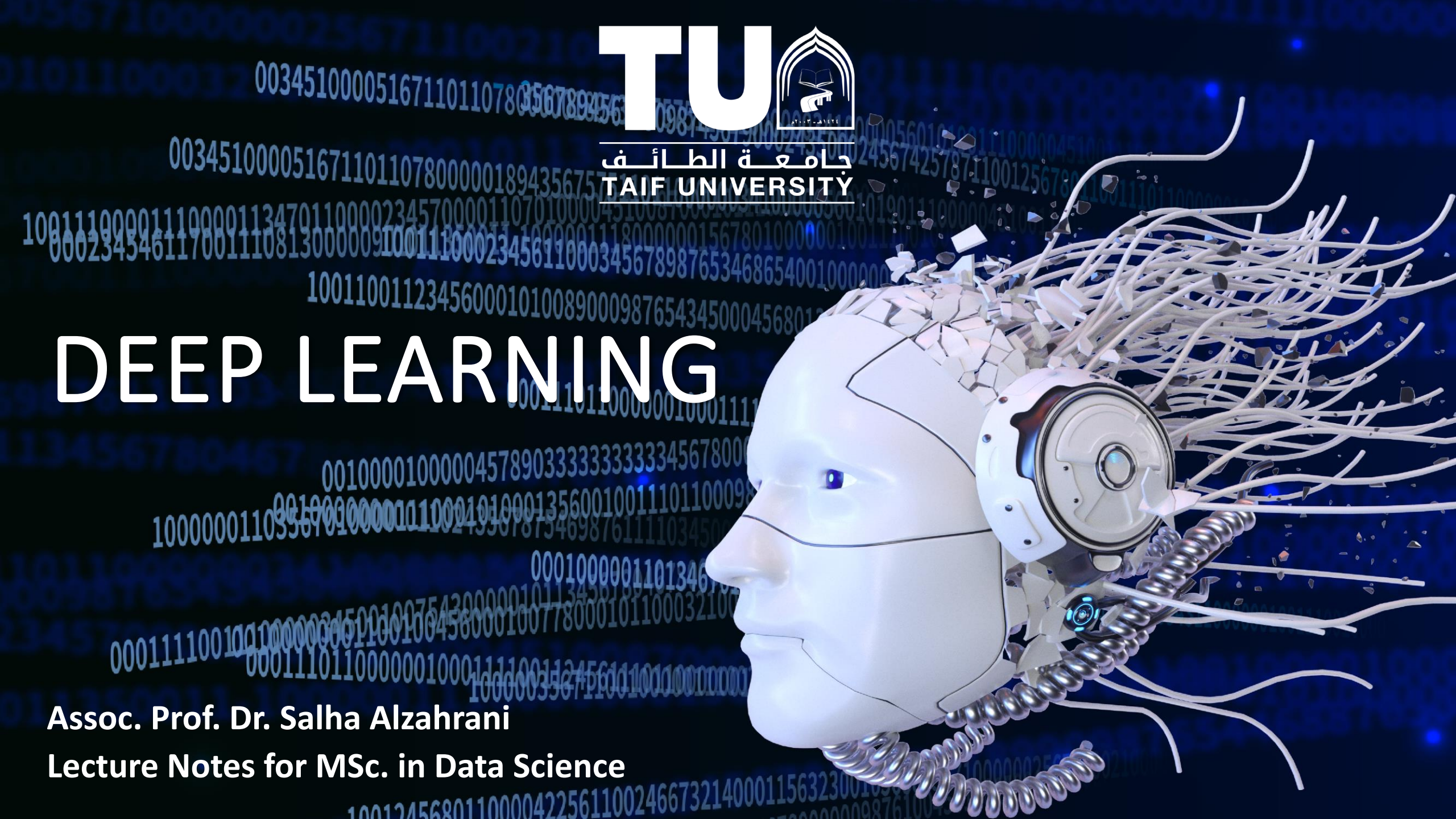
جامعة الطائف
TAIF UNIVERSITY

DEEP LEARNING



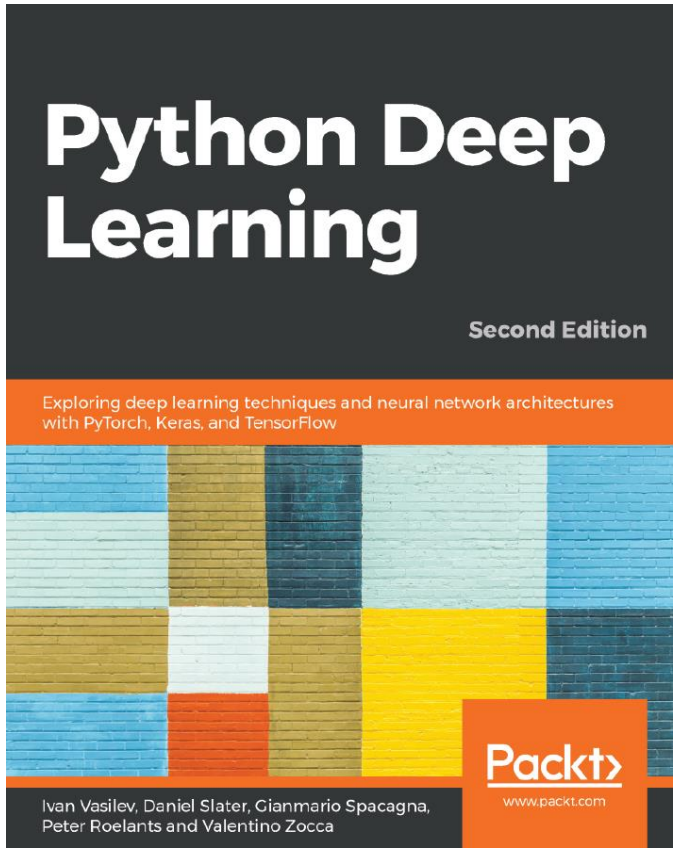
Assoc. Prof. Dr. Salha Alzahrani

Lecture Notes for MSc. in Data Science

My Notes

In this chapter, we will cover the following topics:

- Introduction to deep learning
- Fundamental deep learning concepts
- Deep learning algorithms
- Applications of deep learning
- The reasons for deep learning's popularity
- Introducing popular open-source libraries



Introduction to deep learning

- In 2012, Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton published a milestone paper titled ImageNet Classification with Deep Convolutional Neural Networks.
- The paper describes their use of neural networks to win the ImageNet competition of the same year, which we mentioned in Module (1).

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

Introduction to deep learning

- At the end of their paper, they wrote the following:

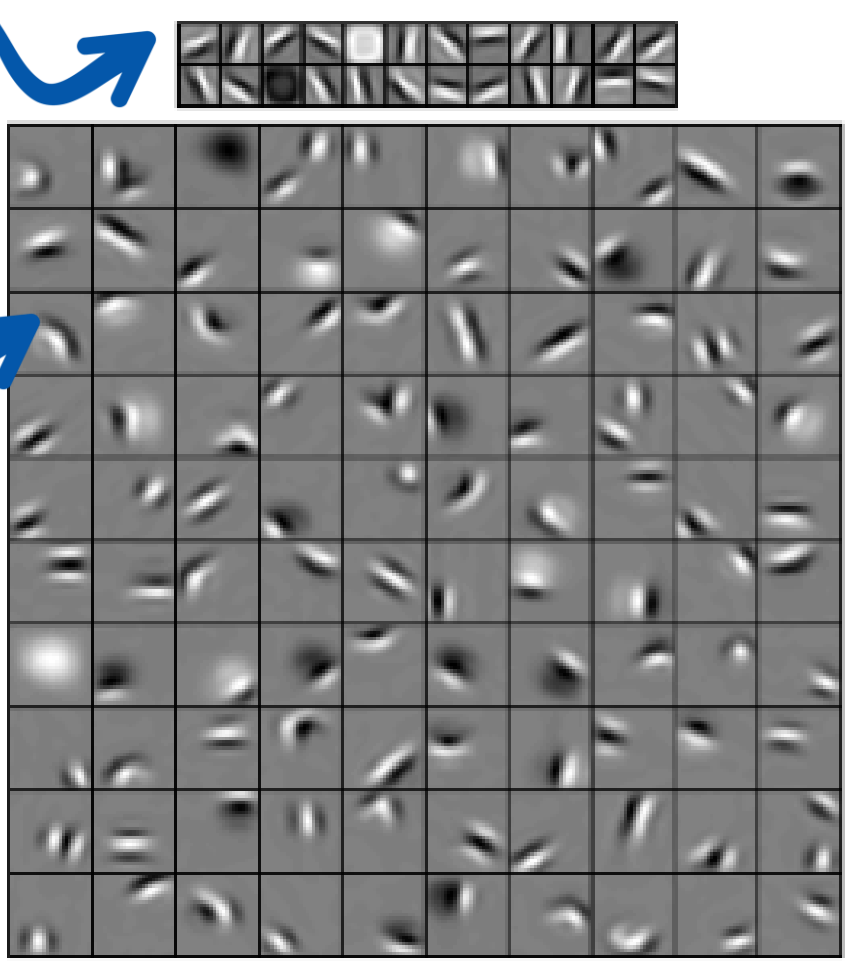
> *"It is notable that our network's performance degrades if a single convolutional layer is removed. For example, removing any of the middle layers results in a loss of about 2% for the top-1 performance of the network. So the depth really is important for achieving our results."*

- We will cover convolutional neural networks (CNN) in Module (3), Computer Vision With Convolutional Networks, but the basic question remains:

What do those hidden layers do?

Introduction to deep learning

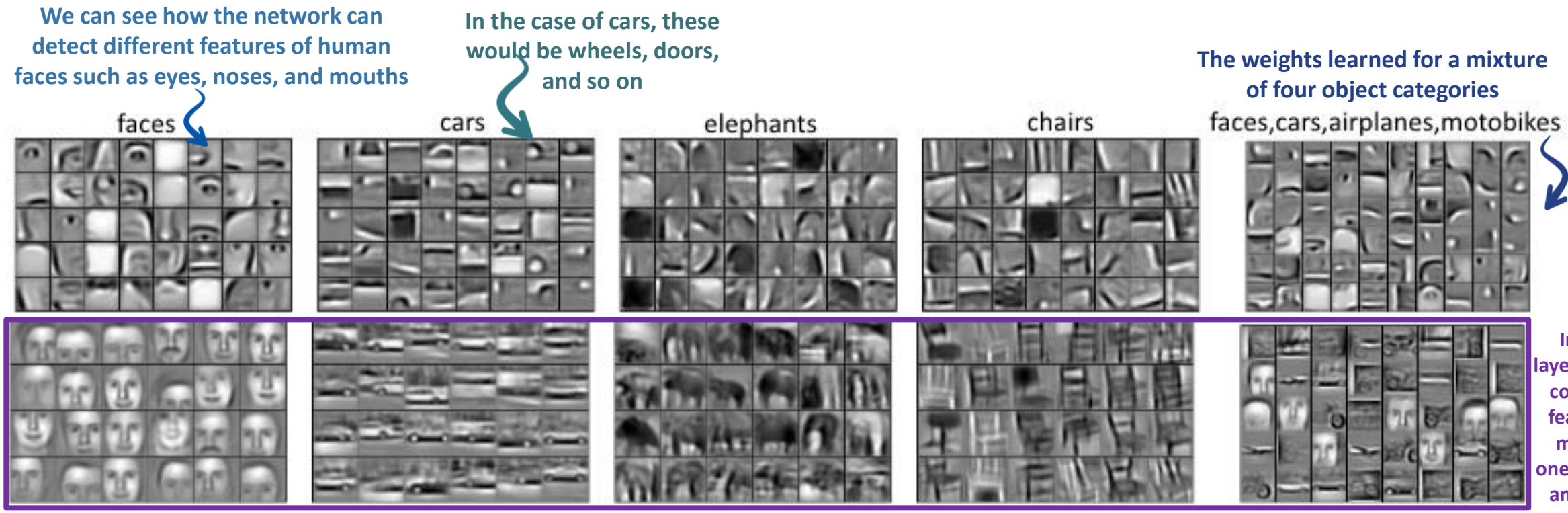
- A typical English saying is a picture is worth a thousand words. Let's use this approach to understand what deep learning is.
- We'll use images from the highly-cited paper : “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations”.
- The authors train a neural network with pictures of different categories of either objects or animals.
- We can see how the different layers of the network learn different characteristics of the input data. In the first layer, the network learns to detect some small basic features such as lines and edges, which are common for all images in all categories.



https://ai.stanford.edu/~ang/papers/icml09-ConvolutionalDeepBeliefNetworks.pdf

Introduction to deep learning

- But in the next layers, it combines those lines and edges to compose **more complex features** that are **specific for each category**.



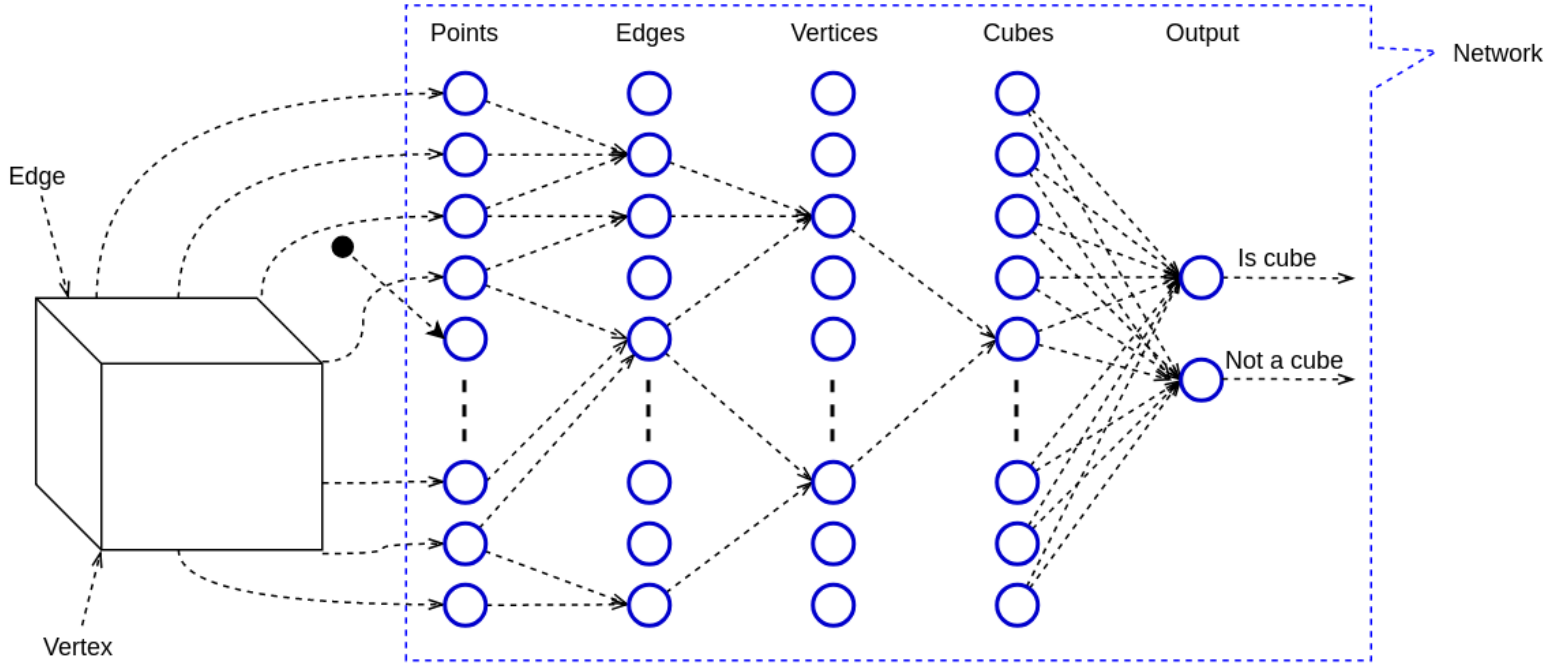
Fundamental deep learning concepts

- A deep neural network does not simply recognize what makes a car a car, or a face a face, but **it understands what features are present** in a car and a face respectively.
- A deep network that learns basic representations of its output can make classifications using the assumptions it has made. In this way, the amount of information the network learns is much completer and more robust, and the most exciting part is that deep neural networks learn to do this automatically.

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example.
Deep learning is part of a broader family of machine learning methods based on artificial neural networks with representation learning.
Learning can be supervised, semi-supervised or unsupervised.

Fundamental deep learning concepts: Feature learning

- To illustrate how deep learning works, let's consider the task of recognizing a simple geometric figure, for example, a cube, as seen in the diagram.



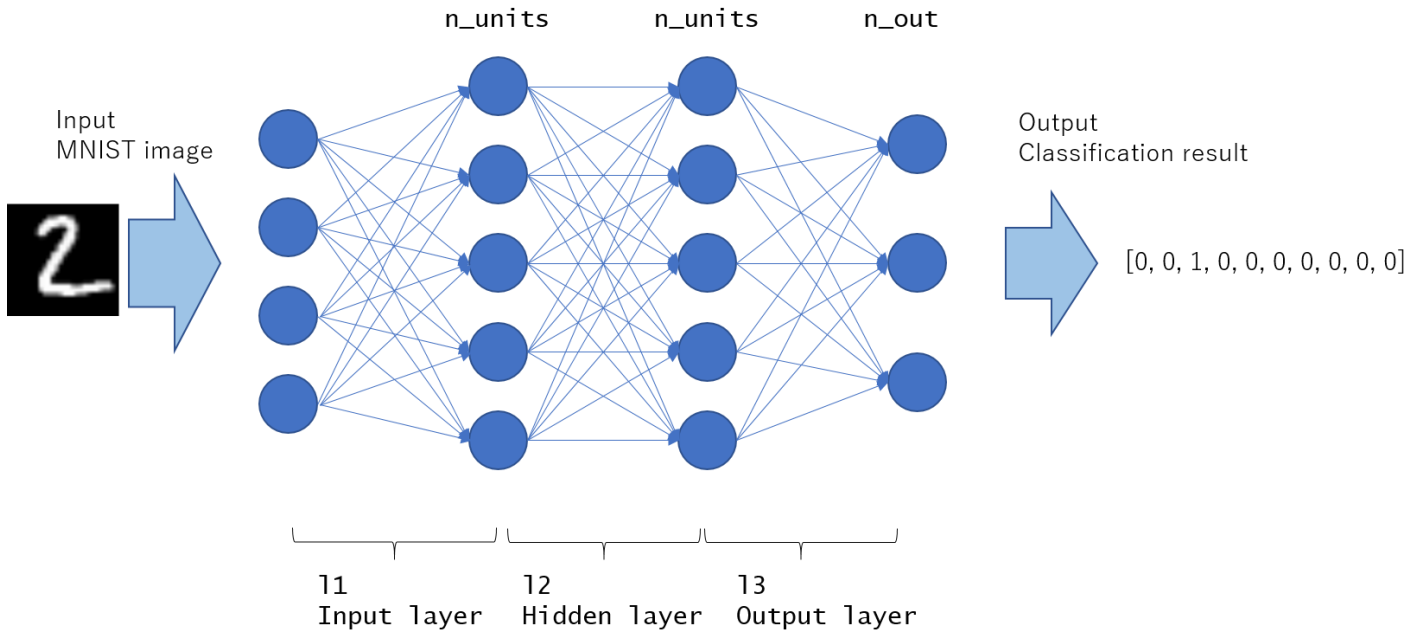
Now we have two hidden layers, with different levels of abstraction—the first for points and the second for edges. But this is not enough to encode a whole cube in the network. Let's try with another layer for vertices. Here, each three active edge/neurons of the second layer, which form a vertex, have a significant positive connection to a single common vertex/neuron of the third layer. Since an edge of the cube forms two vertices, each edge/neuron will have positive connections to two vertices/neurons and negative connections to all others. Finally, we'll introduce the last hidden layer (cube). The four vertices/neurons forming a cube will have positive connections to a single cube/neuron from the cube/layer:

Fundamental deep learning concepts: Feature learning

- Different layers encode features with different levels of abstraction
- The cube representation example is oversimplified, but we can draw several conclusions from it. One of them is that deep neural networks lend themselves well to hierarchically organized data. For example, an image consists of pixels, which form lines, edges, regions, and so on. This is also true for speech, where the building blocks are called phonemes; as well as text, where we have characters, words, and sentences.
- In the preceding example, we dedicated layers to specific cube features deliberately, but in practice, we wouldn't do that. **Instead, a deep network will "discover" features automatically during training.** These features might not be immediately obvious and, in general, wouldn't be interpretable by humans.
- Also, we wouldn't know the level of the features encoded in the different layers of the network.
- In classic machine learning algorithms, where the user has to use his/her own experience to select what they think are the best features. This process is called **feature engineering**, and it can be labor-intensive and time-consuming.
- Allowing a network to automatically discover features is not only easier, but those features are highly abstract, which makes them less sensitive to noise. For example, the network can learn to recognize faces better, even in more challenging conditions.

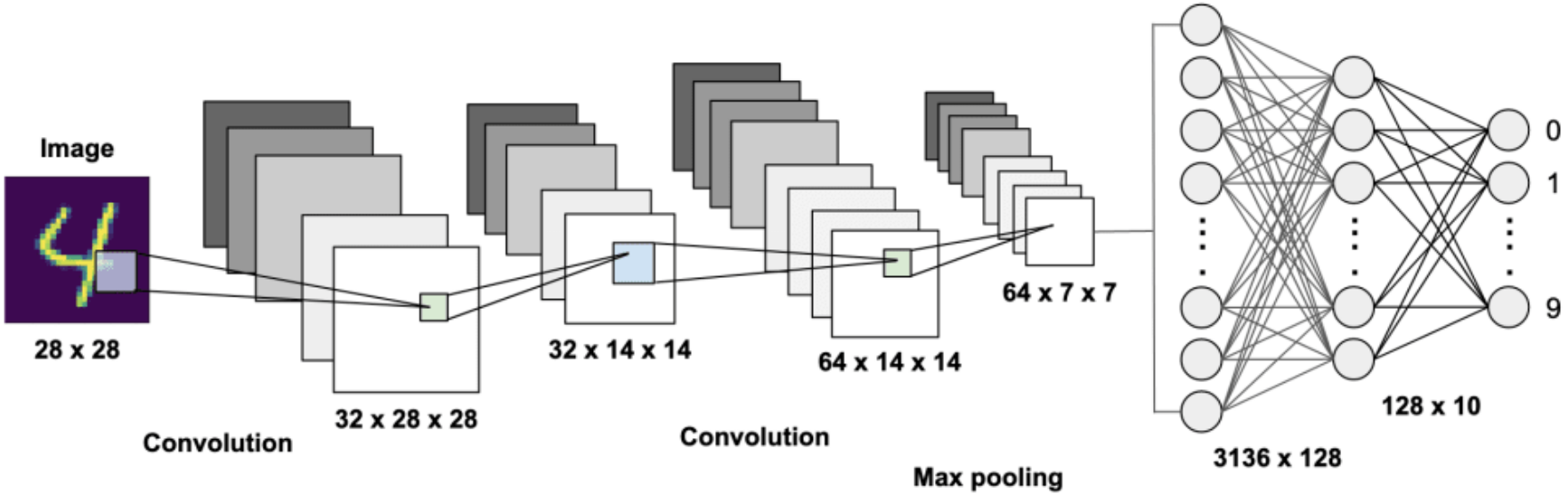
Deep learning algorithms

- **Multi-layer perceptrons (MLPs):** A neural network with feed-forward propagation, fully-connected layers, and at least one hidden layer. We introduced MLPs in Chapter 2, *Neural Networks*.



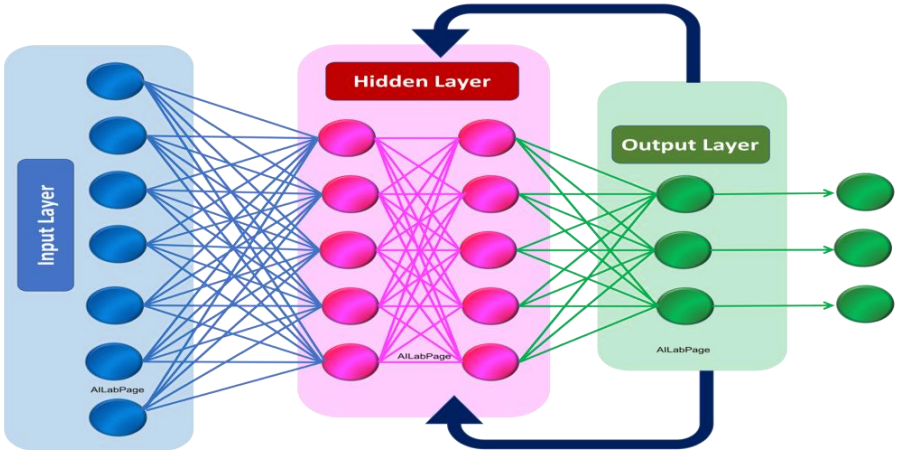
Deep learning algorithms

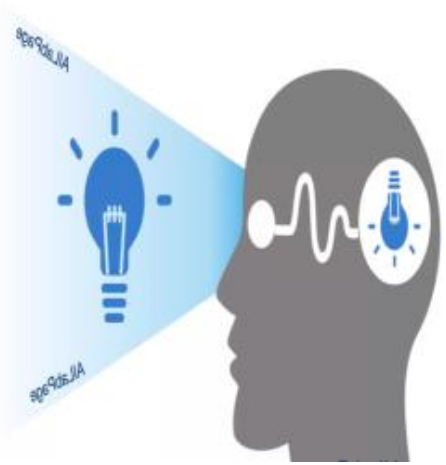
- **Convolutional neural networks (CNNs):** A CNN is a feedforward neural network with several types of special layers. For example, convolutional layers apply a filter to the input image (or sound) by sliding that filter all across the incoming signal, to produce an *n*-dimensional activation map. There is some evidence that neurons in CNNs are organized similarly to how biological cells are organized in the visual cortex of the brain. We've mentioned CNNs several times up to now, and that's not a coincidence – today, they outperform all other ML algorithms on a large number of computer vision and NLP tasks.



Deep learning algorithms

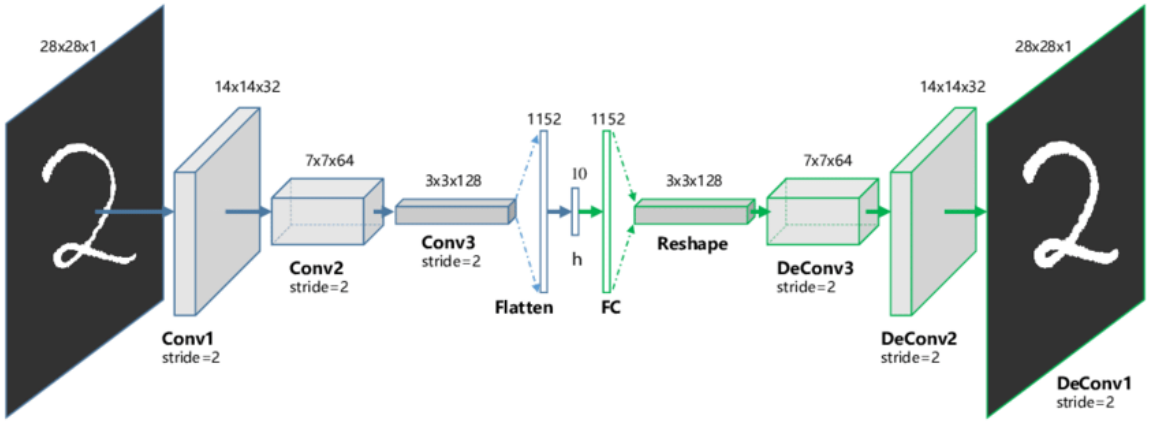
- **Recurrent networks:** This type of network has an internal state (or memory), which is based on all or part of the input data already fed to the network. The output of a recurrent network is a combination of its internal state (memory of inputs) and the latest input sample. At the same time, the internal state changes, to incorporate newly input data. Because of these properties, recurrent networks are good candidates for tasks that work on sequential data, such as text or time-series data. We'll discuss recurrent networks in Chapter 7, *Recurrent Neural Networks and Language Models.*

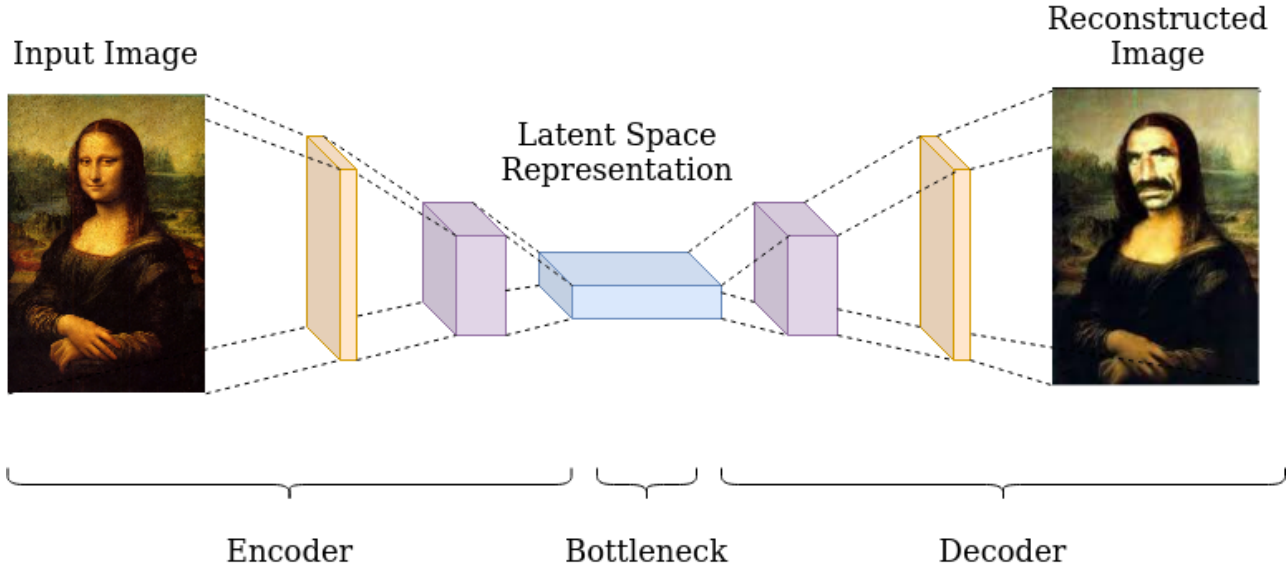




Deep learning algorithms

- **Autoencoders:** A class of unsupervised learning algorithms, in which the output shape is the same as the input that allows the network to better learn basic representations. We'll discuss autoencoders when we talk about generative deep learning, in Chapter 6, *Generating Images with GANs and VAEs.*





Training deep networks

- As we mentioned in chapter 2, Neural Networks, we can use different algorithms to train a neural network.
- In practice, we almost always use **Stochastic Gradient Descent (SGD) and backpropagation**.

- We'll introduce **momentum**, which is an effective improvement over the vanilla gradient descent.

1. $w \rightarrow w - \lambda \nabla(J(w))$, where λ is the learning rate.

To include momentum, we'll add another parameter to this equation.

2. First, we'll calculate the weight update value:

$$\triangle w \rightarrow \mu \triangle w - \lambda(\nabla J(w))$$

3. Then, we'll update the weight:

$$w \rightarrow w + \triangle w$$

From the preceding equation, we see that the first component, $\mu \triangle w$, is the momentum.

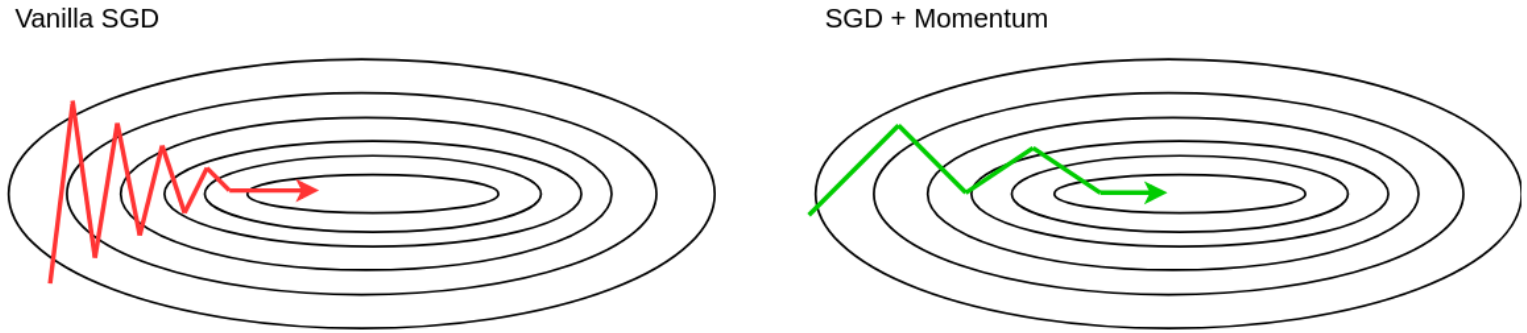
The $\triangle w$ represents the previous value of the weight update and μ is the coefficient,

which will determine how much the new value depends on the previous ones. To explain this, let's look at the following diagram, where you will see a comparison between vanilla SGD and SGD with momentum. The concentric ellipses represent the surface of the error

Training deep networks

- The concentric ellipses represent the surface of the error function, where the innermost ellipse is the minimum and the outermost the maximum. Think of the **loss function surface as the surface of a hill**. Now, imagine that we are holding a ball at the top of the hill (maximum). If we drop the ball, thanks to Earth's gravity, it will start rolling toward the bottom of the hill (minimum). The more distance it travels, the more its speed will increase. In other words, it will gain momentum (hence the name of the optimization). As a result, it will reach the bottom of the hill faster. If, for some reason, gravity didn't exist, the ball would roll at its initial speed and it would reach the bottom more slowly.

- In your practice, you may encounter other gradient descent optimizations, such as Nesterov momentum, ADADELTA, RMSProp, and Adam. Some of these will be discussed throughout the course.



A comparison between vanilla SGD and SGD + momentum

Applications of deep learning

- **Google's Vision API** and **Amazon's Recognition** service use deep learning models to provide various computer vision capabilities. These include recognizing and detecting objects and scenes in images, text recognition, face recognition, and so on.

- If these APIs are not enough, you can run your own models in the cloud. For example, you can use **Amazon's AWS Deep Learning AMIs (Amazon Machine Images)**, virtual machines that come configured with some of the most popular DL libraries. Google offers a similar service with their **Google Cloud AI**, but they've gone one step further. They created Tensor Processing Units TPUs, microprocessors, optimized for fast neural network operations such as matrix multiplication and activation functions.

Applications of deep learning

- Deep learning has a lot of potential for medical applications. However, strict regulatory requirements, as well as patient data confidentiality have slowed down its adoption. Nevertheless, we'll identify two areas in which deep learning could have a high impact:
  - Medical imaging for creating visual representations of the inside of the body include **Magnetic resonance images (MRIs), ultrasound, Computed Axial Tomography (CAT) scans, X-rays, and histology images**. Typically, such an image is analyzed by a medical professional to determine the patient's condition. **Computer-aided diagnosis, and computer vision** in particular, can help specialists by detecting and highlighting important features of images. For example, to determine the degree of malignancy of colon cancer, a pathologist would have to analyze the morphology of the glands, using histology imaging. This is a challenging task, because morphology can vary greatly. A deep neural network could segment the glands from the image automatically, leaving the pathologist to verify the results. This would reduce the time needed for analysis, making it cheaper and more accessible.
  - Another medical area that could benefit from deep learning is the **analysis of medical history records**. When a doctor diagnoses a patient's condition and prescribes treatment, they consult the patient's medical history first. A deep learning algorithm could extract the most relevant and important information from those records, even if they are handwritten. In this way, the doctor's job would be made easier, and the risk of errors would also be reduced.

The reasons for deep learning's popularity

Data

- First, the rise of the internet and software in different industries has generated a lot of computer-accessible data. We also have more benchmark datasets, such as ImageNet. Deep learning algorithms work better when they are trained with a lot of data.

Computing Power

- Second, the increased computing power. This is most visible in the drastically increased processing capacity of Graphical Processing Units (GPUs). Architecturally, Central Processing Units (CPUs) are composed of a few cores that can handle a few threads at a time, while GPUs are composed of hundreds of cores that can handle thousands of threads in parallel. Neural networks are organized in such a way as to take advantage of this parallel architecture.

Algorithms

- Third, a combination of algorithmic advances, it's now possible to the train neural networks with almost arbitrary depth with the help of the combination. These include better activation functions, Rectified Linear Unit (ReLU), overcoming the vanishing gradients problems, better initialization of the network weights before training, new network architectures, as well as new types of regularization techniques such as Batch normalization.

Introducing popular open source libraries

- There are many open-source libraries that allow the creation of deep neural nets in Python, without having to explicitly write the code from scratch.
- Three of the most popular are TensorFlow, Keras, and PyTorch. They all share some common features.

The most popular deep learning library. It's developed and maintained by Google. You don't need to explicitly require the use of a GPU; rather TensorFlow will automatically try to use it if you have one. If you have more than one GPU, you must assign operations to each GPU explicitly, or only the first one will be used.
https://www.tensorflow.org

A high-level neural net Python library that uses TensorFlow on the backend. With Keras, you can perform rapid experimentation and it's relatively easy to use compared to TF. It will automatically detect an available GPU and attempt to use it. Otherwise, it will revert to the CPU. If you wish to specify the device manually, you can import TensorFlow and use the same code as in the previous section,
https://keras.io/

A deep learning library based on Torch and developed by Facebook. It is relatively easy to use, and has recently gained a lot of popularity. It will automatically select a GPU, if one is available, reverting to the CPU otherwise.

https://pytorch.org/

Watch this video: What is Deep Learning!

Let's have fun ☺

Framework to build a neural network using Keras

1. Prepare & Explore Dataset
- Clean data
- Split into train and test
- Split into predictors and target
- Normalize data

2. Define the neural network architecture

3. Compile the neural net

4. Fit / train the neural net

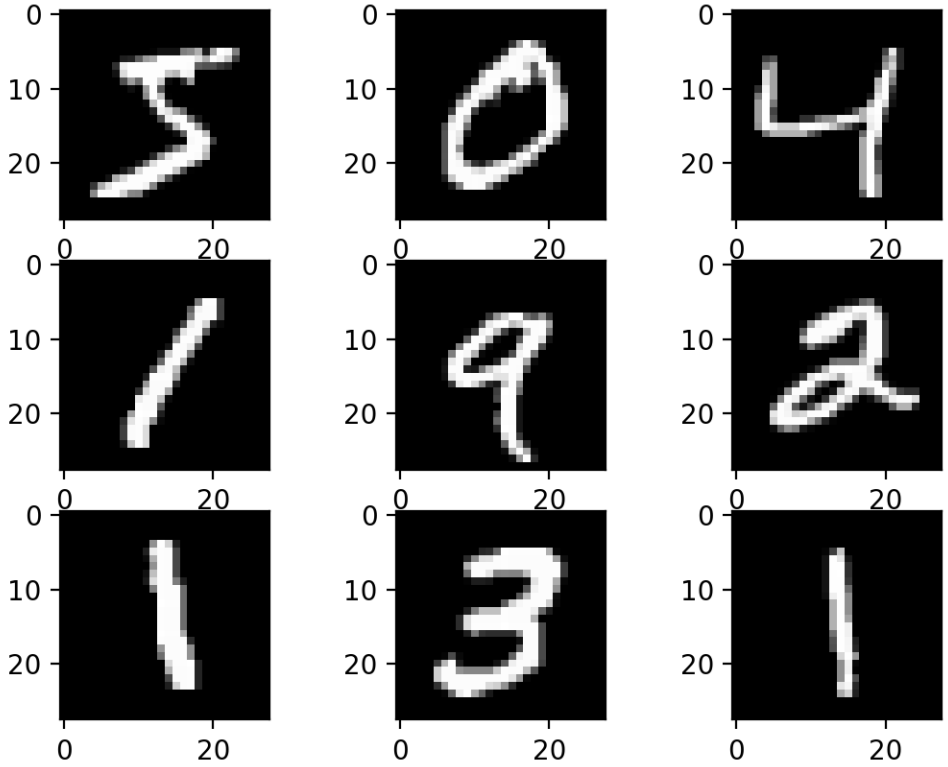
5. Evaluate the neural net

6. Make predictions/ classifications for unseen data

1. Prepare Dataset
2. Define architecture
3. Compile
4. Fit / train
5. Evaluate
6. Predict / Classify

Using Keras to classify handwritten digits

- In this section, we'll use Keras to classify the images of the MNIST dataset. It's comprised of 70,000 examples of handwritten digits by different people. The first 60,000 are typically used for training and the remaining 10,000 for testing:
- Sample of digits taken from the MNIST dataset
- One of the advantages of Keras is that it can import this dataset for you without needing to explicitly download it from the web (it will download it for you)



MNIST dataset

Using Keras to classify handwritten digits

1. Prepare & Explore Dataset

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
# import pachages
from tensorflow import keras
from keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from keras.utils import np_utils
```

In [3]:
```
# load the training and testing data.
# (X_train, Y_train) are the training images and labels,
# (X_test, Y_test) are the test images and labels

(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

Using Keras to classify handwritten digits

```
In [4]: X_train.shape, Y_train.shape
Out[4]: ((60000, 28, 28), (60000,))

In [5]: X_test.shape, Y_test.shape
Out[5]: ((10000, 28, 28), (10000,))

In [ ]: X_train, Y_train

In [ ]: X_test, Y_test
```

```
[[0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 ...,
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0]],

[[0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 ...,
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0]],

[[0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 ...,
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0]]], dtype=uint8),
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8))
```

Using Keras to classify handwritten digits

```
In [10]: X_train, X_train.shape
```

```
Out[10]: (array([[0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0],
                 ...,
                 [0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0]], dtype=uint8),
          (60000, 784))
```

```
In [11]: X_test, X_test.shape
```

```
Out[11]: (array([[0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0],
                 ...,
                 [0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0]], dtype=uint8),
          (10000, 784))
```

Using Keras to classify handwritten digits

In [12]:
```
# The labels indicate the value of the digit depicted in the images.
# We want to convert this into a 10-entry encoded vector comprised of 0s and 1 in the entry corresponding to the digit.
# For example, 4 is mapped to [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
# Conversely, our network will have 10 output neurons

classes = 10
Y_train = np_utils.to_categorical(Y_train, classes)
Y_test = np_utils.to_categorical(Y_test, classes)
```

In [13]:
```
Y_train, Y_test
```

Out[13]:
```
(array([[0., 0., 0., ..., 0., 0., 0.],
        [1., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 1., 0.]], dtype=float32),
 array([[0., 0., 0., ..., 1., 0., 0.],
        [0., 0., 1., ..., 0., 0., 0.],
        [0., 1., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]], dtype=float32))
```

Using Keras to classify handwritten digits

2. Define the neural network architecture

In [14]:
```
# Before calling our main function, we need to set :
## the size of the input layer (the size of the MNIST images),
## the number of hidden neurons,
## the number of epochs to train the network,
## the mini batch size:

input_size = 784
batch_size = 100
hidden_neurons = 100
epochs = 100
```

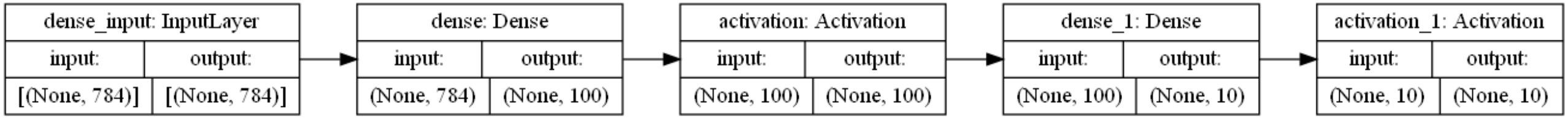
In [15]:
```
model = Sequential([
    Dense(hidden_neurons, input_dim=input_size),
    Activation('sigmoid'),
    Dense(classes),
    Activation('softmax')
])
```

Using Keras to classify handwritten digits

3. Compile the neural net

In [16]:
```
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='sgd')
```

In [17]:
```
keras.utils.plot_model(model, show_shapes=True, rankdir="LR")
```



Using Keras to classify handwritten digits

4. Fit / train the neural net

```
In [18]: model.fit(X_train, Y_train, batch_size=batch_size, epochs=epochs, verbose=1)
Epoch 91/100
600/600 [==============================] - 0s 797us/step - loss: 0.0744 - accuracy: 0.9808
Epoch 92/100
600/600 [==============================] - 0s 808us/step - loss: 0.0752 - accuracy: 0.9806
Epoch 93/100
600/600 [==============================] - 0s 793us/step - loss: 0.0768 - accuracy: 0.9806
Epoch 94/100
600/600 [==============================] - 0s 771us/step - loss: 0.0746 - accuracy: 0.9808
Epoch 95/100
600/600 [==============================] - 0s 763us/step - loss: 0.0768 - accuracy: 0.9802
Epoch 96/100
600/600 [==============================] - 0s 766us/step - loss: 0.0714 - accuracy: 0.9817
Epoch 97/100
600/600 [==============================] - 0s 769us/step - loss: 0.0744 - accuracy: 0.9804
Epoch 98/100
600/600 [==============================] - 0s 769us/step - loss: 0.0720 - accuracy: 0.9803
Epoch 99/100
600/600 [==============================] - 0s 767us/step - loss: 0.0719 - accuracy: 0.9809
Epoch 100/100
600/600 [==============================] - 0s 768us/step - loss: 0.0709 - accuracy: 0.9818
```

Using Keras to classify handwritten digits

5. Evaluate the neural net

In [19]:
```
score = model.evaluate(X_test, Y_test, verbose=1)
print('Test accuracy:', score[1])
```

```
313/313 [==============================] - 0s 502us/step - loss: 0.1202 - accuracy: 0.9633
Test accuracy: 0.9632999897003174
```

Using Keras to classify handwritten digits

6. Make predictions / classifications for unseen data

In [20]:
```
#not yet until we enhanced the results
predictions = model.predict(X_test)
predictions
```

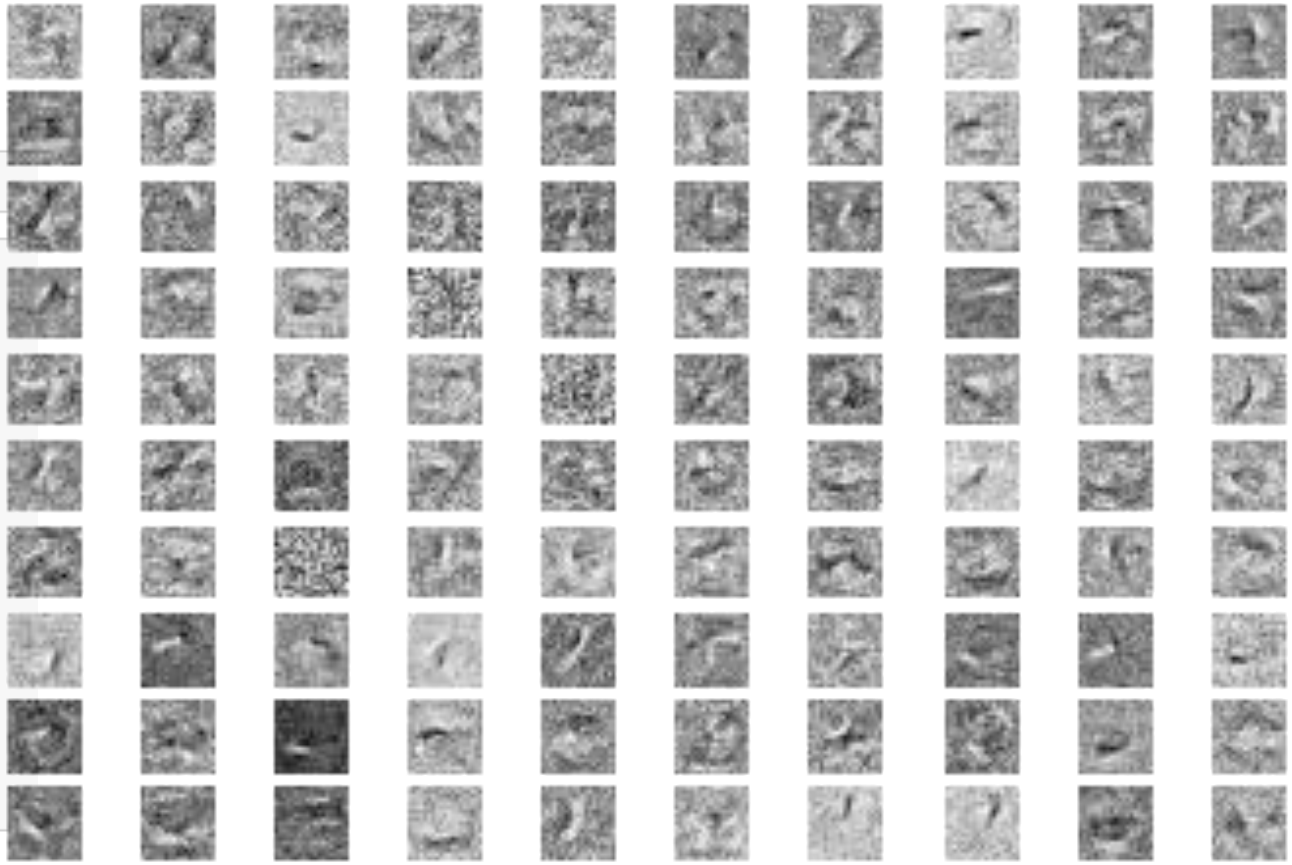
Out[20]:
```
array([[2.2499098e-06, 9.0058893e-06, 2.1370495e-04, ..., 9.9944431e-01,
        9.5456346e-07, 1.8858127e-05],
       [5.1176903e-04, 3.5238878e-03, 9.8659480e-01, ..., 8.7109565e-06,
        8.0965325e-04, 1.8214442e-06],
       [3.3645546e-07, 9.9926156e-01, 2.7820151e-04, ..., 2.0082793e-04,
        5.4758559e-05, 7.3311012e-06],
       ...,
       [1.4397752e-07, 3.3628774e-06, 1.4708672e-05, ..., 1.5689414e-04,
        5.2107323e-04, 1.4624527e-03],
       [5.8550877e-04, 3.0973942e-03, 3.8389757e-04, ..., 4.3055727e-05,
        4.5891270e-02, 2.5504200e-05],
       [9.0980575e-05, 1.6009022e-05, 4.2505338e-04, ..., 1.7255367e-06,
        3.0155705e-05, 3.3327713e-06]], dtype=float32)
```

Using Keras to classify handwritten digits

To see what the network has learned, we can visualize the weights of the hidden layer .

In [21]:
```
weights = model.layers[0].get_weights()
```

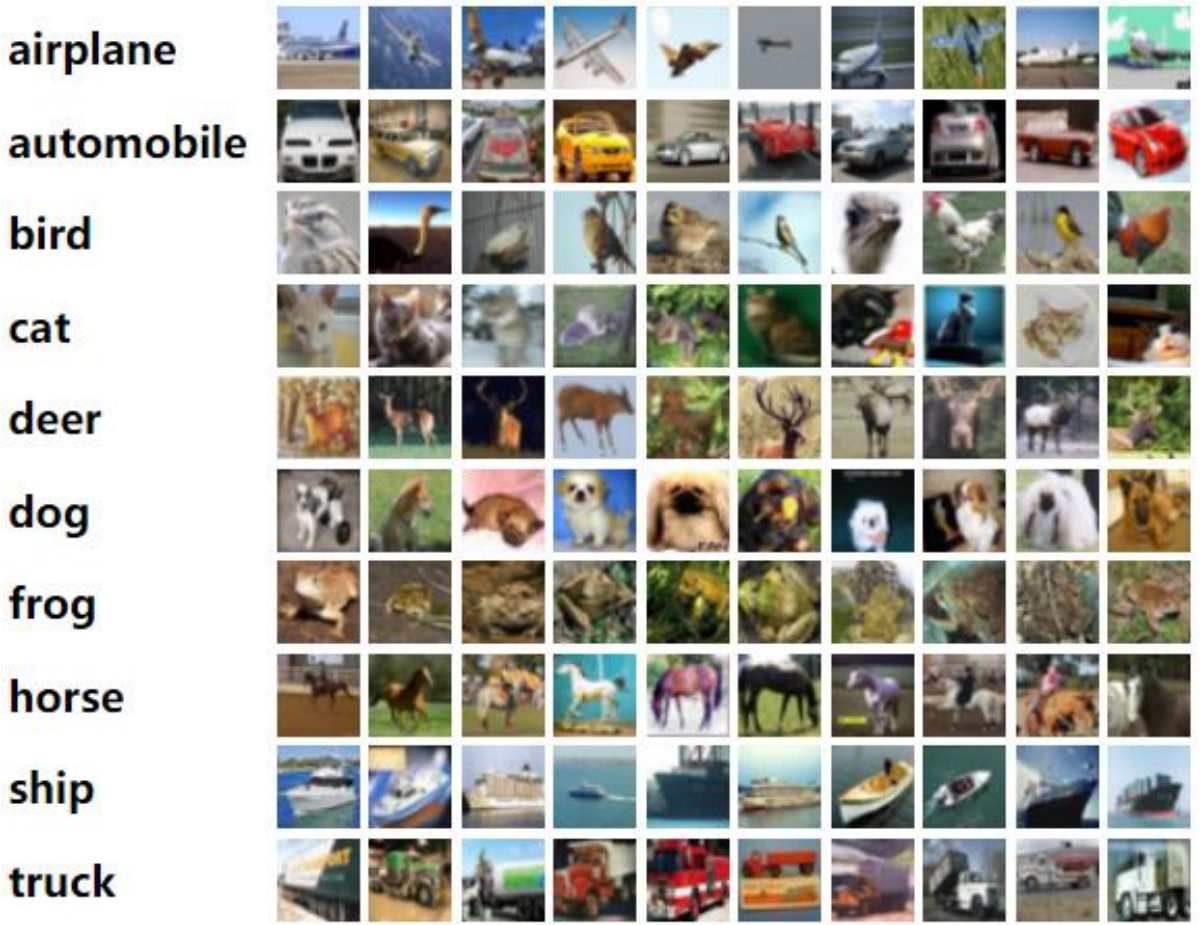
In [22]:
```
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy

fig = plt.figure()

w = weights[0].T
for neuron in range(hidden_neurons):
 ax = fig.add_subplot(10, 10, neuron + 1)
 ax.axis("off")
 ax.imshow(numpy.reshape(w[neuron], (28, 28)), cmap=cm.Greys_r)

plt.savefig("neuron_images.png", dpi=300)
plt.show()
```



Using Keras to classify images of objects

- With Keras, it's easy to create neural nets, but it's also easy to download test datasets. Let's try to use the **CIFAR-10** (Canadian Institute For Advanced Research, https://www.cs.toronto.edu/~kriz/cifar.html) dataset.

- It consists of 60,000 32x32 RGB images, divided into 10 classes of objects, namely:



Using Keras to classify images of objects

1. Prepare & Explore Dataset

In [1]:
```
import numpy as np
import pandas as pd
```

In [19]:
```
# import pachages
from tensorflow import keras
from keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from keras.utils import np_utils
```

In [3]:
```
# load the training and testing data.
# (X_train, Y_train) are the training images and labels,
# (X_test, Y_test) are the test images and labels

(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 40s 0us/step
```

Using Keras to classify images of objects

In [6]:
```
X_train.shape, Y_train.shape
```

Out[6]:
```
((50000, 32, 32, 3), (50000, 1))
```

In [7]:
```
X_test.shape, Y_test.shape
```

Out[7]:
```
((10000, 32, 32, 3), (10000, 1))
```

In [8]:
```
# we'll split the data into 50,000 training images and 10,000 testing images.
# We need to reshape the image to a one-dimensional array.
# In this case, each image has 3 color channels (red, green, and blue) of 32x32 pixels, hence 3 x32x3 = 3072

X_train = X_train.reshape(50000, 3072)
X_test = X_test.reshape(10000, 3072)
```

Using Keras to classify images of objects

In [11]:
```
# 10 classes of objects, namely: airplanes, automobiles, birds, cats, deers, dogs, frogs, horses, ships, and trucks:
# We want to convert this into a 10-entry encoded vector comprised of 0s and 1 in the entry corresponding to the digit.
# Conversely, our network will have 10 output neurons

classes = 10
Y_train = np_utils.to_categorical(Y_train, classes)
Y_test = np_utils.to_categorical(Y_test, classes)
```

In [12]:
```
Y_train, Y_test
```

Out[12]:
```
(array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 1.],
        [0., 0., 0., ..., 0., 0., 1.],
        ...,
        [0., 0., 0., ..., 0., 0., 1.],
        [0., 1., 0., ..., 0., 0., 0.],
        [0., 1., 0., ..., 0., 0., 0.]], dtype=float32),
 array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 1., 0.],
        [0., 0., 0., ..., 0., 1., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 1., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 1., 0., 0.]], dtype=float32))
```

Using Keras to classify images of objects

2. Define the neural network architecture

In [13]:
```
# Before calling our main function, we need to set :
## the size of the input layer (the size of the MNIST images),
## the number of hidden neurons,
## the number of epochs to train the network,
## the mini batch size:

input_size = 3072
batch_size = 100
epochs = 100
```

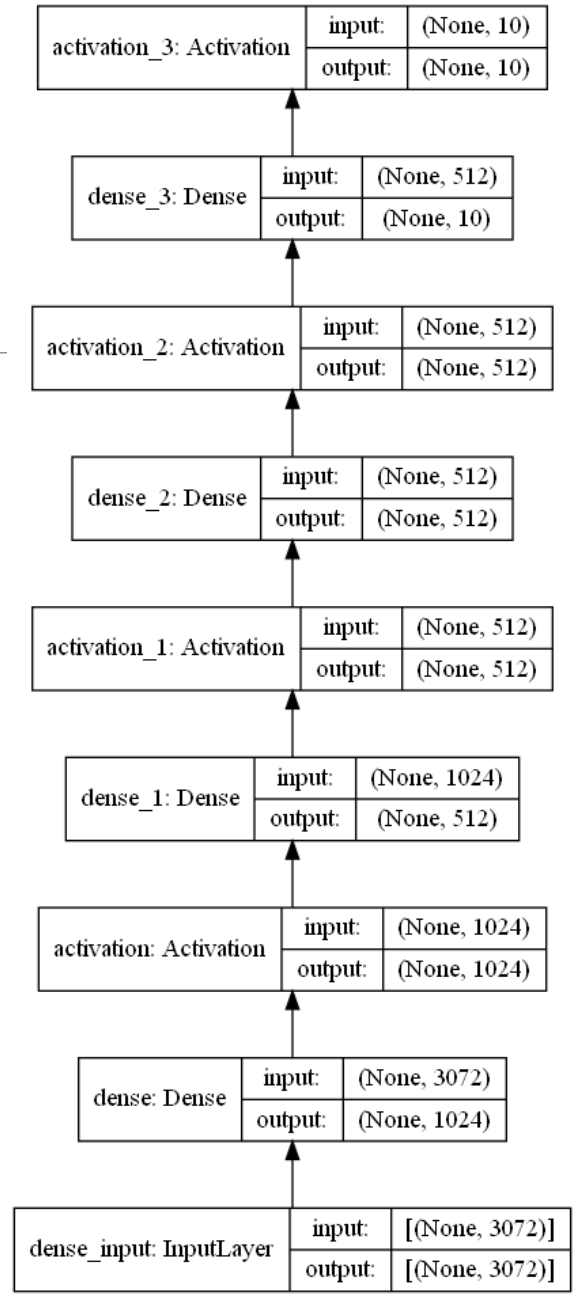
In [14]:
```
model = Sequential([
    Dense(1024, input_dim=input_size),
    Activation('relu'),
    Dense(512),
    Activation('relu'),
    Dense(512),
    Activation('sigmoid'),
    Dense(classes),
    Activation('softmax')
])
```

Using Keras to classify images of objects

3. Compile the neural net

In [17]:
```
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='sgd')
```

In [23]:
```
keras.utils.plot_model(model, show_shapes=True, rankdir="BT")
```



Using Keras to classify images of objects

4. Fit / train the neural net

In [24]:
```
model.fit(X_train, Y_train, batch_size=batch_size, epochs=epochs, validation_data=(X_test, Y_test), verbose=1)
```

```
Epoch 95/100
500/500 [==============================] - 8s 15ms/step - loss: 1.0345 - accuracy: 0.6335 - val_loss: 1.3761 - val_accura
cy: 0.5173
Epoch 96/100
500/500 [==============================] - 8s 15ms/step - loss: 1.0266 - accuracy: 0.6337 - val_loss: 1.3970 - val_accura
cy: 0.5226
Epoch 97/100
500/500 [==============================] - 8s 16ms/step - loss: 1.0242 - accuracy: 0.6399 - val_loss: 1.3769 - val_accura
cy: 0.5262
Epoch 98/100
500/500 [==============================] - 8s 16ms/step - loss: 1.0235 - accuracy: 0.6375 - val_loss: 1.3865 - val_accura
cy: 0.5195
Epoch 99/100
500/500 [==============================] - 8s 15ms/step - loss: 1.0315 - accuracy: 0.6310 - val_loss: 1.3887 - val_accura
cy: 0.5233
Epoch 100/100
500/500 [==============================] - 8s 15ms/step - loss: 0.9986 - accuracy: 0.6487 - val_loss: 1.3763 - val_accura
cy: 0.5290
```

Using Keras to classify images of objects

5. Evaluate the neural net

In [25]:
```
# Done in the previous step, how?
```

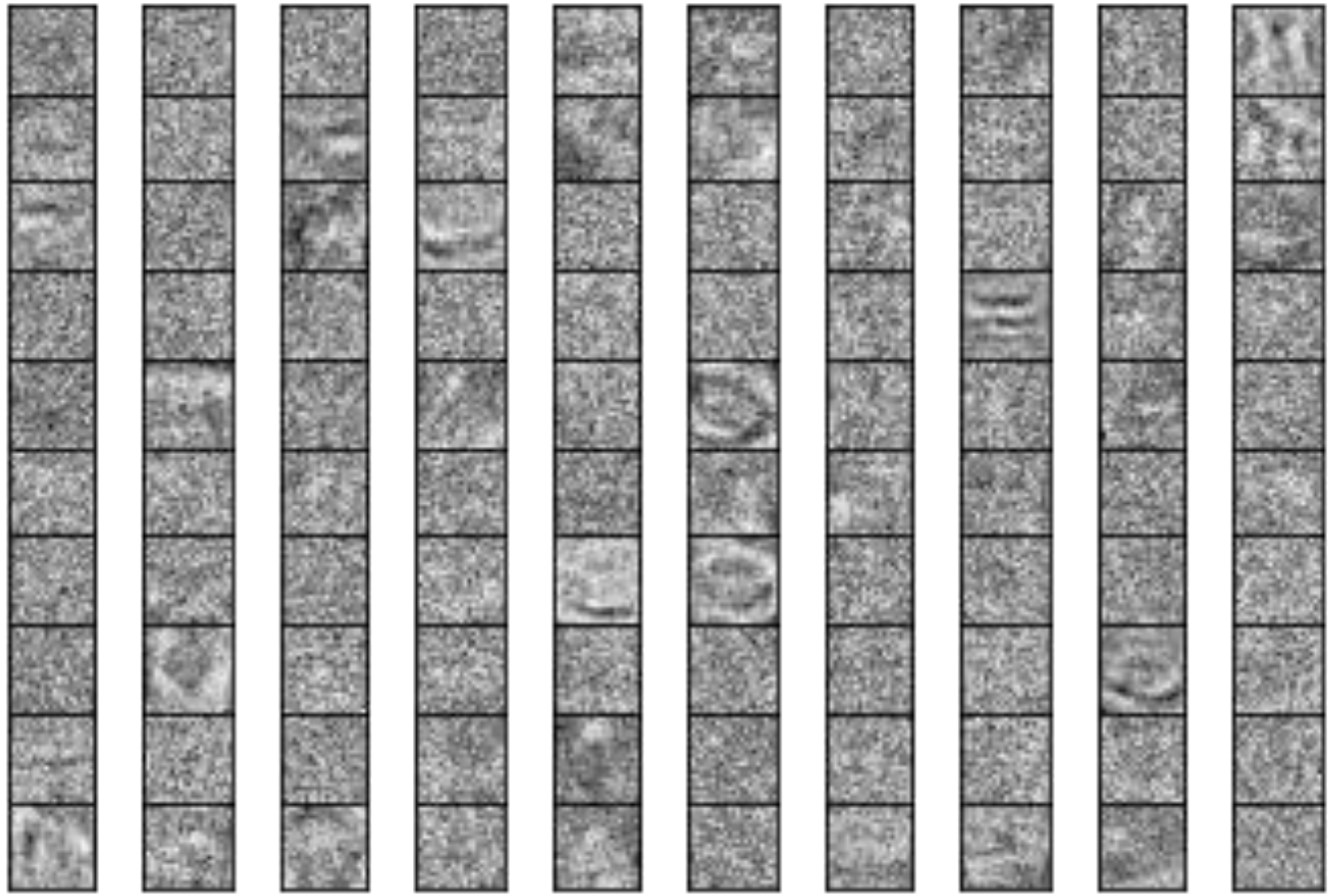
6. Make predictions / classifications for unseen data

In [26]:
```
#not yet until we enhanced the results
predictions = model.predict(X_test)
predictions
```

Out[26]:
```
array([[3.1462580e-02, 6.0446355e-03, 7.5248957e-02, ..., 7.1296850e-03,
        1.2931229e-01, 2.2290808e-03],
       [2.5457250e-02, 7.7308036e-02, 1.7517004e-03, ..., 3.1612776e-04,
        7.3568988e-01, 1.5735437e-01],
       [3.9775747e-01, 3.5304338e-02, 1.3587039e-02, ..., 1.3170856e-02,
        4.7061589e-01, 4.6115007e-02],
```

Using Keras to classify images of objects

To see what the network has learned, we can visualize the weights of the hidden layer



My Notes

Recap!

- ❑ Introduction to deep learning
- ❑ Fundamental deep learning concepts
- ❑ Deep learning algorithms
- ❑ Applications of deep learning
- ❑ The reasons for deep learning's popularity
- ❑ Introducing popular open-source libraries
- ❑ MLP : MNIST dataset
- ❑ MLP : CIFAR-10 dataset

The true measure of success is how many times you can bounce back from failure.

Stephen Richards

@SalhaAlzahrani