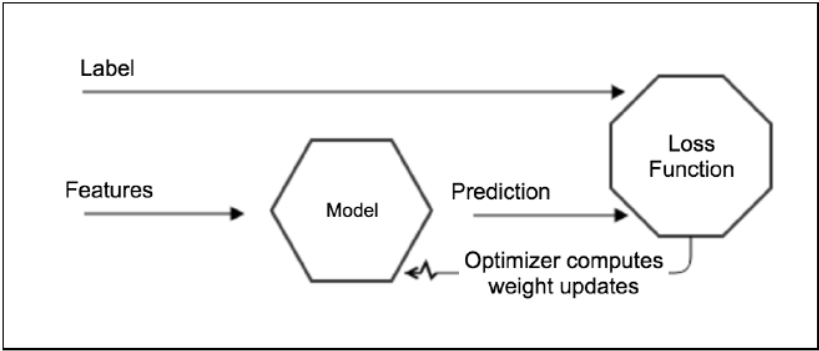
Hyperparameter tuning is the process of finding the optimal combination of those parameters that minimize the cost functions.

#### Predicting output

### A practical overview of backpropagation

Multilayer perceptrons learn from training data through a process called backpropagation. The process can be described as a way of progressively correcting mistakes as soon as they are detected.

The process of forward propagation from input to output and backward propagation of errors is repeated several times until the error gets below a predefined threshold. The whole process is represented in the following diagram:



## Chapter 2 pg 59-70

### Keras API

Keras has a modular, minimalist, and easy extendable architecture. Francois Chollet, the author of Keras, says:

The library was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Keras defines high-level neural networks running on top of either TensorFlow (for more information, refer to https://github.com/tensorflow/tensorflow ) or Theano (for more information, refer to https://github.com/Theano/Theano ). In details:

* Modularity: A model is either a sequence or a graph of standalone modules that can be combined together like LEGO blocks for building neural networks. Namely, the library predefines a very large number of modules implementing different types of neural layers, cost functions, optimizers, initialization schemes, activation functions, and regularization schemes.
* Minimalism: The library is implemented in Python and each module is kept short and self-describing.
* Easy extensibility: The library can be extended with new functionalities, as we will describe in Chapter 7, Additional Deep Learning Models.

### Getting started with Keras architecture

#### What is a tensor

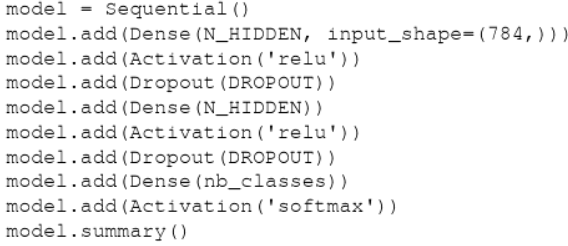
A tensor is nothing but a multidimensional array or matrix.

#### Composing models in Keras

two ways: Sequential composition and functional composition

##### Sequential Composition

different predefined models are stacked together in a linear popeline of layers similar to a stack or a queue.



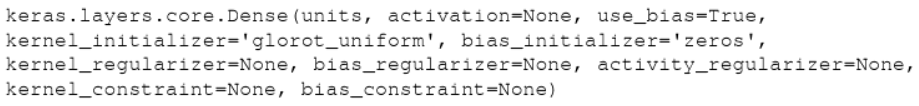
##### Functional composition

composing models via the functional API, where it is possible to define complex models, such as directed acyclic graphs, models with shared layers, or multi-output models.

### An overview of predefined neural network layers

#### Regular dense

found in chapter 1

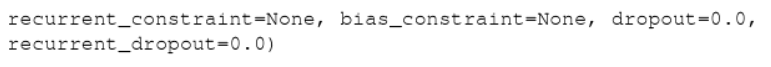


#### Recurrent neural networks – simple, LSTM, and GRU

class of neural networks that exploit the sequential nature of the input.

Example: text, speech, time, etc.

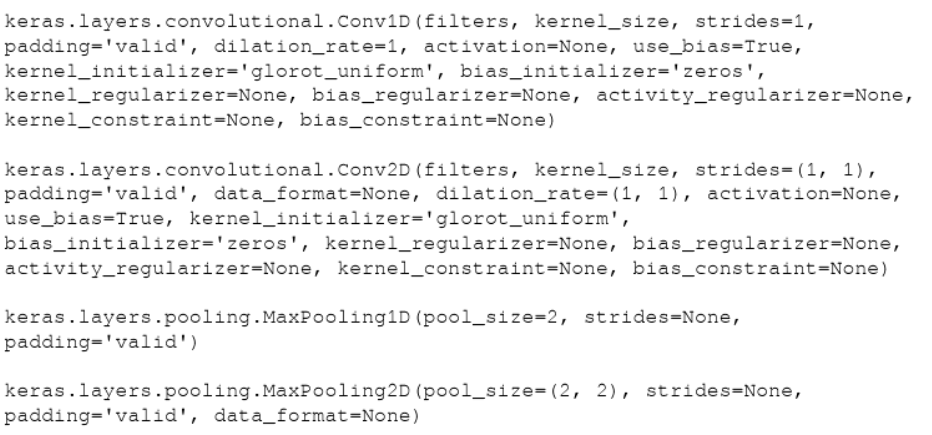
Found in chapter 6



#### Convolutional and pooling layers

ConvNets are a class of neural networks using convolutional and pooling operations for progressively learning rather sophisticated models based on progressive levels of abstraction

found in chapter 3



#### Regularization

A way to prevent overfitting

found in chapter 1

list of parameters commonly used for dense and convolutional modules:

* kernel\_regularizer : Regularizer function applied to the weight matrix
* bias\_regularizer : Regularizer function applied to the bias vector
* activity\_regularizer : Regularizer function applied to the output of the layer (its activation)

dropout for regularization is frequently an effective choice

keras.layers.core.Dropout(rate, noise\_shape=None, Where: seed=None)

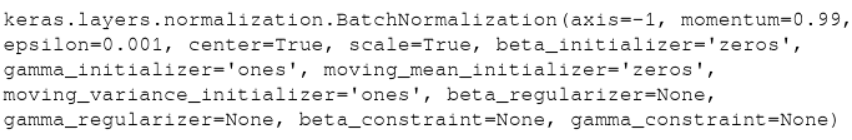
where:

* rate : It is a float between 0 and 1 which represents the fraction of the input units to drop
* noise\_shape : It is a 1D integer tensor which represents the shape of the binary dropout mask that will be multiplied with the input
* seed : It is a integer which is used use as random seed

#### Batch normalization

accelerate learning and generally achieve better accuracy

found in chapter 4



### An overview of predefined activation functions

sigmoid, linear, hyperbolic tangent, and ReLU.

### An overview of losses functions

* Accuracy which is used for classification problems. There are multiple choices: binary\_accuracy (mean accuracy rate across all predictions for binary classification problems), categorical\_accuracy (mean accuracy rate across all predictions for multiclass classification problems), sparse\_categorical\_accuracy (useful for sparse targets), and top\_k\_categorical\_accuracy (success when the target class is within the top\_k predictions provided).
* Error loss, which measures the difference between the values predicted and the values actually observed. There are multiple choices: mse (mean square error between predicted and target values), rmse (root square error between predicted and target values), mae (mean absolute error between predicted and target values), mape (mean percentage error between predicted and target values), and msle (mean squared logarithmic error between predicted and target values).
* Hinge loss, which is generally used for training classifiers. There are two versions: hinge defined as and squared hinge defined as the the squared value of the hinge loss.
* Class loss is used to calculate the cross-entropy for classification problems. There are multiple versions, including binary cross-entropy (for more information, refer to https://en.wikipedia.org/wiki/Cross\_entropy ), and categorical crossentropy.

### An overview of metrics

similar to an objective function.

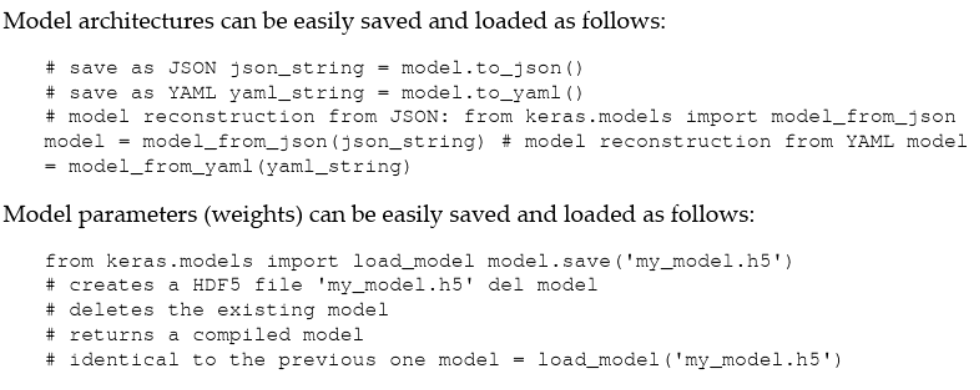
the results of evaluating a metric are not used when trainign a model.

### An overview of optimizers

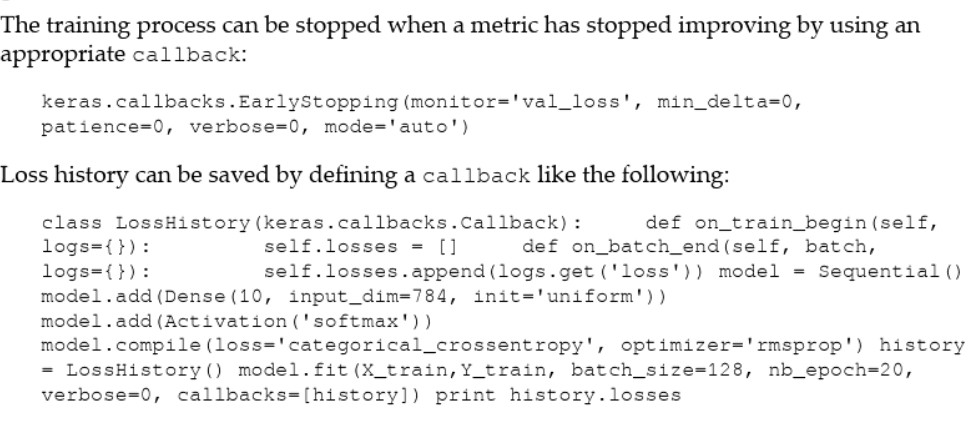
SGD, RMSprop, and Adam

### Some useful operations

#### Saving and loading the weights and the architecture of a model



#### Callbacks for customizing the training process



#### Checkpointing

saves a snapshot at regular intervals

* if you want the ability to restart from your last checkpoint after your AWS spot instance
* if you want to stop training, perhaps to test your model, then continue training
* if you want to retain the best version

### Using TensorBoard and Keras

### Using Quiver and Keras

found in chapter 3

### Summary

In this chapter, we discussed how to install Theano, TensorFlow, and Keras on the following:

* Your local machine
* A dockerized infrastructure based on containers
* In the cloud with Google GCP, Amazon AWS, and Microsoft Azure

In addition to that, we looked at a few modules defining Keras APIs and some commonly useful operations such as loading and saving neural networks' architectures and weights, early stopping, history saving, checkpointing, interactions with TensorBoard, and interactions with Quiver.

In the next chapter, we will introduce the concept of convolutional networks a fundamental innovation in deep learning which has been used with success in multiple domains from text, to video, to speech going well beyond the initial image processing domain where they were originally conceived.

# Reading: [Overview of Neuron Structure and Function](https://www.khanacademy.org/science/biology/human-biology/neuron-nervous-system/a/overview-of-neuron-structure-and-function)

This reading provides a very brief overview of the structure of a neuron in the human brain, as well as the structure of neural networks in the human brain. Understanding the basic structure of neurons and neural networks will give you a better understanding of how AI neural networks were designed to mimic structures and functions of the human brain. As you read, consider the following:

* What are the three basic functions of a neuron?
* Why are neural networks important?

All of these processes depend on the interconnected cells that make up your nervous system. Like the heart, lungs, and stomach, the nervous system is made up of specialized cells. These include nerve cells (or neurons) and glial cells (or glia). Neurons are the basic functional units of the nervous system, and they generate electrical signals called action potentials, which allow them to quickly transmit information over long distances. Glia are also essential to nervous system function, but they work mostly by supporting the neurons.

# Reading: [AI Is Not Similar to Human Intelligence. Thinking So Could Be Dangerous](https://www.forbes.com/sites/fernandezelizabeth/2019/11/30/ai-is-not-similar-to-human-intelligence-thinking-so-could-be-dangerous/#3fbd0b846c22)

Fernandez, E. (2019, November 30). AI Is Not Similar To Human Intelligence. Thinking So Could Be Dangerous. Forbes.com. Retrieved March 12, 2024, from <https://www.forbes.com/sites/fernandezelizabeth/2019/11/30/ai-is-not-similar-to-human-intelligence-thinking-so-could-be-dangerous/?sh=5d57e8266c22>

(Fernandez, 2019)

This reading discusses some of the important differences between human and machine intelligence. It contains a brief overview of what a neural network is before expanding upon some key differences regarding how neural networks “learn”. As you read, consider the following:

* How is the concept of a neural network related to the function of a human brain?
* What are the three key differences Watson sees between human and machine intelligence?
* Why is it important to understand that machines ‘think’ differently?

“neural networks” - an algorithm modeled after the human brain

The name originates from the idea behind neurons and synapses within the brain.

No doubt, these algorithms are powerful, but to think that they “think” and “learn” in the same way as humans would be incorrect, Watson [says](https://link.springer.com/article/10.1007/s11023-019-09506-6). There are many differences, and he outlines three.

* The first - DNNs are easy to fool. For example, imagine you have a picture of a banana. A neural network successfully classifies it as a banana. But it’s possible to create a generative adversarial network that can fool your DNN. By adding a slight amount of noise or another image besides the banana, your DNN might now think the picture of a banana is a toaster. A human could not be fooled by such a trick. Some argue that this is because DNNs can see things humans can’t, but Watson says, “This disconnect between biological and artificial neural networks suggests that the latter lack some crucial component essential to navigating the real world.”
* Secondly, DNNs need an enormous amount of data to learn. An image classification DNN might need to “see” thousands of pictures of zebras to identify a zebra in an image. Give the same test to a toddler, and chances are s/he could identify a zebra, even one that’s partially obscured, by only seeing a picture of a zebra a few times. Humans are great “one-shot learners,” says Watson. Teaching a neural network, on the other hand, might be very difficult, especially in instances where data is hard to come by.
* Thirdly, neural nets are “myopic”. They can see the trees, so to speak, but not the forest. For example, a DNN could successfully label a picture of Kim Kardashian as a woman, an entertainer, and a starlet. However, switching the position of her mouth and one of her eyes actually improved the confidence of the DNN’s prediction. The DNN didn’t see anything wrong with that image. Obviously, something is wrong here. Another example - a human can say “that cloud looks like a dog”, whereas a DNN would say that the cloud is a dog.

|  |
| --- |
| Additional Support (Optional) |

**Textbook:** [*Deep Learning with Keras* opens in new window](http://ezproxy.snhu.edu/login?url=https://ebookcentral.proquest.com/lib/snhu-ebooks/detail.action?docID=4850536), Chapter 2 (pp. 44-58)  
This optional section of Chapter 2 guides you through installing Theano, TensorFlow, and Keras in different environments. Although the environment you will be working in throughout this course is set up for you, it is still helpful to understand the installation process.

**Reading:** [A Beginner’s Guide to Neural Networks and Deep Learning opens in new window](https://pathmind.com/wiki/neural-network)This optional reading presents a very high-level view of neural networks as well as providing some specific examples of how they can be used. This reading is considered optional because it reinforces many of the same concepts as the textbook readings, just presented in a different way. However, it may be helpful if you would like to see more examples of neural networks.

# Textbook: [Deep Learning with Keras](http://ezproxy.snhu.edu/login?url=https://ebookcentral.proquest.com/lib/snhu-ebooks/detail.action?docID=4850536) chapter 3

In this chapter from your Shapiro Library textbook, you will review the concepts of convolutional neural networks (ConvNets) and how they are used to classify images based on psychological experiments done on the visual cortex. The focus for this reading is improving the performance and accuracy of image recognition using deep learning ConvNets, deep learning classifiers (CIFAR-10 and ImageNet datasets), large deep learning networks (VGG16), and very deep networks (InceptionV3). As you read, consider the following:

* How are convolution and pooling applied in convolutional neural nets (CNNs)?
* How do CNNs compare to the neural networks you learned about in the previous unit in their speed and accuracy?

In the context of dense nets for the handwritten dataset, each pixel was assigned to a neuron.

Convolutional neural networks (ConvNet) leverage spatial information and are therefore well suited for classifying images

inspired by biological and psychological experiments done on the human visual cortex

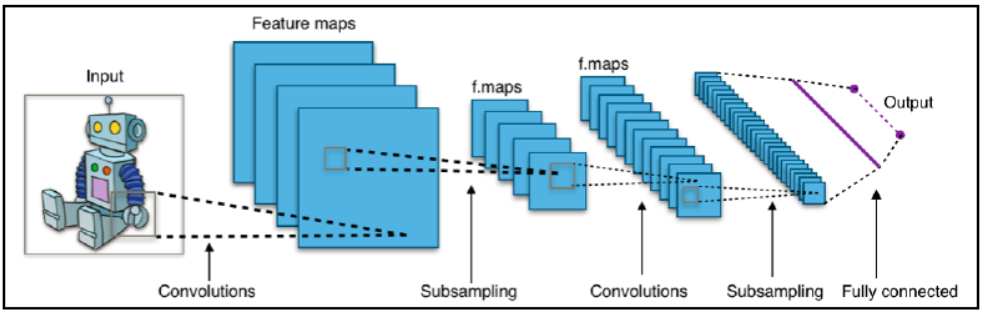
They have become a disruptive tech in text, video, speech, etc.

## Deep convolutional neural network (DCNN)

consists of many neural network layers.

two types are typically alternated

* convolutional
* pooling

Three key intuitions beyond ConvNets

* Local Receptive Fields
* Shared weights
* Pooling

### Local Receptive Fields

**Stride Length**: the size of each single submatrix in Keras

Hyperparameter that can be fine-tunes during the construction of our nets

### Shared weights and bias

In Keras, if we want to add a convolutional layer with dimensionality of the output 32 and extension of each filter 3x3, we will write:

model = Sequential()

model.add(Conv2D(32, (3, 3), input\_shape=(256, 256, 3))

Alternatively, we will write:

model = Sequential()

model.add(Conv2D(32, kernel\_size=3, input\_shape=(256, 256, 3))

applying a 3x3 convolution on a 256x256 image with 3 input channels (or input filters), resulting in 32 output channels (or output filters).

### Pooling Layers

summarize the output of a feature map.

we can use the spatial contiguity of the output produced from a single feature map and aggregate the values of a submatrix into a single output value that synthetically describes the meaning associated with that physical region.

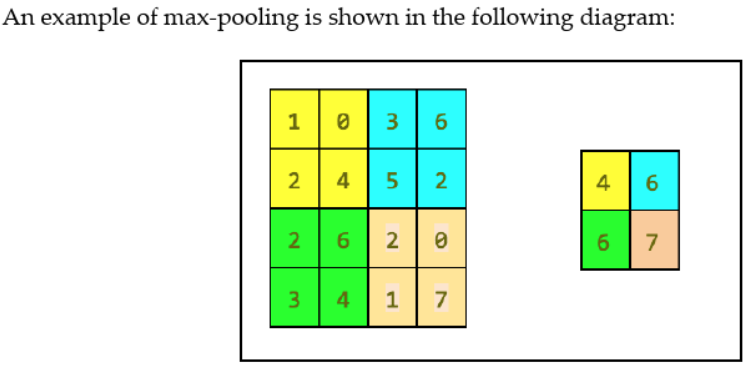
#### Max-Pooling

easy and common choice

outputs the maximum activation as observed in the region.

define a max-pooling layer of size 2x2

model.add(MaxPooling2D(pool\_size = (2, 2)))



#### Average Pooling

same as max pooling, but instead of taking the max, you take the average of the region

### ConvNets summary

* 1 dimension for audio and text along time
* 2 dimension for image along the height x width
* 3 dimensions for video heigh x width x time

## An Example of DCNN – LeNet

family of convnets trained for recognizing MNIST handwritten characters with robustness to simple geometric transformations

### LeNet code in Keras

define LeNet code, use convolutional 2D module

keras.layers.convolutional.conv2D(filters, kernel\_size, padding=’valid’)

* filters: the number of convolution kernels to use
* kernel\_size integer or tuple/list of two integers, specifying width and height of the 2D convolution window
* padding='valid’ means that the convolution is only computed where the input and the filter fully overlap, and therefore the output is smaller than the input
* padding=’same’ means that we have an output that is the same size as the input for which the area around the input is padded with zeros

in addition we use a MaxPooling2D module:

keras.layers.pooling.MaxPooling2D(pool\_size=(2, 2), strides=(2, 2))

* pool\_size(2, 2) is a tuple of two integers representing the factors by which the image is vertically and horizontally downscaled
  + (2, 2) will halve the image in each dimension
* strides=(2, 2) is the stride used for processing

Now, let us review the code. First we import a number of modules:

from keras import backend as K

from keras.models import Sequential

from keras.layers.convolutional import Conv2D

from keras.layers.convolutional import MaxPooling2D

from keras.layers.core import Activation

from keras.layers.core import Flatten

from keras.layers.core import Dense

from keras.datasets import mnist

from keras.utils import np\_utils

from keras.optimizers import SGD, RMSprop, Adam

import numpy as np

import matplotlib.pyplot as plt

Then we define the LeNet network:

#define the ConvNet

class LeNet:

@staticmethod

def build(input\_shape, classes):

model = Sequential()

# CONV => RELU => POOL

Note that since the Convolution2D is the first stage of our pipeline, we are also required to define its input\_shape . The max-pooling operation implements a sliding window that slides over the layer and takes the maximum of each region with a step of two pixels vertically and horizontally:

model.add(Convolution2D(20, kernel\_size=5, padding="same",

input\_shape=input\_shape))

model.add(Activation("relu"))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))

# CONV => RELU => POOL

Then a second convolutional stage with ReLU activations follows, again by a max-pooling. In this case, we increase the number of convolutional filters learned to 50 from the previous 20. Increasing the number of filters in deeper layers is a common technique used in deep learning:

model.add(Conv2D(50, kernel\_size=5, border\_mode="same"))

model.add(Activation("relu"))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))

Then we have a pretty standard flattening and a dense network of 500 neurons, followed by a softmax classifier with 10 classes:

# Flatten => RELU layers

model.add(Flatten())

model.add(Dense(500))

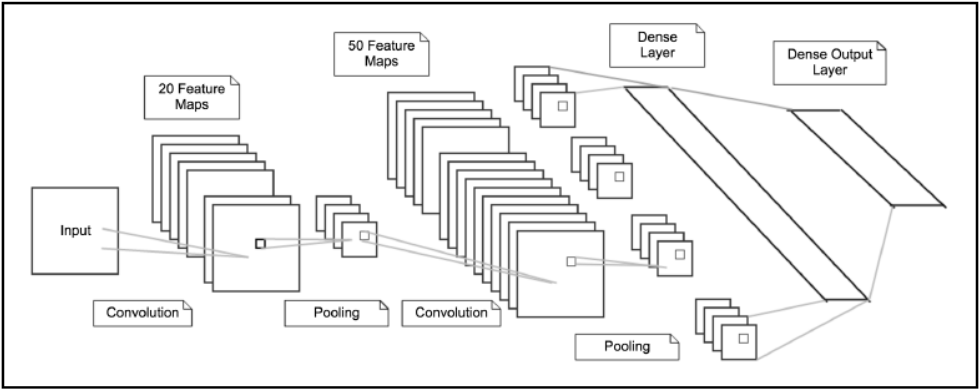
model.add(Activation("relu"))

# a softmax classifier

model.add(Dense(classes))

model.add(Activation("softmax"))

return model

Code for training the network

# network and training

NB\_EPOCH = 20

BATCH\_SIZE = 128

VERBOSE = 1

OPTIMIZER = Adam()

VALIDATION\_SPLIT=0.2

IMG\_ROWS, IMG\_COLS = 28, 28 # input image dimensions

NB\_CLASSES = 10 # number of outputs = number of digits

INPUT\_SHAPE = (1, IMG\_ROWS, IMG\_COLS)

# data: shuffled and split between train and test sets

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

k.set\_image\_dim\_ordering("th")

# consider them as float and normalize

X\_train = X\_train.astype('float32')

X\_test = X\_test.astype('float32')

X\_train /= 255

X\_test /= 255

# we need a 60K x [1 x 28 x 28] shape as input to the CONVNET

X\_train = X\_train[:, np.newaxis, :, :]

X\_test = X\_test[:, np.newaxis, :, :]

print(X\_train.shape[0], 'train samples')

print(X\_test.shape[0], 'test samples')

# convert class vectors to binary class matrices

y\_train = np\_utils.to\_categorical(y\_train, NB\_CLASSES)

y\_test = np\_utils.to\_categorical(y\_test, NB\_CLASSES)

# initialize the optimizer and model

model = LeNet.build(input\_shape=INPUT\_SHAPE, classes=NB\_CLASSES)

model.compile(loss="categorical\_crossentropy", optimizer=OPTIMIZER,

metrics=["accuracy"])

history = model.fit(X\_train, y\_train,

batch\_size=BATCH\_SIZE, epochs=NB\_EPOCH,

verbose=VERBOSE, validation\_split=VALIDATION\_SPLIT)

score = model.evaluate(X\_test, y\_test, verbose=VERBOSE)

print("Test score:", score[0])

print('Test accuracy:', score[1])

# list all data in history

print(history.history.keys())

# summarize history for accuracy

plt.plot(history.history['acc'])

plt.plot(history.history['val\_acc'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

# summarize history for loss

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

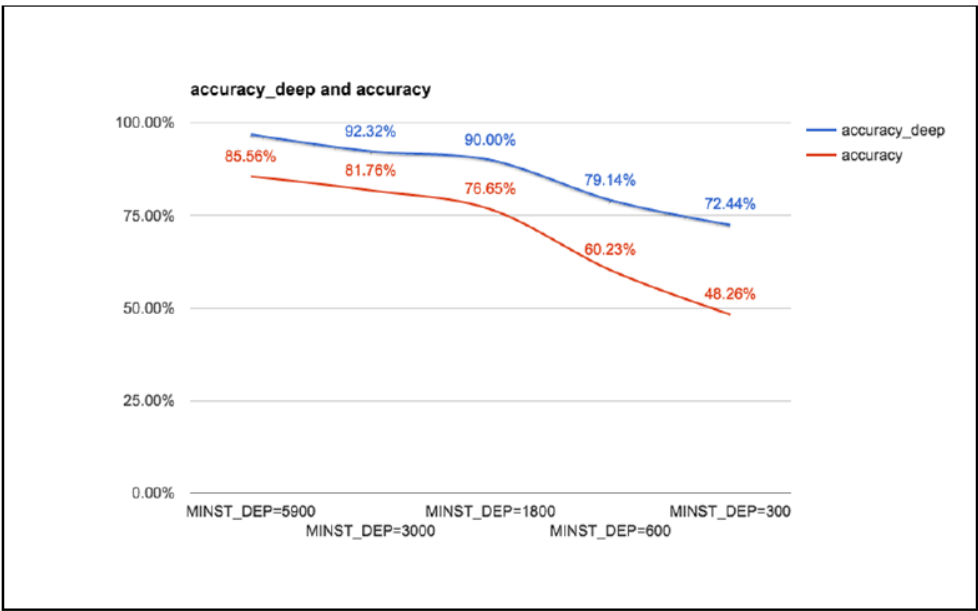
The time increased from 1-2 sec per iteration to 134 seconds, but the accuracy is now 99.06%

using the plots the training can be done in 4-5 iterations and achieve 99.2 percent accuracy

### Understanding the power of deep learning

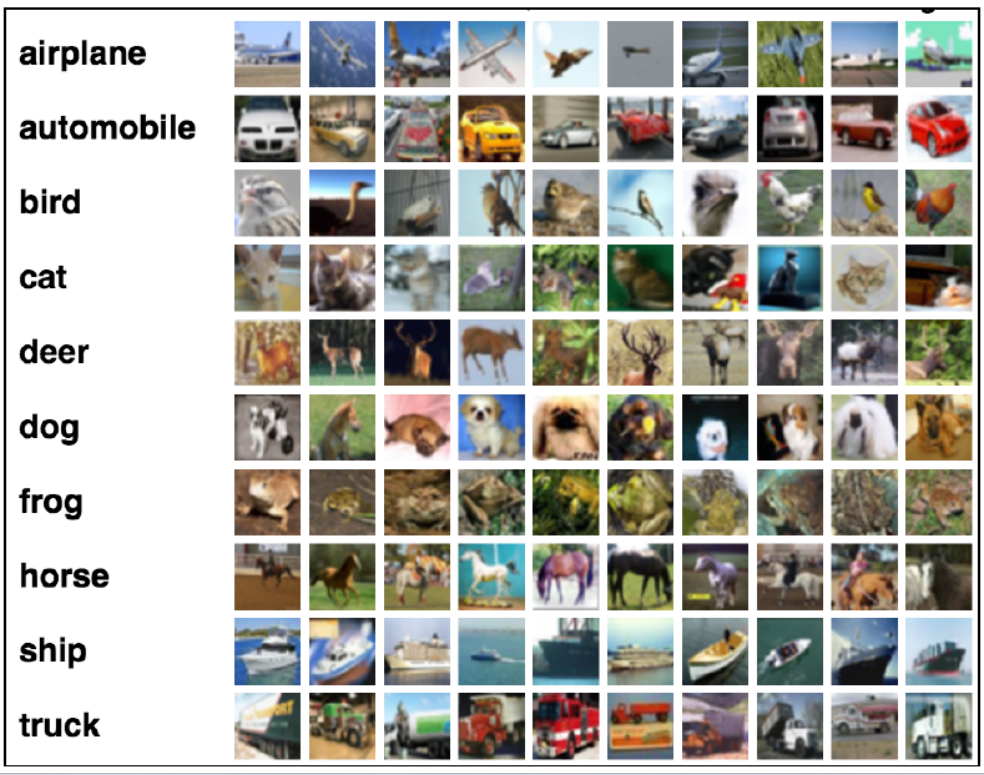
Another test is to reduce the size of the training set and observe the consequent decay in performance

the better the net, the less training examples are necessary. when comparing the ConvNet to the simple one in chapter 1, the convnet always outperforms and the gap widens with less training



## Recognizing CIFAR-10 images with deep learning

The CIFAR-10 dataset contains 60,000 color images of 32 x 32 pixels in 3 channels divided into 10 classes. Each class contains 6,000 images. The training set contains 50,000 images, while the test sets provides 10,000 images. This image taken from the CIFAR repository ( htt ps://www.cs.toronto.edu/~kriz/cifar.html ) describes a few random examples from the 10 classes:



First of all we import a number of useful modules, define a few constants, and load the dataset:

from keras.datasets import cifar10

from keras.utils import np\_utils

from keras.models import Sequential

from keras.layers.core import Dense, Dropout, Activation, Flatten

from keras.layers.convolutional import Conv2D, MaxPooling2D

from keras.optimizers import SGD, Adam, RMSprop

import matplotlib.pyplot as plt

# CIFAR\_10 is a set of 60K images 32x32 pixels on 3 channels

IMG\_CHANNELS = 3

IMG\_ROWS = 32

IMG\_COLS = 32

#constant

BATCH\_SIZE = 128

NB\_EPOCH = 20

NB\_CLASSES = 10

VERBOSE = 1

VALIDATION\_SPLIT = 0.2

OPTIM = RMSprop()

#load dataset

(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()

print('X\_train shape:', X\_train.shape)

print(X\_train.shape[0], 'train samples')

print(X\_test.shape[0], 'test samples')

# one-hot encoding and normalize the images

# convert to categorical

Y\_train = np\_utils.to\_categorical(y\_train, NB\_CLASSES)

Y\_test = np\_utils.to\_categorical(y\_test, NB\_CLASSES)

# float and normalization

X\_train = X\_train.astype('float32')

X\_test = X\_test.astype('float32')

X\_train /= 255

X\_test /= 255

# net will learn 32 convolutional filters, each with a 3x3 size. max-pooling operation with pool size 2x2 and a dropout at 25%

# network

model = Sequential()

model.add(Conv2D(32, (3, 3), padding='same',

input\_shape=(IMG\_ROWS, IMG\_COLS, IMG\_CHANNELS)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

# Dense network with 512 units and ReLU activation followed by a dropout at 50% and by a softmax layer with 10 classes as output, one for each category

model.add(Flatten())

model.add(Dense(512))

model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(NB\_CLASSES))

model.add(Activation('softmax'))

model.summary()

# After defining the network, train the model.

# Split the data into validation, training, and testing sets.

# Training is used to build model, validation is used to select the best performing approach, the test set is to check the performance of the best model on fresh unseen data

# train

model.compile(loss='categorical\_crossentropy', optimizer=OPTIM,

metrics=['accuracy'])

model.fit(X\_train, Y\_train, batch\_size=BATCH\_SIZE,

epochs=NB\_EPOCH, validation\_split=VALIDATION\_SPLIT,

verbose=VERBOSE)

score = model.evaluate(X\_test, Y\_test, batch\_size=BATCH\_SIZE, verbose=VERBOSE)

print("Test score:", score[0])

print('Test accuracy:', score[1])

# Save the architecture of the deep network

#save model

model\_json = model.to\_json()

open('cifar10\_architecture.json', 'w').write(model\_json)

# And the weights learned by our deep network on the training model.save\_weights('cifar10\_weights.h5', overwrite=True)

### Improving the CIFAR-10 performance with deeper a network

in this example we have a sequence of modules

conv+conv+maxpool+dropout+conv+conv+maxpool

followed by a standard dense+dropout+dense All the activation functions are ReLU

model = Sequential()

model.add(Conv2D(32, (3, 3), padding='same',

input\_shape=(IMG\_ROWS, IMG\_COLS, IMG\_CHANNELS)))

model.add(Activation('relu'))

model.add(Conv2D(32, (3, 3), padding='same'))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))

model.add(Activation('relu'))

model.add(Conv2D(64, 3, 3))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512))

model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(NB\_CLASSES))

model.add(Activation('softmax'))

### Improving the CIFAR-10 performance with data augmentation

another way to improve performance is to generate more images for training. The key intuition is that we can take the standard CIFAR training set and augment this set with multiple types of transformations including rotation, rescaling, horizontal/vertical flip, zooming, channel shift, and many more

from keras.preprocessing.image import ImageDataGenerator

from keras.datasets import cifar10

import numpy as np

NUM\_TO\_AUGMENT=5

#load dataset

(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()

# augumenting

print("Augmenting training set images...")

datagen = ImageDataGenerator(

rotation\_range=40,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True,

fill\_mode='nearest')

* rotation\_range is a value in gegrees (0-180) for random rotation
* width\_shift and height\_shift are ranges for randomly translating pictures vertically or horizontally
* zoom\_range is for random zooming
* horizontal\_flip is for randomly flippling half of the images
* fill\_mode is the strategy used for filling in new pixels that can appear after a rotation or a shift

xtas, ytas = [], []

for i in range(X\_train.shape[0]):

num\_aug = 0

x = X\_train[i] # (3, 32, 32)

x = x.reshape((1,) + x.shape) # (1, 3, 32, 32)

for x\_aug in datagen.flow(x, batch\_size=1, save\_to\_dir='preview', save\_prefix='cifar', save\_format='jpeg'):

if num\_aug >= NUM\_TO\_AUGMENT:

break

xtas.append(x\_aug[0])

num\_aug += 1

we can apply this intuition directly for training. Using the same ConvNet defined preiouslt we can generate more augmented images and then we train. For efficiency the generation runs parallel to the model

#fit the dataget

datagen.fit(X\_train)

# train

history = model.fit\_generator(datagen.flow(X\_train, Y\_train,

batch\_size=BATCH\_SIZE), samples\_per\_epoch=X\_train.shape[0],

epochs=NB\_EPOCH, verbose=VERBOSE)

score = model.evaluate(X\_test, Y\_test,

batch\_size=BATCH\_SIZE, verbose=VERBOSE)

print("Test score:", score[0])

print('Test accuracy:', score[1])

### Predicting with CIFAR-10

if we want to use the deep learning model that we just trained, since we saved the model and the weights, we do not need to train every time:

import numpy as np

import scipy.misc

from keras.models import model\_from\_json

from keras.optimizers import SGD

#load model

model\_architecture = 'cifar10\_architecture.json'

model\_weights = 'cifar10\_weights.h5'

model = model\_from\_json(open(model\_architecture).read())

model.load\_weights(model\_weights)

#load images

img\_names = ['cat-standing.jpg', 'dog.jpg']

imgs = [np.transpose(scipy.misc.imresize(scipy.misc.imread(img\_name), (32,

32)),

(1, 0, 2)).astype('float32')

for img\_name in img\_names]

imgs = np.array(imgs) / 255

# train

optim = SGD()

model.compile(loss='categorical\_crossentropy', optimizer=optim,

metrics=['accuracy'])

# predict

predictions = model.predict\_classes(imgs)

print(predictions)

## Very deep convolutional networks for large-scale image recognition

from keras.models import Sequential

from keras.layers.core import Flatten, Dense, Dropout

from keras.layers.convolutional import Conv2D, MaxPooling2D, ZeroPadding2D

from keras.optimizers import SGD

import cv2, numpy as np

# define a VGG16 network

def VGG\_16(weights\_path=None):

model = Sequential()

model.add(ZeroPadding2D((1,1),input\_shape=(3,224,224)))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(ZeroPadding2D((1,1)))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))

model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(ZeroPadding2D((1,1)))

model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))

model.add(Conv2D(256, (3, 3), activation='relu'))

model.add(ZeroPadding2D((1,1)))

model.add(Conv2D(256, (3, 3), activation='relu'))

model.add(ZeroPadding2D((1,1)))

model.add(Conv2D(256, (3, 3), activation='relu'))

model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))

model.add(Conv2D(512, (3, 3), activation='relu'))

model.add(ZeroPadding2D((1,1)))

model.add(Conv2D(512, (3, 3), activation='relu'))

model.add(ZeroPadding2D((1,1)))

model.add(Conv2D(512, (3, 3), activation='relu'))

model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))

model.add(Conv2D(512, (3, 3), activation='relu'))

model.add(ZeroPadding2D((1,1)))

model.add(Conv2D(512, (3, 3), activation='relu'))

model.add(ZeroPadding2D((1,1)))

model.add(Conv2D(512, (3, 3), activation='relu'))

model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Flatten())

#top layer of the VGG net

model.add(Dense(4096, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(4096, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(1000, activation='softmax'))

if weights\_path:

model.load\_weights(weights\_path)

return model

### Recognizing cats with a VGG-16 net

Can recognize an Egyptian cat

### Utilizing Keras built-in VGG-16 net module

Using built-in code is very eay and weights are downloaded aoutomatically when instantiating a model and stored at `/.keras/models/

from keras.models import Model

from keras.preprocessing import image

from keras.optimizers import SGD

from keras.applications.vgg16 import VGG16

import matplotlib.pyplot as plt

import numpy as np

import cv2

# prebuild model with pre-trained weights on imagenet

model = VGG16(weights='imagenet', include\_top=True)

sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

model.compile(optimizer=sgd, loss='categorical\_crossentropy')

# resize into VGG16 trained images' format

im = cv2.resize(cv2.imread('steam-locomotive.jpg'), (224, 224))

im = np.expand\_dims(im, axis=0)

# predict

out = model.predict(im)

plt.plot(out.ravel())

plt.show()

print np.argmax(out)

#this should print 820 for steaming train

To conclude this section, note that VGG-16 is only one of the modules that are pre-built in Keras. A full list of pre-trained Keras models is available at: https://keras.io/applications/ .

### Recycling pre-built deep learning models for extracting features

One very simple idea is to use VGG-16 and, more generally, DCNN, for feature extraction.

from keras.applications.vgg16 import VGG16

from keras.models import Model

from keras.preprocessing import image

from keras.applications.vgg16 import preprocess\_input

import numpy as np

# pre-built and pre-trained deep learning VGG16 model

base\_model = VGG16(weights='imagenet', include\_top=True)

for i, layer in enumerate(base\_model.layers):

print (i, layer.name, layer.output\_shape)

# extract features from block4\_pool block

model = Model(input=base\_model.input,

output=base\_model.get\_layer('block4\_pool').output)

img\_path = 'cat.jpg'

img = image.load\_img(img\_path, target\_size=(224, 224))

x = image.img\_to\_array(img)

x = np.expand\_dims(x, axis=0)

x = preprocess\_input(x)

# get the features from this block

features = model.predict(x)

### Very deep inception-v3 net used for transfer learning

if you want to fine tune a model, do not include the top layer

# Video: [Machine Learning and Human Bias](https://www.youtube.com/watch?v=59bMh59JQDo)

This video explains the concept of machine learning and how human bias affects biases in the technology we create. As you watch, consider the following:

* How do biases become a part of technology?
  + by a programmer instituting their own bias, often by accident
* What are the different biases mentioned in the video? Are there other biases that you can think of that were not covered here?

# Reading: [Human Bias in Machine Learning](https://towardsdatascience.com/bias-what-it-means-in-the-big-data-world-6e64893e92a1)

Bias is a part of human nature and can be caused by experiences or environment, but it also exists in machine learning and is referred to as machine bias. This article addresses different types of bias that exist with machine learning algorithms, how and why they occur, and their implications toward society. As you read, consider the following:

* In addition to biases that exist in data, there are cognitive biases that affect human beings. What are some of the cognitive biases mentioned? How might these cognitive biases affect someone developing AI technology?
* What are the different categories of machine bias mentioned in the article? What dangers or inequities might result from each of these types of bias?
  + Sampling bias, a form of bias that comes from imbalanced training data which doesn’t represent the environment that the model will operate in.
    - People of color are harder for self driving cars to recognize
  + Prejudice Bias arises when algorithms take in subtle biases from the data source, even if it was sampled perfectly.
    - if the sample has more women in kitchens then men, then the model will infer that if someone is in a kitchen then they are women
  + Algorithmic bias is when the algorithm, due to how it is designed, will have bias built in.
    - spotifys year playlist is biased on the songs you listened to 2 years ago because of the playtime on those songs
* What steps can be taken to counteract machine biases?
  + be aware of the possible biases
  + have domain knowledge on the data you are working with to be aware of the intricacies that exist within it
  + bias isn’t intrinsically a bad thing
    - bias towards favorite songs don’t tend to change much
    - bias for messages to be business formal on a site like LinkedIn

The term for the bias that affects Machine Learning algorithms is **Machine Bias**.

# Reading: [Ethics and Privacy in AI and Big Data: Implementing Responsible Research and Innovation](https://ezproxy.snhu.edu/login?url=https://dx.doi.org/10.1109/MSP.2018.2701164)

There are concerns about privacy and ethical issues involved with the combination of big data and artificial intelligence. This article from the Shapiro Library discusses ways to address these issues to ensure that the advantages outweigh the disadvantages using the concept of responsible research and innovation (RRI). As you read, consider the following:

* What are some of the concerns raised by Smart Information Systems (SIS)?
  + privacy and data protection, autonomy of users, their agency, trust, consent, identity, inclusion and digital divides, security, harm, misuse, and deception, to name just a few.
* What different measures have been proposed to address ethical issues arising from AI? How does the GDPR specifically attempt to address these ethical issues?
  + European General Data Protection Regulation (GDPR) explicitly address the impact of SIS. Among the novel features relevant to SIS are breach notifications, hefty financial penalties, data protection impact assessments,**22** privacy by design,**16** and the so-called *right to be forgotten*.
* What is RRI and how does it differ from other approaches? How might RRI be implemented?
  + responsible research and innovation (RRI)
  + The concept of RRI and the discourses and practices that it has spawned offer a promising and realistic way of dealing with these challenges. RRI is a relatively novel concept that has gained prominence since about 2010. It is an attempt to rethink research and innovation governance with a view to ensuring that processes as well as outcomes of research and innovation are acceptable, desirable, and sustainable.

Information and communication technologies (ICTs) have long been recognized as having significant social and economic impact, such that they require regulatory supervision and call for ethical and social assessment.

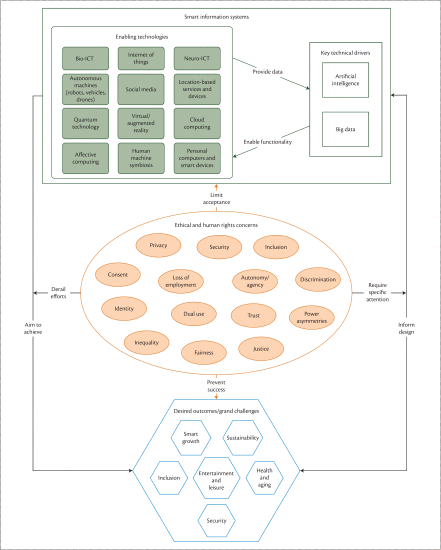
Artificial intelligence (AI) and big data analytics are the key technological drivers of what we call “smart information systems” (SIS). Examples of such intelligent sociotechnical systems abound—Google's search engine, Google Translate, Amazon's recommendation system, Amazon's Alexa home assistant, Facebook's likes, smartphones with GPS tracking, predictive policing systems, automated share dealing, healthcare and surgery robots, personal fitness applications, virtual and augmented reality, and many others, ranging from social network data analysis for advertising to traffic data prediction for energy conservation.

Concerns range from questions of fairness and hidden biases in big data all the way to the possibility of truly autonomous machines that may harm or kill people, but that may also be subjects of ethical rights.

## Smart Information Systems: The Combination of AI and Big Data Analytics

*smart information systems* as a shorthand for technologies that involve artificial intelligence, machine learning, and big data because these are all relevant for the understanding of the social consequences of these technologies.

SIS are formed and developed in an environment of what we call “enabling technologies” that generate and collect data and act on the world and interact with humans. SIS should not be seen in isolation as they tend to be enveloped in the broader technical infrastructure.



## The Ethics of Smart Information Systems

Here we discuss the ethical concerns in more detail as well as current proposals and established mechanisms for addressing them.

### Ethical Issues

privacy and data protection, autonomy of users, their agency, trust, consent, identity, inclusion and digital divides, security, harm, misuse, and deception, to name just a few.

Fairness, Accountability, and Transparency in Machine Learning (FAT ML) forum, which has developed “Principles for Accountable Algorithms and a Social Impact Statement for Algorithms.”

“Ethical training for AI practitioners and students is a necessary part of the solution. Ideally, every student learning AI, computer science, or data science would be exposed to a curriculum and discussion on related ethics and security topics.”

### Current Responses to Ethical Issues

Aimed at the technical level, value-sensitive design**23**, **24** is meant to address ethical questions.

### Combining Viable Approaches: Responsible Research and Innovation

The concept of RRI and the discourses and practices that it has spawned offer a promising and realistic way of dealing with these challenges. RRI is a relatively novel concept that has gained prominence since about 2010. It is an attempt to rethink research and innovation governance with a view to ensuring that processes as well as outcomes of research and innovation are acceptable, desirable, and sustainable.

builds on older streams of activities including technology ethics, technology assessment, science and technology studies, and philosophy of technology.

One key proponent of RRI is the European Commission, which emphasizes six keys of RRI: public engagement, ethics, science education, gender equality, open access, governance.

The debate about the exact definition of RRI and ways to implement it is ongoing and unlikely to lead to consensus anytime soon.

All proponents of RRI agree that stakeholders need to be engaged at an early stage of research and innovation. RRI requires openness and transparency and a willingness to be flexible and responsive to concerns, and needs to be integrated into projects as well as the broader funding and support environment.

## Implementing RRI in SIS Research

one of the 12 HBP subprojects is dedicated to a program of RRI under the heading of ethics and society.

The work in this subproject is divided into four main components: The foresight lab of the HBP investigates possible outcomes and consequences of the work undertaken by the HBP. It is accompanied by a work package that undertakes philosophical research on complex issues, such as the nature of consciousness or simulation. The third major component is public engagement that reaches out to stakeholders and the general public to discuss issues of relevance to the HBP and the public. Finally, the HBP has a work package on ethics management, that is, the administration of ethics-related issues. This includes the support of additional structures that help with ethics-related issues and structures, notably an external Ethics Advisory Board and a network of Ethics Rapporteurs who draw attention to ethics issues in the various subprojects.

## RRI as a Response to Ethics and Privacy in Smart Information Systems

# Reading: [Rethinking Data Privacy: The Impact of Machine Learning](https://medium.com/luminovo/data-privacy-in-machine-learning-a-technical-deep-dive-f7f0365b1d60)

This article discusses the potential for security breaches in personal information collected from social media applications and businesses. As you read, consider the following:

* What are the different dimensions of data sets? How does this affect the ability of data to be anonymized?
* How does machine learning exacerbate the problem of data privacy?
* What are the current trends to help preserve privacy in the context of machine learning?

# Reading: [How Facebook Biases Your News Feed](https://www.forbes.com/sites/nelsongranados/2016/06/30/how-facebook-biases-your-news-feed/)

This article describes how a particular social media site, Facebook, uses algorithms to personalize its news feed. This blog discusses the pros, cons, and biases involved in using personalization algorithms. The purpose of this reading is to help you understand one example of how a social media feed can be affected by algorithms. It is important to understand that these algorithms are constantly changing, which may solve some issues while creating others. As you read, consider the following:

* What does the author suggest as an unbiased method of curating a news feed? Do you agree that this is an unbiased method?
* What are some of the consequences for how these algorithms might shape our perception of the news? What are the societal and ethical implications of this?

# Reading: [The Secretive Company That Might End Privacy as We Know It](https://ezproxy.snhu.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edsbig&AN=edsbig.A611773186&site=eds-live&scope=site)

This reading from the Shapiro Library discusses the ethical and privacy implications of a company called Clearview AI, a facial recognition app. The app has collected approximately three billion publicly available photos and used them to train its algorithms, which then can identify any user uploaded to the app. As you read, consider the following:

* Who developed the app? What were their intentions for how it might be used?
* Whose data is being used for the app? Based on how the data was collected, was Clearview AI given any type of informed consent for the use of their images? Why does this matter?
* How is the app currently being used by private users? By larger entities, such as law enforcement? What ethical implications does this have?

# Additional Support (Optional)

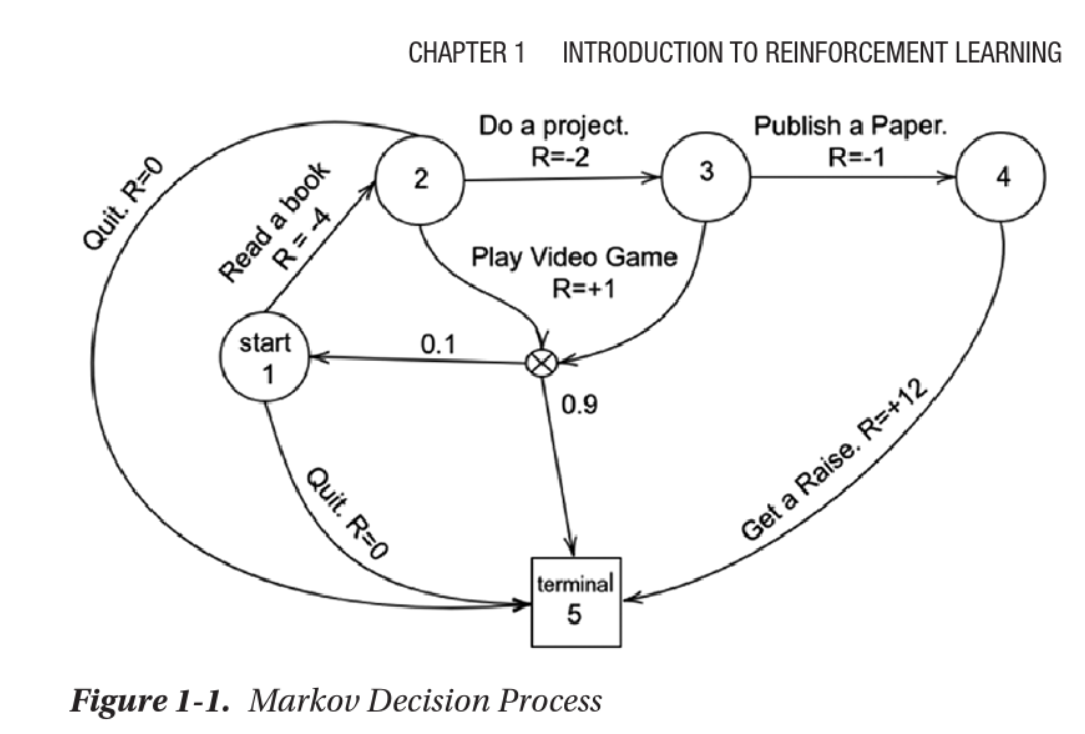
## Reading: [The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks](https://arxiv.org/abs/1802.08232?)

This optional reading from the Shapiro Library discusses how the data sets used to train neural networks often contain sensitive data. This article reviews how this data can be retained by neural networks and used maliciously, and how this can be prevented. As you read, consider the following:

* What type of information does a neural network “memorize”? What are the ethical and privacy implications of this?
* What steps can be taken to mitigate this “memorization”?

# Textbook: [Applied Reinforcement Learning with Python](http://ezproxy.snhu.edu/login?url=https://ebookcentral.proquest.com/lib/snhu-ebooks/detail.action?docID=5880718), Chapters 1 and 2, pp. 1-16, 19-23, 29-36

Chapter 1 of this Shapiro Library textbook will introduce you to the concept of reinforcement learning, including a brief history, its connection to Markov Decision Processes (MDPs), and an introduction to some of the algorithms used in reinforcement learning. In Chapter 2, you will see examples of different reinforcement learning algorithms and how they can be applied. These chapters will help give you an understanding of the concepts which you will be applying in later modules. As you read, consider the following questions:

* What is a Markov Decision Process?
  + We describe MDPs as discrete time stochastic control process. Specifically, we define discrete time stochastic processes as a random process in which the index variable is characterized by a set of discrete, or specific, values (in contrast to continuous values). MDPs are specifically useful for situations in which outcomes are partially affected by participants in the process but the process also exhibits some degree of randomness as well. MDPs and dynamic programming thus become the basis of reinforcement learning theory.
  + 
* What is policy-based learning? What are the different classes of policies?
  + Policy-based gradient methods focus on optimizing the policy function directly rather than trying to learn a value function that would yield information on the expected rewards in a given state. Simply stated, we are selecting an action separately from choosing to utilize a value function. Policies bifurcate into the following classes:
    - Deterministic – A policy that maps a given state to an action(s), specifically where the actions taken “determine” what the outcome will be. For example, you are typing on a keyboard on a word file. When you press “y,” you are certain the character “y” will appear on the screen.
    - Stochastic – A policy that yields a probability distribution over a set of actions, such that there is a probability that the action taken will not be the action that occurs. This is specifically used in instances where the environment is not deterministic and is an example of a partially observable Markov decision process (POMDP).
* What are discounted rewards, why are they important, and how are they used?
  + The reasoning behind discounted rewards is fairly straightforward in that by discounting rewards, we make an otherwise infinite sum finite. If we do not discount rewards, the sum of these rewards would grow infinitely and therefore we would not be able to converge upon an optimal solution.

**NOTE:** There are some complicated math equations referenced in this chapter. Understanding the details of the equations is *not* essential for your work in this course. Be sure to pay special attention to the code sections that are included.

The variables can be broken down as follows:

* Action – Refers to action taken by the agent within an environment that subsequently yields a reward
* Reward – Yielded to the agent. Indicates the quality of action with respect to accomplishing some goal
* Observation – Yielded by the action: Refers to the state of the environment after an action has been performed
* Done – Boolean that indicates whether the environment needs to be reset
* Info – Dictionary with miscellaneous information for debugging

# Reading: [A Brief Introduction to Reinforcement Learning](https://medium.com/free-code-camp/a-brief-introduction-to-reinforcement-learning-7799af5840db)

This reading will provide you with an introduction to the basic concepts behind reinforcement learning and three different kinds of reinforcement learning. Pay special attention to the section on episodic tasks, as this type is what you will be exploring in this course. As you read, consider the following questions:

* What is the basic goal of a reinforcement learning agent?
* What is the difference between exploration and exploitation?

|  |
| --- |
| Additional Support (Optional) |

## Reading: [Math Symbols List](https://www.rapidtables.com/math/symbols/Basic_Math_Symbols.html)

# Q Learning Week 5 Announcement FAQ

**Q-Learning Frequently Asked Questions:**

* **What is the disadvantage of Q-learning?**
  + The learning process in Q-learning can be expensive for the agent, especially in the beginning steps. This is because, to converge to the optimal policy, every state and action pair must be visited frequently.
* **Why is Q learning called Q learning?**
  + In Q-learning, 'Q' stands for quality, representing how useful a given action is in achieving future rewards. This 'Q' is used to create a map system of state and action to maximize expected rewards.
* **Why is Q-Learning off-policy?**
  + Q-learning is off-policy because the updated policy is different from the behavior (action) policy.
* **Does Q-learning always converge?**
  + Yes, during training, the Q-learning algorithm always converges to the optimal policy.
* **Why do we need deep Q-learning?**
  + Q-learning is designed for smaller and discrete environments. In larger environments, an extremely large Q-table is needed, requiring significant memory and computing power to train. Deep Q-learning addresses this by replacing the Q-table with a neural network, allowing it to handle large environments involving continuous action and states.

# Textbook: [Deep Learning with Keras](http://ezproxy.snhu.edu/login?url=https://ebookcentral.proquest.com/lib/snhu-ebooks/detail.action?docID=4850536), Chapter 8

In this reading from the Shapiro Library textbook, you will review the concepts of reinforcement learning, and learn more about Q-learning algorithms and the Keras deep Q-network. This reading does overlap slightly with the chapter in *Applied Reinforcement Learning with Python*, but it is important to read both chapters to have a solid understanding of the different components of Q-learning. The chapter also provides a step-by-step walkthrough and explanation of the code for a “catch” game that applies these algorithms. As you read, consider the following:

* What is Q-learning and how can it be used? How does the Q-learning algorithm work?
  + Deep reinforcement learning utilizes a model-free reinforcement learning technique called Q-learning . Q-learning can be used to find an optimal action for any given state in a finite markov decision process. Q-learning tries to maximize the value of the Q-function which represents the maximum discounted future reward when we perform action a in state s :
  + Once we know the Q-function, the optimal action a at a state s is the one with the highest Q- value. We can then define a policy Ï€(s) that gives us the optimal action at any state:
  + We can define the Q-function for a transition point ( s t , a t , r t , s t+1 ) in terms of the Q-function at the next point ( s t+1 , a t+1 , r t+1 , s t+2 ) similar to how we did with the total discounted future reward. This equation is known as the Bellman equation :
* What is the difference between exploration and exploitation? How does the Epsilon greedy exploration method help to balance exploration with exploitation?
  + In case of ε-greedy exploration, the agent chooses the action suggested by the network with probability 1-ε or an action uniformly at random otherwise. That is why this strategy is called exploration/exploitation.
  + ε-greedy exploration ensures that in the beginning the system balances the unreliable predictions made from the Q-network with completely random moves to explore the state space, and then settles down to less aggressive exploration (and more aggressive exploitation) as the predictions made by the Q-network improve.

initialize Q-table Q

observe initial state s

repeat

select and carry out action a

observe reward r and move to new state s'

Q(s, a) = Q(s, a) + α(r + γ max\_a' Q(s', a') - Q(s, a))

s = s'

until game over

# Textbook: [Applied Reinforcement Learning With Python](http://ezproxy.snhu.edu/login?url=https://ebookcentral.proquest.com/lib/snhu-ebooks/detail.action?docID=5880718), Chapter 3

This chapter from the Shapiro Library textbook will build on the work you did in the previous module with reinforcement learning. You will explore different algorithms used in reinforcement learning, including the Q-learning and deep Q-learning algorithms and their variants. You will also see examples of how these algorithms can be applied to programming problems. As you read, consider the following:

* What is the difference between a Q-matrix and an R-matrix? How are they related?
  + In this algorithm, there are two matrices which we will frequently reference: the Q matrix and the R matrix. The former represents the algorithm’s namesake and contains the accumulated knowledge on the environment in which we are implementing a policy. All of the entries in this matrix are initialized at 0 and the goal is to maximize the reward yielded. Upon each step in the environment, the Q matrix is updated. The R matrix is the environment where each row represents a state and the columns represent the awards for moving to another state. The structure of this matrix is similar to a correlation matrix, where each row and column index mirror one another.
* What are the advantages of Q-learning? What are the disadvantages?
  + The main advantage to Q learning to some degree is that it does not require a model and that the algorithm is fairly transparent. It is easy to explain why the agent at a given state in time will choose an action.
  + the main drawback to this is that the experience necessary to gain knowledge of what to do at a given state is very computationally expensive when we are dealing with very large environments if we are to sufficiently fill the Q matrix with information.
* How does deep Q-learning improve upon Q-learning? What are its limitations?
  + Deep Q Learning is fairly straightforward coming from Q learning In that the only real difference between the two methods is that DQL approximates the values in the Q table rather than trying to populate them manually.
  + Readers should be aware that tasks like these, as we have spoken about due to the preprocessing and computation being utilized, are considerably memory intensive. In addition to this, there are times where the neural network does not learn appropriately the right course of action to take as it gets stuck in local optima. Although the parameters listed have in general yielded out-of-sample solutions that are acceptable, there were also times where this neural network did not perform well. This is one of the limitations.
  + Deep Q Networks often learned very high action values because of overestimation.

**Note**: There are some complicated math equations referenced in this chapter. Understanding the details of the equations is *not* essential for your work in this course. Be sure to pay special attention to the code sections that are included.

# Reading: [An Introduction to Q-Learning: Reinforcement Learning](https://medium.com/free-code-camp/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc)

In this reading, you will see another example of how Q-learning can be applied to a pathfinding problem similar to the one you will explore in Project Two. This reading will help you see a more step-by-step breakdown of what the Q-learning algorithm is doing at each step, and how it is updating the Q-table. As you read, consider the following:

* What are the inputs of the Q-function? What are its returns?
* What are the steps the Q-algorithm is taking? How does the reward mechanism affect this process?

# Reading: [Finding Shortest Path Using Q-Learning Algorithm](https://towardsdatascience.com/finding-shortest-path-using-q-learning-algorithm-1c1f39e89505)

In this reading, you will explore the application of Q-learning to a “shortest path problem”. This scenario differs from the pathfinding problems in the other readings and in Project Two, as the “world” the agent is navigating is not a simple grid. However, the same basic concepts of reward, state, and action apply. As you read, consider the following:

* How is the reward matrix set up? Why are the reward values set the way they are?
* How does the set of steps for the shortest path between 0 and 10 relate to the Q-matrix values?

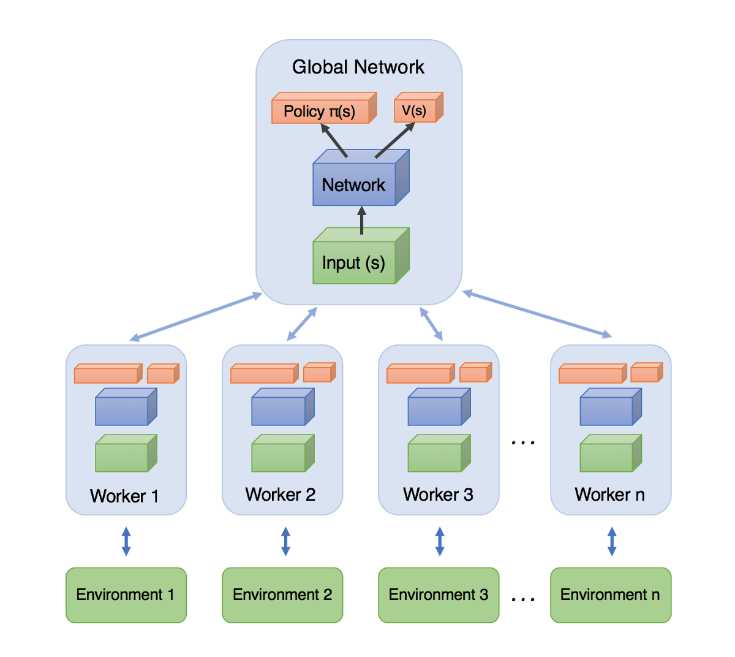
# Reading: [Cartpole - Introduction to Reinforcement Learning (DQN - Deep Q-Learning)](https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288)

This reading explains a solution to a “cartpole problem” using reinforcement learning, specifically deep Q-learning. Though the problem is different than the one you are exploring in Project Two, the type of algorithm (deep Q-learning) is the same. As you read, consider the following:

* What does each line of code do in the cartpole.py file?
* What are the purposes of the “remember” and “experience replay” steps in the algorithm?

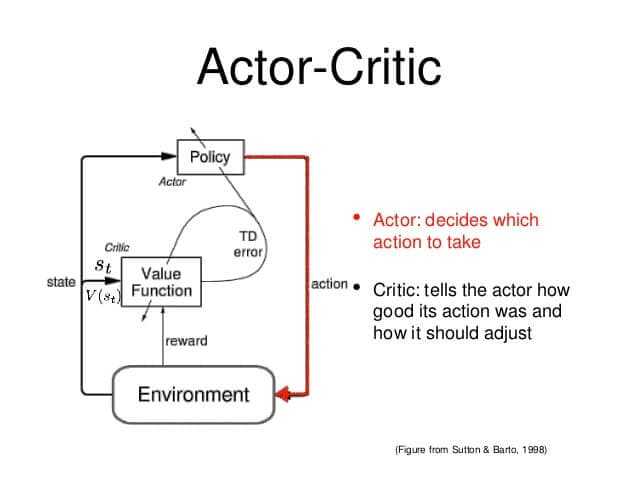
# Reading: [The Idea Behind Actor-Critics and How A2C and A3C Improve Them](https://theaisummer.com/Actor_critics/)

This reading explains the main concepts behind actor-critic algorithms, which are important for reinforcement learning. You will also be exposed to advantage actor-critics (A2C) and asynchronous advantage actor-critics (A3C), which improved upon the original actor-critic model. As you read, consider the following:

* What is the purpose of the actor in the actor-critic model? The critic? How do they work together?
  + The actor critic model is a combination of both the value-based and policy-based algorithms
  + The principal idea is to split the model in two: one for computing an action based on a state and another one to produce the Q values of the action.
  + The actor takes as input the state and outputs the best action. It essentially controls how the agent behaves by **learning the optimal policy** (policy-based). The critic, on the other hand, **evaluates the action by computing the value function** (value based). Those two models participate in a game where they both get better in their own role as the time passes. The result is that the overall architecture will learn to play the game more efficiently than the two methods separately.
* How do the A2C and A3C models improve upon the original actor-critic model? What are the potential disadvantages of these models?
  + Advantage Actor-Critic model (A2C) uses the advantage value.
    - The advantage of the advantage function is that it reduces the high variance of policy networks and stabilize the model.
    - A2C will wait for all the agents to finish their segment and then update the global network weights and reset all the agents.
  + Asynchronous Advantage Actor-Critic (A3C) consists of multiple independent agents with their own weights, who interact with a different copy of the environment in parallel
    - Released by DeepMind
    - Makes DQN obsolete
    - 
    - The agents (or workers) are trained in parallel and update periodically a global network, which holds shared parameters.
    - The main drawback of asynchrony is that some agents will be playing with an older version of the parameters.

But don’t be in a hurry. Let’s refresh for a moment on our previous knowledge. As you may know, there are two main types of RL methods out there:

* Value Based: They try to find or approximate the optimal **value** function, which is a mapping between an action and a value. The higher the value, the better the action. The most famous algorithm is [Q learning](https://theaisummer.com/Deep_Q_Learning/) and all its [enhancements](https://theaisummer.com/Taking_Deep_Q_Networks_a_step_further/) like Deep Q Networks, Double Dueling Q Networks, etc
* Policy-Based: Policy-Based algorithms like [Policy Gradients](https://theaisummer.com/Policy-Gradients/) and REINFORCE try to find the optimal policy directly without the Q -value as a middleman.

A good analogy of the actor-critic is a young boy with his mother. The child (actor) constantly tries new things and exploring the environment around him. He eats its own toys, he touches the hot oven, he bangs his head in the wall (I mean why not). His mother (the critic) watches him and either criticize or compliment him. The child listen to what his mother told him and adjust his behavior. As the kid grows, he learns what actions are bad or good and he essentially learns to play the game called life.

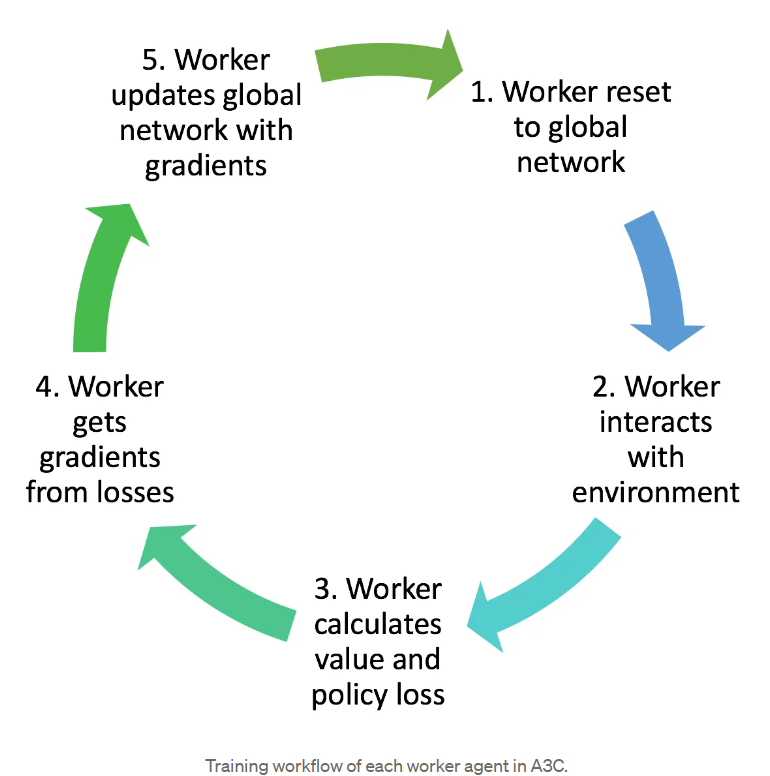
# Reading: [Simple Reinforcement Learning With Tensorflow Part 8: Asynchronous Actor-Critic Agents (A3C)](https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2)

In this reading, you will learn more details about the A3C algorithm developed by Google’s DeepMind. This resource describes the three A’s of A3C (asynchronous, actor-critic, and advantage), connecting them to concepts that you have studied previously, such as the policy gradient. You will also see examples of well-commented code to help you understand the implementation of the A3C algorithms. As you read, consider the following:

* How does the A3C model improve upon Q-learning?
  + It was faster, simpler, more robust, and able to achieve much better scores on the standard battery of Deep RL tasks.
  + Uses multiple environment incarnations to learn more efficiently.
  + The reason this works better than having a single agent (beyond the speedup of getting more work done), is that the experience of each agent is independent of the experience of the others. In this way the overall experience available for training becomes more diverse.
* How does the algorithm implementation work?

In A3C there is a global network, and multiple worker agents which each have their own set of network parameters. Each of these agents interacts with it’s own copy of the environment at the same time as the other agents are interacting with their environments.

The insight of using advantage estimates rather than just discounted returns is to allow the agent to determine not just how good its actions were, but how much better they turned out to be than expected. Intuitively, this allows the algorithm to focus on where the network’s predictions were lacking.



The general outline of the code architecture is:

* **AC\_Network** — This class contains all the Tensorflow ops to create the networks themselves.
* **Worker** — This class contains a copy of AC\_Network, an environment class, as well as all the logic for interacting with the environment, and updating the global network.
* High-level code for establishing the Worker instances and running them in parallel.

# Reading: [Emergent Tool Use From Multi-Agent Interaction](https://openai.com/blog/emergent-tool-use/)

This article describes how multiple agents were trained in an environment to play “hide and seek”. The agents learned how to interact with and manipulate objects in the environment to achieve the goal of winning the game. As you read, consider the following:

* How does the reward structure work in this example? What behavior does this incentivize?
* How does the strategy of each of the different agents adapt as they learn more?
* What is the difference between the “autocurriculum” of these agents and “intrinsic motivation”?

# Video: [Multi-Agent Hide and Seek](https://www.youtube.com/watch?v=kopoLzvh5jY)

This video is an accompaniment to the previous article, though it emphasizes different aspects. In this video, you will watch an example from OpenAI of different agents playing “hide-and-seek”. These agents use reinforcement learning algorithms to interact with their environment, as well as objects within their environment.

* How are the concepts of collaboration and competition exhibited between the different agents in this video? How does this compare to how human beings evolved and developed?
* Based on your knowledge of AI and different RL algorithms, what elements of the algorithms you have learned about do you think are at work here?

# Reading: [Kinds of RL Algorithms](https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#a-taxonomy-of-rl-algorithms)

This reading, from OpenAI’s educational resources, provides a “taxonomy” of reinforcement learning (RL) algorithms to help you see how they are related to one another. You will recognize some algorithms that you have already learned about, as well as additional algorithms that are beyond the scope of this course. As you read, consider the following:

* In this module’s discussion, you will be exploring algorithms like AlphaZero. What makes AlphaZero different from some of the other algorithms you have learned about thus far?
* What is the distinction between policy optimization and Q-learning? What are the strengths and weaknesses of each approach?

# Additional Support (Optional)

**Website**: [OpenAI Resources](https://openai.com/resources/)

**Textbook**: [*Applied Reinforcement Learning With Python*](http://ezproxy.snhu.edu/login?url=https://ebookcentral.proquest.com/lib/snhu-ebooks/detail.action?docID=5880718)Chapter 5, pp. 103-112

# Reading: [Reinforcement Learning, Fast and Slow](https://ezproxy.snhu.edu/login?url=https://dx.doi.org/10.1016/j.tics.2019.02.006)

This Shapiro Library reading will introduce you to two newer techniques in deep reinforcement learning (deep RL): episodic memory and meta-learning. You will learn about the “first wave” of deep RL, where RL algorithms are powered by neural network models, and why this type of learning is considered “slow”. You will also learn about how the emerging trends of episodic memory and meta-learning can potentially speed up deep RL. As you read, consider the following:

* What is sample efficiency? How does it differ for humans versus machines? What are the implications of this for deep RL?
  + Sample efficiency refers to the amount of data required for a learning system to attain any chosen target level of performance.
  + In the initial wave of deep RL systems, they required a lot more training data than humans
  + The implications are that without an increase in training data efficiency, deep RL will be too slow to offer a plausible model for human learning
* What are episodic deep RL and meta-RL, and how do they differ from “slower” deep RL?
  + two of the primary sources of sample inefficiency
    - *incremental parameter adjustment*
      * this form of learning must be small, in order to maximize generalization
    - *weak inductive bias (weak initial assumption)*
      * A learning procedure with weak inductive bias will be able to master a wider range of patterns (greater variance), but will in general be less sample-efficient
  + Episodic Deep RL (solution to incremental parameter adjustment
    - keep an explicit record of past events and use this record directly as a point of reference in making new decisions.
    - parallels ‘**non-parametric**’ approaches in machine learning [[28]](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#bib0140) and resembles ‘instance-’ or ‘exemplar-based’ theories of learning in psychology
    - The fast learning of episodic deep RL depends critically on slow incremental learning.
  + Meta-RL (solution to weak inductive bias)
    - leveraging of past experience to accelerate new learning is referred to in machine learning as meta-learning
    - “Learning to learn”
    - Effectively, one RL algorithm gives birth to another, and hence the moniker ‘meta-RL’.
* What are the implications of this type of “faster” RL? How might this affect AI developments in the near future?
  + The implication of faster RL is that it shows that these models can be used to model human and animal thinking by disproving the main counter argument
  + this will create a focus on using the fast learning enabled by slow learning in AI development in the near future and will likely speed up the progress made by AI

]

the concern has been raised that deep RL may be too sample-inefficient – that is, it may simply be too slow – to provide a plausible model of how humans learn.

deep RL marries neural network modeling with reinforcement learning

Deep Reinforcement Learning

RL centers on the problem of learning a behavioral policy, a mapping from states or situations to actions, which maximizes cumulative long-term reward [[12]](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#bib0060). In simple settings, the policy can be represented as a look-up table, listing the appropriate action for any state. In richer environments, however, this kind of simple listing is infeasible, and the policy must instead be encoded implicitly as a parameterized function. Pioneering work in the 1990s showed how this function could be approximated using a multilayer (or deep) neural network ([[78]](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#bib0390), L.J. Lin, PhD Thesis, Carnegie Melon University, 1993), allowing gradient-descent learning to discover rich, nonlinear mappings from perceptual inputs to actions (see panel A and below). However, technical challenges prevented the integration of deep neural networks with RL until 2015, when lll}breakthrough work demonstrated how deep RL could be made to work in complex d\]

mains such as Atari video games [[13]](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#bib0065) (see [Figure I](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#fig0010)B and below). Since then, rapid progress has been mad/

e toward improving and scaling deep RL [[79]](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#bib0395), allowing its application to complex task klkdomains such as Go [[16]](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#bib0080) and Capture the Flag [[80]](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#bib0400). In many cases, the later advances have involved integrating deep RL with architectural and algorithmic complements, such as tree search [[16]](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#bib0080) or slot-based, episodic-like memory [[52]](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#bib0260) (see [Figure I](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#fig0010)C and below). Other developments have focused on the goal of learning speed, allowing deep RL to make progress based on just a few observations, as reviewed in the main text.

The figure illustrates the evolution of deep RL methods, starting in panel A with Tesauro's groundbreaking backgammon-playing system ‘TD-gammon’ [[78]](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#bib0390). This centered on a neural network which took as input a representation of the board and learned to output an estimate of the ‘state value,’ defined as expected cumulative future rewards, here equal simply to the estimated probability of eventually winning a game from the current position. Panel B shows the Atari-playing DQN network reported by Mnih and colleagues [[13]](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#bib0065). Here, a convolutional neural network (see [[3]](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#bib0015)) takes screen pixels as input and learns to output joystick actions. Panel C shows a schematic representation of a state-of-the art deep RL system reported by Wayne and colleagues [[51]](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#bib0255). A full description of the detailed ‘wiring’ of this RL agent is beyond the scope of the present paper (but can be found in [[51]](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#bib0255)). However, as the figure indicates, the architecture comprises multiple modules, including a neural network that leverages an episodic-like memory to predict upcoming events, which 'speaks’ to a reinforcement-learning module that selects actions based on the predictor module's current state. The system learns, among other tasks, to perform goal-directed navigation in maze-like environments, as shown in [Figure I](https://www-sciencedirect-com.ezproxy.snhu.edu/science/article/pii/S1364661319300610?via%3Dihub#fig0010).

# Website: [Open AI: Progress](https://openai.com/progress/)

This website from OpenAI has two distinct sections: milestone releases and research papers. Milestone releases exhibit advancements that the OpenAI team has recently developed so that they can be explored and tested right away. The research papers are more formal academic research that OpenAI publishes on their developments. Explore several different milestones and papers related to the topics you’ve learned about in this class, or browse other titles that seem interesting.

# Website: [DeepMind: Research](https://deepmind.com/research)

This website from Google’s DeepMind includes links to different publications and blog posts based on their current research projects. Use this website to explore different articles related to the topics that you’ve learned about in this class, or browse other titles that seem interesting.

# Website: [Pew Research Center: Internet and Technology](https://www.pewinternet.org/)

This website from the Pew Research Center contains links to recent articles about the internet and technology. Explore this website to learn about newsworthy emerging trends in AI, and to find an article for use in your discussion. Use the search bar to look for recent articles on specific topics such as neural networks or reinforcement learning that you’ve learned about in this course.

|  |
| --- |
| Additional Support (Optional) **Reading**: [CS Program GitHub Portfolio Tutorial](https://learn.snhu.edu/d2l/lor/viewer/viewFile.d2lfile/1536117/24276,-1/) |

This tutorial walks you step-by-step through setting up your account, setting up your repository, adding collaborators, uploading files, and creating your README file. Reference this tutorial as needed for the assignments in this module.