Chapter 1. The Machine Learning Landscape

1. How would you define Machine Learning?
2. Can you name four types of problems where it shines?
3. What is a labeled training set?
4. What are the two most common supervised tasks?
5. Can you name four common unsupervised tasks?
6. What type of Machine Learning algorithm would you use to allow a robot to walk in various unknown terrains?
7. What type of algorithm would you use to segment your customers into multiple groups?
8. Would you frame the problem of spam detection as a supervised learning problem or an unsupervised learning problem?
9. What is an online learning system?
10. What is out-of-core learning?
11. What type of learning algorithm relies on a similarity measure to make predictions?
12. What is the difference between a model parameter and a learning algorithm's hyperparameter?
13. What do model-based learning algorithms search for? What is the most common strategy they use to succeed? How do they make predictions?
14. Can you name four of the main challenges in Machine Learning?
15. If your model performs great on the training data but generalizes poorly to new instances, what is happening? Can you name three possible solutions?
16. What is a test set and why would you want to use it?
17. What is the purpose of a validation set?
18. What can go wrong if you tune hyperparameters using the test set?
19. What is cross-validation and why would you prefer it to a validation set?

Chapter 2. End-to-End Machine Learning Project

In this chapter, you will go through an example project end to end, pretending to be a recently hired data scientist in a real estate company.1 Here are the main steps you will go through:

1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data to gain insights.
4. Prepare the data for Machine Learning algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.
8. Launch, monitor, and maintain your system.

Chapter 3. Classification

1. Try to build a classifier for the MNIST dataset that achieves over 97% accuracy on the test set. Hint: the KNeighborsClassifier works quite well for this task; you just need to find good hyperparameter values (try a grid search on the weights and n_neighbors hyperparameters.
2. Write a function that can shift an MNIST image in any direction (left, right, up, or down) by one pixel. Hint: You can use the shift() function from the scipy.ndimage.interpolation module. For example, shift(image, [2,1], cval=0) shifts the image 2 pixels down and 1 pixel to the right. Then, for each image in the training set, create four shifted copies (one per direction) and add them to the training set. Finally, train your best model on this expanded training set and measure its accuracy on the test set. You should observe that your model performs even better now. This technique of artificially growing the dataset is referred to as dataset augmentation or training set expansion.
3. Tackle the Titanic dataset from Kaggle.
4. Build a spam classifier:
    - Download examples from Apache SpamAssassin dataset
    - Unzip and familiarize yourself with the dataset
    - Split the datasets into training and test set
    - Write a data preparation pipeline to convert each email into a feature vector. Your pipeline should transform an email into a (sparse) vector indicating the presence or absence of each possible word. For example, if all emails only ever contain the words ('Hello', 'how', 'are', 'you') then the email 'Hello you Hello Hello you' would be converted to the vector [1,0,0,1] indicating the words that are present. Alternatively, [3,0,0,2] is also acceptable and represents a count of word occurrences in the sentence.
    - You may want to add hyperparameters to your preparation pipeline to control whether or not the to strip off email headers, convert each email to lowercase, remove punctuation, replace all URLs with 'URL', replace all numbers with 'NUMBER', or even perform stemming (there are python libraries you can utilize for this exercise).
    - Try out several classifiers and see if you can build a spam classifier with both high recall and high precision.

Chapter 4. Training Models

1. What Linear Regression training algorithm can you use if you have a training set with millions of features?
2. Suppose the features in your training set have very different scales. What algorithms might suffer from this, and how? What can you do about it?
3. Can Gradient Descent get stuck in a local minimum when training a Logistic Regression model?
4. Do all Gradient Descent algorithms lead to the same model provided you let them run long enough?
5. Suppose you use Batch Gradient Descent and you plot the validation error at every epoch. If you notice that the validation error consistently goes up, what is likely going on? How can you fix this?
6. Is it a good idea to stop Mini-batch Gradient Descent immediately when the validation error goes up?
7. Which Gradient Descent algorithm (among those we discussed) will reach the vicinity of the optimal solution the fastest? Which will actually converge? How can you make the others converge as well?
8. Suppose you are using Polynomial Regression. You plot the learning curves and you notice that there is a large gap between the training error and the validation error. What is happening? What are three ways to solve this?
9. Suppose you are using Ridge Regression and you notice that the training error and the validation error are almost equal and fairly high. Would you say that the model suffers from high bias or high variance? Should you increase the regularization hyperparameter α or reduce it?
10. Why would you want to use:
    - Ridge Regression instead of Linear Regression?
    - Lasso instead of Ridge Regression?
    - Elastic Net instead of Lasso?
11. Suppose you want to classify pictures as outdoor/indoor and daytime/nighttime. Should you implement two Logistic Regression classifiers or one Softmax Regression classifier?
12. Implement Batch Gradient Descent with early stopping for Softmax Regression (without using Scikit-Learn).

Chapter 5. Support Vector Machines

1. What is the fundamental idea behind SVMs?
2. What is a support vector?
3. Why is it important to scale the inputs when using SVM?
4. Can an SVM output a confidence score when it classifies an instance? What about a probability?
5. Should you use the primal or the dual form of the SVM problem to train a model on a training set with millions of instances and hundreds of features?
6. Say you trained an SVM classifier with an RBF kernel. It seems to underfit the training set: should you increase or decrease γ (gamma)? What about C?
7. How should you set the QP parameters (H, f, A, b) to solve the soft margin linear SVM classifier problem using an off-the-shelf QP solver?
8. Train a LinearSVC on a linearly separable dataset. Then train an SVC and an SGDClassifier on the same dataset. See if you can get them to produce roughly the same model.
9. Train an SVM classifier on the MNIST dataset. Since the SVM classifiers are binary classifiers, you will need to use one-versus-all to classify all 10 digits. You may want to tune the hyperparameters using small validation sets to speed up the process. What accuracy can you reach?
10. Train an SVM regressor on the California housing dataset.

Chapter 6. Decision Trees

1. what is the approximate depth of a Decision Tree (DT) trained without restriction on a training set with 1,000,000 instances?
2. Is a node's Gini impurity generally less than or greater than its parent's? Is generally this way or always this way?
3. If a Decision Tree is overfitting a dataset is it a good idea to try decreasing max_depth?
4. If a Decision Tree is underfitting is it a good idea to scale input features?
5. If it takes one hour to train a DT on a training set containing 1 million instances, roughly how long will it take to train another DT on a dataset containing 10 million instances?
6. If the training set contains 100,000 instances will setting presort=True speed up training?
7. Train and fine-tune a DT for the moons dataset.
    - Generate a moons dataset using make_moons(n_samples=10000, noise=0.4).
    - Split it into a training set and a s test set using train_test_split().
    - Use grid search with cross-validation (with the help of GridSearchCV class) to find good hyperparameter values for a DecisionTreeClassifier. Hint: try various values for max_leaf_nodes.
    - Train it on the full training set using these hyperparameters and measure your model's performance on the test set. You should achieve roughly 85% to 87% accuracy.
8. Grow a forest.
    - Continuing the previous exercise, generate 1000 subsets of the training set in which each contains 100 instances selected randomly. Hint: you can use Scikit-Learn's ShuffleSplit class for this.
    - Train one DT on each subset using the best hyperparameters found above. Evaluate these 1000 DTs on the test set. Since they were trained on smaller sets they will likely perform worse than the first decision tree thus only achieving about 80% accuracy.
    - Now comes the magic. For each test set instance generate the predictions of the 1000 DTs and keep only the most frequent prediction (use SciPy's mode() function for this) in order to obtain the majority-vote predictions over the test set.

Chapter 7. Ensemble Learning and Random Forests

1. Assume you have trained five different models on the exact same training data and they all achieve 95% precision. Is it possible to combine these models to get better results? If so, how? If not, why?
2. Explain the difference between hard and soft voting classifiers.
3. Is it possible to speed up training of a bagging ensemble by distributing it across multiple servers? What about pasting ensembles, boosting ensembles, random forest, or stacking ensembles?
4. What is the benefit of out-of-bag evaluation?
5. What makes Extra-Trees more random than regular Random Forests? How can this extra randomness help? Are Extra-Trees slower or faster than regular Random Forests?
6. What hyperparameters should you alter (and in what way and why) if your AdaBoost ensemble underfits the training data?
7. Should you increase or decrease the learning rate if your Gradient Boosting ensemble overfits the training set? Why? How do you do it?
8. Load the MNIST data (chapter 3) and split it into training, validation, and test sets (e.g. 40,000 training, 10,000 validation, 10,000 testing). Then train various classifiers (Random Forest, Extra-Trees, SVM, etc.). Next, try to combine them into an ensemble classifier that outperforms each individual classifier on the validation set using a hard or soft voting classifier. Once you have found one try it on the test set. How much better does it perform compared to the individual classifiers?
9. Run the individual classifiers from the previous exercise to make predictions on the validation set, and create a new training set with the resulting predictions: each training instance is a vector containing the set of predictions from all your classifiers for an image, and the target is the image's class. Congratulations, you have just trained a blender, and together with the classifiers they form a stacking ensemble! Now let's evaluate the ensemble on the test set. For each image in the test set make predictions with all your classifiers and then feed the predictions to the blender to get the ensemble's predictions. How does it compare to the voting classifier you trained earlier?

Chapter 8. Dimensionality Reduction

1. What are the main motivations when reducing a dataset's dimensionality? Explain the advantages as well as any disadvantages. When should you use this technique and when should it be avoided (if ever)?
2. What is the curse of dimensionality?
3. Is it possible to reverse a reduction in a dataset's dimensionality? If so, how? If not, why?
4. Can PCA be used to reduce the dimensionality of a highly nonlinear dataset?
5. Suppose you perform PCA on a 1,000-dimensional dataset, setting the explained variance ratio to 95%. How many dimensions will the resulting dataset have?
6. In what cases would you use vanilla PCA, Incremental PCA, Randomized PCA, or Kernel PCA?
7. How can you evaluate performance of a dimensionality reduction algorithm on your dataset?
8. Does it make any sense to chain two different dimensionality reduction algorithms?
9. Load the MNIST dataset (chapter 3) and split it into Train (60,000 instances) and Test (10,000 instances). Train a Random Forest classifier on the dataset and time ohow long it takes, then evaluate the resulting model on the test set. Next, use PCA to reduce the dataset's dimensionality with an explained variance of 95%. Train a new Random Forest classifier on the reduced dataset and see how long it takes to complete. Was training faster? Why or why wouldn't you expect it to be? How does a Random Forest classifier work? Finally, evaluate the classifier on the test set: how does it compare to the previous classifier?
10. Use t-SNE to reduce the MNIST dataset down to two dimensions and plot the result using Matplotlib. You can use a scatterplot with 10 different colors to represent each image's target class. You could alternatively write colored digits at the location of each instance, or even plot scaled down versions of the digit images themselves (choose a subset of images as plotting all will quickly become cluttered). You should end up with a nice visualization with well separated clusters of digits. Try using other dimensionality reduction algorithms such as PCA, LLE, or MDS and compare the resulting visualizations.

Chapter 9. Up and Running with TensorFlow

1. What are the main benefits of creating a computation graph rather than directly executing the computations? Explain what a computation graph is with examples.
2. Is the statement a_val = a.eval(session=sess) equivalent to a_val = sess.run(a)? Explain what each does, highlighting similarities and differences between them.
3. Is the statement a_val, b_val = a.eval(session=sess), b.eval(session=sess) equivalent to a_val, b_val = sess.run([a,b])?
4. Can you run two graphs simultaneously?
5. If you create a graph g containing a variable w, then start two threads and open a session in each thread in which both are using graph g, will each session have its own copy of the variable w or will it be shared?
6. When is a variable initialized? When is it destroyed?
7. What is the difference between a placeholder and a variable?
8. What happens when you run the graph to evaluate an operation that depends on a placeholder but you don't feed its value? What happens if the operation does not depend on the placeholder?
9. Can you feed the values of any operation when you run a graph, or just the value of the placeholders?
10. How can you set a variable to any value you want (during the execution phase)?
11. How many times does reverse-mode autodiff need to traverse the graph in order to compute the gradients of the cost function with regards to 10 variables? What about forward-mode autodiff? And symbolic differentiation?
12. Implement Logistic Regression and Mini-batch Gradient Descent using TF. Train and evaluate a model on the moons dataset from chapter five and try adding all the bells and whistles:
    - Define the graph within a logistic_regression() function that can be reused.
    - Save checkpoints using a Saver at regular intervals during training and save the final model at the end of the training.
    - Restore the last checkpoint upon startup if training was interrupted.
    - Define the graph using nice scopes so the graph looks good in TensorBoard.
    - Add summaries to visualize learning curves in TensorBoard.
    - Tweak some hyperparameters such as the learning rate or the mini-batch size and look at the shape of the learning curve.

Chapter 10. Introduction to Artificial Neural Networks

1. Draw an ANN using the original artificial neurons (like the ones in figure 10-3) that computes $A \oplus B$ $(where \ \oplus \ represents \ the \ XOR \ operation)$. Hint: $A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$. (todo: explain what these mean)
2. Why is it generally preferable to use a Logistic Regression classifier rather than a classical Perceptron (i.e., a single layer of linear threshold units trained using the Perceptron training algorithm)? How can a Perceptron be tweaked to make it equivalent to a Logistic Regression classifier?
3. Why was the logistic activation function a key ingredient in training MLPs?
4. Name three common activation functions and describe their properties and strengths and weaknesses. Sketch them.
5. Suppose you have an MLP composed of one input layer with 10 passthrough neurons, followed by one hidden layer with 50 artificial neurons, and finally one output layer with 3 artificial neurons. All artificial neurons use the ReLU activation function.
    - What is the shape of the input matrix **X**?
    - What about the shape of the hidden layer's weight vector $\mathbf{W}_h$, and the shape of its bias vector $\mathbf{b}_h$?
    - What is the shape of the output layer's weight vector $\mathbf{W}_0$, and it's bias vector $\mathbf{b}_0$?
    - What is the shape of the network's output matrix **Y**?
    - Write the equation that computes the network's output matrix Y as a function of **X**, $\mathbf{W}_h$, $\mathbf{b}_h$, $\mathbf{W}_0$, and $\mathbf{b}_0$.
6. How many neurons do you need in the output layer if you want to classify email into spam or ham? What activation function should you use in the output layer? If you wanted to tackle MNIST instead how many neurons do you need in the output layer? Which activation function should you use? What if you wanted a network to predict housing prices from chapter 2?
7. What is backpropagation and how does it work? What is the difference between backprop and reverse-mode autodiff? Explain how reverse-mode autodiff works.
8. Can you list all the hyperparameters you can modify in MLP? If the MLP overfits the training data how would you tweak these hyperparameters to try and solve the problem?

9. Train a deep MLP on the MNIST dataset and see if you can get over 98% precision. Just like in the last exercise of chapter 9, try adding other parameters (i.e., save checkpoints, restore the last checkpoint in case of interruption, add summaries, plot learning curves using TensorBoard, etc).

Chapter 11. Training Deep Neural Nets

1. Is it alright to initialize all weights to the same value as long as that value is selected randomly using He initialization?
2. Is it ok to initialize the bias terms to zero? Why or why not? What happens if you do this? (try it!)
3. Name three advantages to the ELU activation function over ReLU. Are there any disadvantages?
4. In which cases would you want to use each of the following activation functions:
    - ELU
    - Leaky ReLU (and its variants)
    - ReLU
    - Tanh
    - Logistic
    - Softmax
5. What may happen if the momentum hyperparameter is set too close to 1 (e.g., 0.99999) when using the MomentumOptimizer?
6. Name three ways you can produce a sparse model.
7. Does dropout slow down training? Does it slow down inference (i.e., making predictions on new instances)?
8. Deep Learning
    - Build a DNN with five hidden layers of 100 neurons each, He initialization, and the ELU activation function.
    - Using Adam optimization and early stopping, try training on MNIST but only on digits 0 to 4, as we will use transfer learning for digits 5 to 9 in the next exercise. You will need a softmax output layer with five neurons, and as always make sure to save checkpoints at regular intervals and save the final model so you can reuse it later.
    - Tune the hyperparameters using cross-validation and see what precision you achieve.
    - Try adding Batch Normalization and compare the learning curves: is it converging faster than before? Does it produce a better model?
    - Is the model overfitting the training set? Try adding dropout to each layer. Does it help?
9. Transfer Learning

- Create a new DNN that reuses all the pretrained hidden layers of the previous model, freezes them, and replaces the softmax output layer with a new one.
- Train this new DNN on digits 5 to 9, using only 100 images per digit, and time how long it takes. Is it possible to obtain high precision with this model despite the small sample size?
- Try caching the frozen layers and train the model again: how much faster is it now?
- Try again reusing just four hidden layers instead of five. Can you achieve a higher precision result?
- Unfreeze the top two hidden layers and continue training: can you get the new model to perform even better?

10. Pretraining on an auxiliary task
    - Build a DNN that compares two MNIST digit images and predicts whether they represent the same digit or not. Then you will reuse the lower layers of this network to train an MNIST classifier using very little training data. Start by building two DNNs (name them DNN A and B), both similar to the one you built earlier but without the output layer: each DNN should have five hidden layers with 100 neurons each, He initialization, and ELU activation. Next, add a single output layer on top of both DNNs. You should use TFs concat() function with axis=1 to concatenate the outputs of both DNNs along the horizontal axis, then feed the result to the output layer. This output layer should contain a single neuron using the logistic activation function.
    - Split the MNIST dataset into two sets: split #1 should contain 55,000 images, and split #2 should contain 5,000 images. Create a function that generates a training batch where each instance is a pair of MNIST images picked from split #1. Half of the training instances should be pairs of images that belong to the same class, while the other half should be images from different classes. For each pair the training label should be 0 if the images are from the same class and 1 if they are different.
    - Train the DNN on this training set. For each image pair you can simultaneously feed the first image to DNN A and the second image to DNN B. The whole network will gradually learn to tell if the images belong to the same class or not.
    - Create a new DNN by reusing and freezing the hidden layers of DNN A and adding a softmax output layer with 10 neurons. Train this network on split #2 and see if you can achieve high performance despite having only 500 images per class.

Chapter 12. Distributing TensorFlow Across Devices and Servers

1. If you get a CUDA_ERROR_OUT_OF_MEMORY when starting TF, what is probably causing it? How can you fix it?
2. What is the difference between pinning and operation on a device and placing an operation on a device?
3. If you are running on a GPU-enabled TF installation using the default placement, will all operations be placed on the first gpu?
4. If you pin a variable to “/gpu:0” can it be used by operations placed on /gpu:1? What about operations placed on “/cpu:0”? How about operations pinned to devices located on other servers?
5. Can two operations placed on the same device run in parallel?
6. What is a control dependency and when would you want to use one?
7. Suppose you train a DNN for days on a TF cluster and immediately after your training program ends you realize that you forgot to save the model using a Saver. Is your model lost?
8. Train several DNNs in parallel on a TF cluster using different hyperparameter values. This could be DNNs for MNIST classification or any other task that interests you. The simplest option is to write a single client program that trains only one DNN then run this program in multiple processes in parallel with different hyperparameter values for each client. The program should have command-line options to control what server and device the DNN should be place on, and what resource container and hyperparameter values to use (make sure to use a different resource container for each DNN). Use a validation set or cross-validation to select the top three models
9. Create and ensemble using the top three models from the previous exercise. Define it in a single graph ensuring that each DNN runs on a different device. Evaluate it on the validation set: does the ensemble perform better than the individual DNNs?
10. Train the DNN using between-graph replication and data parallelism with asynchronous updates, timing how long it takes to reach a level of satisfying performance. Try again using synchronous updates to see if they produce a better model. Is training faster? Split the DNN vertically and place each vertical slice on a different device and train the model again. Is training faster? Is performance different in any other way?

Chapter 13. Convolutional Neural Networks

1. What are the advantages of a CNN over fully connected DNN for image classification?
2. Consider a CNN composed of three convolutional layers, each with 3x3 kernels, a stride of 2, and SAME padding. The lowest layer outputs 100 feature maps, the middle outputs 200, and the top outputs 400. The input images are RGB images of 200x300 pixels. What is the total number of parameters in the CNN? If we are using 32-bit floats approximately how much RAM will the network require when making a prediction for a single instance. What about when training on a mini-batch of 50 images?
3. What are five things you could do while attempting to solve the problem of your gpu running out of memory while training a CNN?
4. Why would you want to add a max pooling layer rather than a convolutional layer with the same stride?
5. When would you want to add a local response normalization layer?
6. Can you name the main innovations in AlexNet, compared to LeNet-5? What about the main innovations in GoogLeNet and ResNet?
7. Build your own CNN and try to achieve the highest possible accuracy on MNIST.
8. Classifying large images using Inception v3.
    - Download some images of various animals. Load them in python, for example using matplotlib.image.mpimg.imread() function or the scipy.misc.imread() function. Resize or crop them to 299x299 pixels and ensure that they have three channels (RGB) with no transparency channel.
    - Download the latest pretrained Inception v3 modelL the checkpoint is available at https://goo.gl/nxSQvl
    - Create the Inception v3 model by calling the inception_v3() function below. This must be done with an argument scope created by inception_v3_arg_scope() function. Also, you must set is_training=False and num_classes=1001 like so:

```
import tensorflow as tf
from tensorflow.contrib.slim.nets import inception
import tensorflow.contrib.slim as slim
```

```
            X = tf.placeholder(tf.float32, shape=[None,
299,299,3],name="X")
            with slim.arg_scope(inception.inception_v3_arg_scope()):
                  logits, end_points = inception.inception_v3(
                        X, num_classes=1001, is_training=False)
            predictions = end_points["Predictions"]
            saver = tf.train.Saver()
```

- Open a session and use the Saver to restore the pretrained model checkpoint you downloaded earlier.
- Run the model to classify the images you prepared. Display the top five predictions for each image, along with the estimated probability (the list of class name is available at https://goo.gl/brXRtZ). How accurate is the model?

9. Transfer learning for large image classification.
    - Create a training set containing at least 100 images per class. For example, you could classify your own pictures based on the location (beach, mountain, city, etc.), or alternatively use an existing dataset such as the flowers dataset (https://goo.gl/EgJVXZ) or MIT's places dataset (http://places.csail.mit.edu/ (requires registration and the dataset is huge)).
    - Write a preprocessing step that will resize and crop the images to 299x299, with some randomness for data augmentation.
    - Use the pretrained Inception v3 model from the previous exercise, freeze all layers up to the bottleneck layer (i.e., the last layer before the output layer), and replace the output layer with the appropriate number of outputs for your new classification task (e.g., the flowers dataset has five mutually exclusive classes so the output layer must have five neurons and use the softmax activation function).
    - Split your dataset into a training and test set. Train the model on the training set and evaluate it on the test set.
10. Go through the TF DeepDream tutorial (https://goo.gl/4b2s6g). This serves as an engaging exercise to familiarize yourself with various ways of visualizing the patterns learned by a CNN, and to generate art using Deep Learning.

Chapter 14. Recurrent Neural Networks

1. Write down a few potential applications for a sequence-to-sequence RNN. What about a sequence-to-vector RNN? Vector-to-sequence? Define each.
2. Why do people use encoder-decoder RNNs rather than plain sequence-to-sequence RNNs for automatic translation?
3. How could you combine a CNN with an RNN to classify video?
4. What are the advantages of building an RNN using dynamic_rnn() rather than static_rnn()?
5. How can you deal with variable-length input sequences? What about variable-length output sequences?
6. What is a common way to distribute training and execution of a deep RNN across multiple GPUs?
7. Embedded Reber grammars were used by Hochreiter and Schmidhuber in their paper on LSTMs. These artificial grammars produce strings such as “BPBTSXXVPSEPE”. See Jenny Orr’s intro to this topic. Choose a particular embedded Reber grammar (such as the one represented on Orr’s page), then train an RNN to identify whether a string respects that grammar or not. You will first need to write a function capable of generating a training batch containing about 50% strings that respect the grammar, and 50% that don’t.
8. Tackle the “How much did it rain? II” Kaggle competition. This is a time series prediction task: you are given snapshots of polarimetric radar values and asked to predict the hourly rain gauge total. Luis Andre Dutra e Silva’s interview (https://goo.gl/fTA90W) gives some insights into the the techniques he used to to reach second place in the competition. In particular he used and RNN composed of two LSTM layers.
9. Go through TF’s Word2Vec tutorial to create word embeddings, then go through Seq2Seq tutorial to train an English-to-French translation system

Chapter 15. Autoencoders

1. What are the main tasks that autoencoders are used for? Define what an autoencoder is.
2. Suppose you want to train a classifier and you have plenty of unlabeled training data, but only a few thousand labeled instances. How can autoencoders help? How would you proceed?
3. If an autoencoder perfectly reconstructs the inputs, is it necessarily a good autoencoder? How can you evaluate its performance?
4. What are undercomplete and overcomplete autoencoders? What is the main risk of an excessively undercomplete autoencoder? Overcomplete?
5. How do you tie weights in a stacked autoencoder? What is the point of doing so?
6. What is a common technique to visualize features learned by the lower layer of a stacked autoencoder? What about higher layers?
7. What is a generative model? Name and define types of generative autoencoders.
8. Let's use a denoising autoencoder to pretrain an image classifier:
    - You can use MNIST (easiest) or other large datasets (CIFAR10) if you want a bigger challenge. If you choose CIFAR10 you need to write code to load batches of images for training. If you want to skip this part TF's model zoo contains tools to do just that.
    - Split the dataset into training and test. Train a deep denoising autoencoder on the full training set.
    - Check that the images are fairly well reconstructed and visualize the low level features. Visualize the images that most activate each neuron in the coding layer.
    - Build a classification deep neural network reusing the lower layer of the autoencoder. Train it using only 10% of the training set. Can you get it to perform as well as the same classifier trained on the full training set?

9. Semantic hashing is a technique used for efficient information retrieval: a document (e.g., an image) is passed through a system, typically a neural network, which outputs a fairly low-dimensional binary vector (e.g., 30 bits). Two similar documents are likely to have identical or very similar hashes. By indexing each document by its hash it is possible to retrieve many documents similar to a particular document almost instantly even if there are billions of documents: just compute the hash of the document and look up all documents with the same

hash (or hashes that differ by just one or two (how many can you be off by before relevance drops significantly?)). Implement semantic hashing using a slightly tweaked stacked autoencoder:

- Create a stacked autoencoder containing two hidden layers below the coding layer, and train it on the image dataset you used in the previous exercise the coding layer should contain 30 neurons and use the logistic activation function to output values between 0 and 1. After training, to produce the hash of an image, you can simply run it through the autoencoder, take the output of the coding layer, and round every value to the closest integer (0 or 1).
- One neat trick proposed by the researchers is to add Gaussian noise with zero mean to the inputs of the coding layer during training only. In order to preserve a high signal-to-noise ratio the autoencoder will learn to feed large values to the coding layer so that the noise becomes negligible. In turn this means that the logistic function of the coding layer will likely saturate at 0 or 1, so rounding them will not distort them too much and will improve the reliability of the hashes.
- Compute the hash of every image and see if the images with identical hashes look alike. Since MNIST and CIFAR10 are labeled a more objective way to measure the performance of the autoencoder for semantic hashing is to ensure that images with the same hash generally have the same class. One way to do this is to measure the average Gini purity (chapter 6) of the set of images with identical (or very similar) hashes.
- Try fine-tuning the hyperparameters using cross-validation.
- Note that with labeled datasets another approach is to train a CNN (chapter 13) for classification, then use the layer below the output layer to produce the hashes. See Jinma Gua and Jianmin Li's 2015 paper for details. See if that performs better.

10. Train a variational autoencoder on the image dataset used in the previous exercises and make it generate images. You can also find an unlabeled dataset that you are interested in and see if you can generate new samples.

Chapter 16. Reinforcement Learning

1. How would you define reinforcement learning? How is it different from regular supervised and unsupervised learning?
2. Can you think of three possible applications of RL that were not mentioned in this chapter? For each of them, what is the environment? What is the agent? What are the possible actions? What are the rewards?
3. What is the discount rate? Can the optimal policy change if you modify the discount rate?
4. How do you measure performance of a RL agent?
5. What is the credit assignment problem? When does it occur? How can you alleviate it?
6. What is the point of using a replay memory?
7. What is an off-policy RL algorithm?
8. Use Deep Q-Learning to tackle OpenAI gym's "BypedalWalker-v2". The Q-networks do not need to be very deep for this task.
9. Use policy gradients to train an agent to play Pong. Beware: an individual observation is insufficient to tell the direction and speed of the ball. One solution is to pass two observations to at a time to the NN policy. To reduce dimensionality and speed up training you should preprocess these images (crop, resize, convert to b&w) and possibly merge them into a single image (e.g., by overlaying them).
10. Build cool shit with everything you have learned up to this point.