Intro to Applied Deep Learning

Florian Muellerklein
fmuellerklein@minerkasch.com

MINER & KASCH
DATA SCIENCE

Contents

What is deep learning?

Conventional machine learning algorithms have some problems with feature engineering:

- Time consuming
- Error prone
- Difficult for complex data types (images, text)
- Requires domain expertise

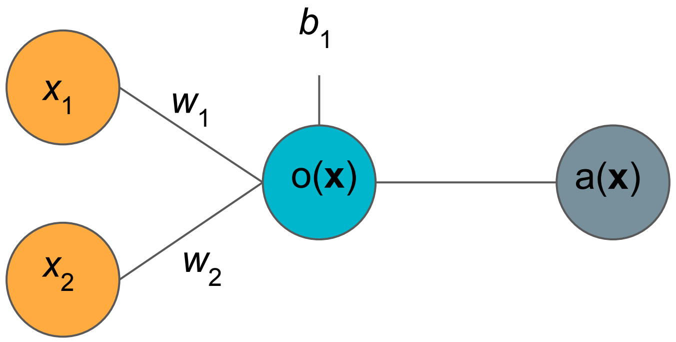
With Deep Learning:

- Model learns its own way to represent the features
- Able to work on data in a more raw form

http://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf

What is deep learning?

Deep learning models are essentially large neural networks. Neural networks are basically the function (neuron) below chained together many times.



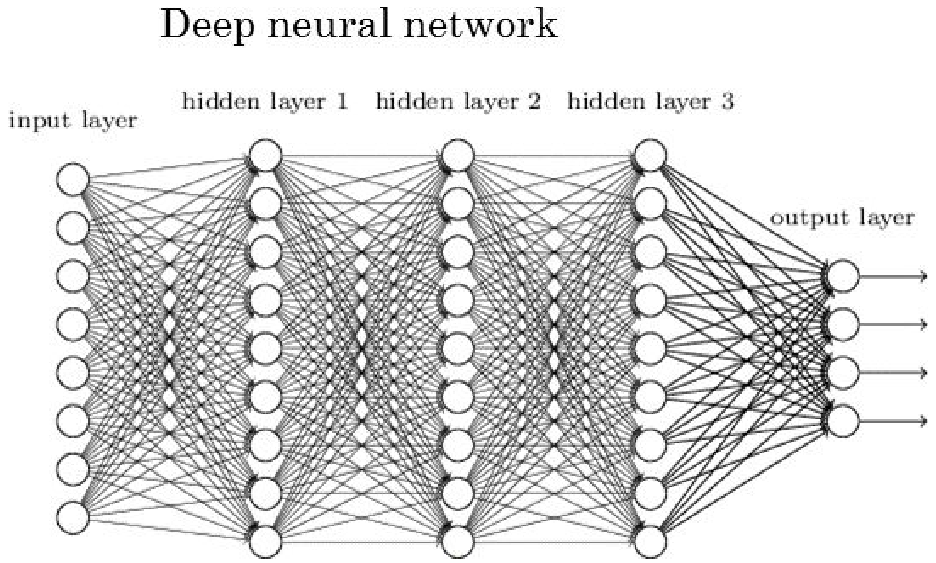
$$a(x) = a(o(x)) = a(b + \sum_i^n x_i w_i)$$

What is deep learning?

Deep learning models are essentially large neural networks. Neural networks are basically the function (neuron) below chained together many times.

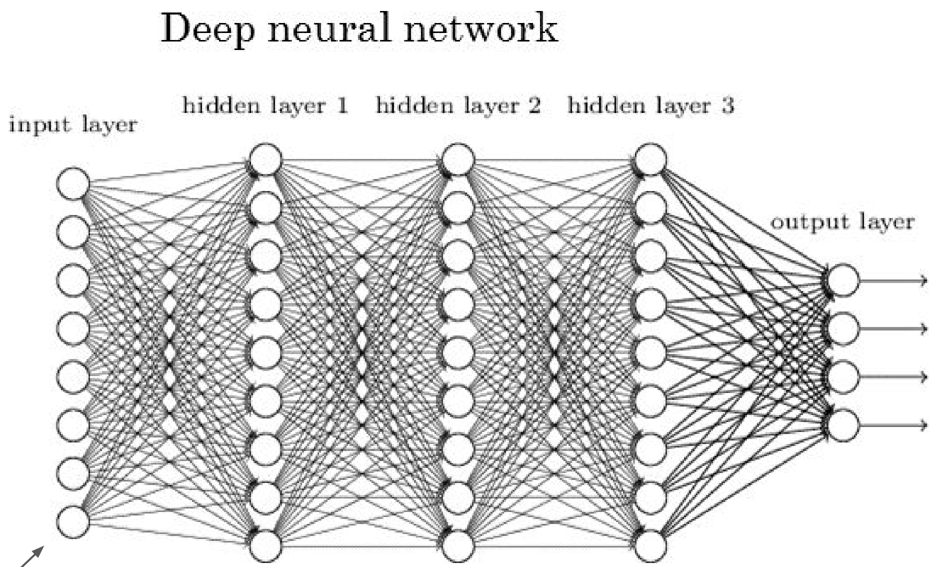
$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$$

What is deep learning?



http://www.rsipvision.com/exploring-deep-learning/

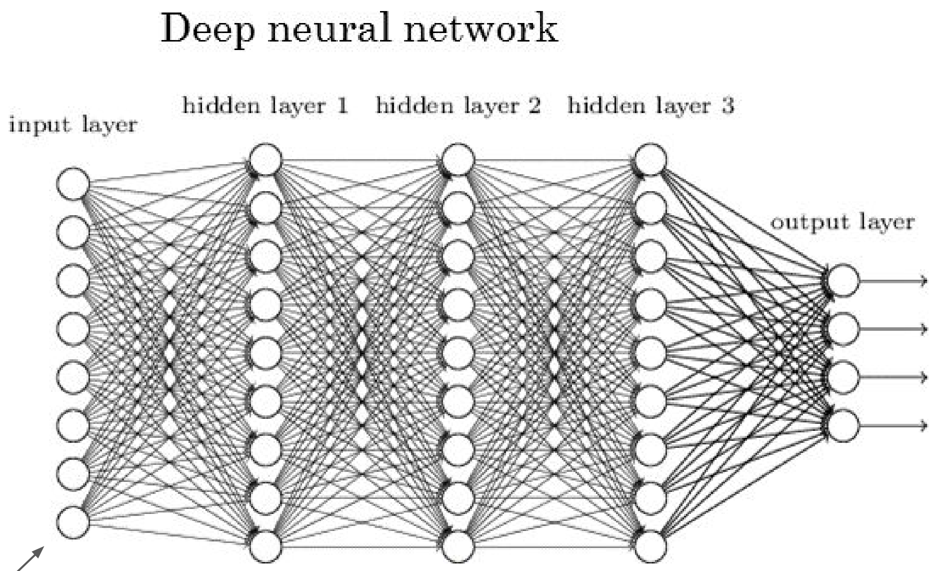
What is deep learning?



Input data

http://www.rsipvision.com/exploring-deep-learning/

What is deep learning?

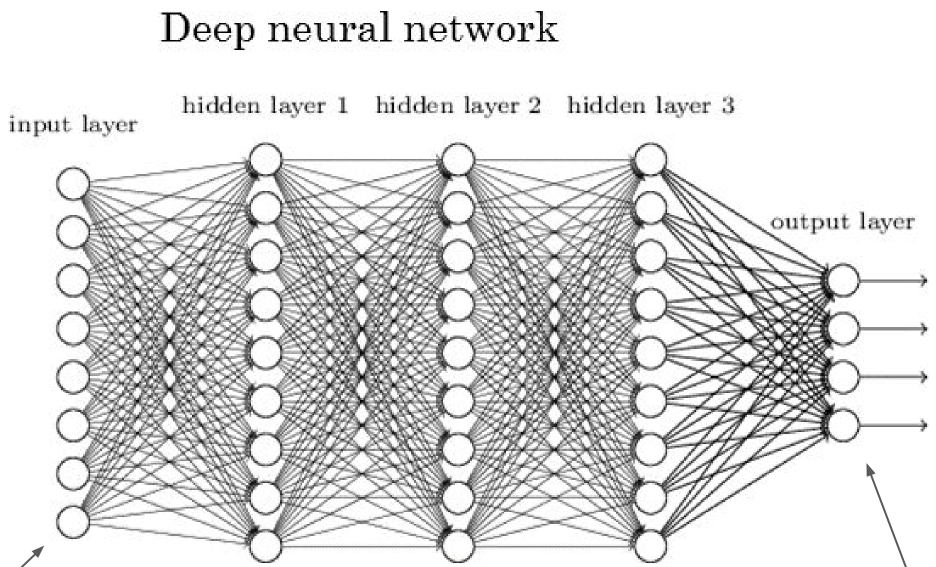


Input data

Data Transformations

http://www.rsipvision.com/exploring-deep-learning/

What is deep learning?



Input data

Data Transformations

Output - usually some shallow machine learning layer (logistic regression, linear svm, linear regression).

http://www.rsipvision.com/exploring-deep-learning/

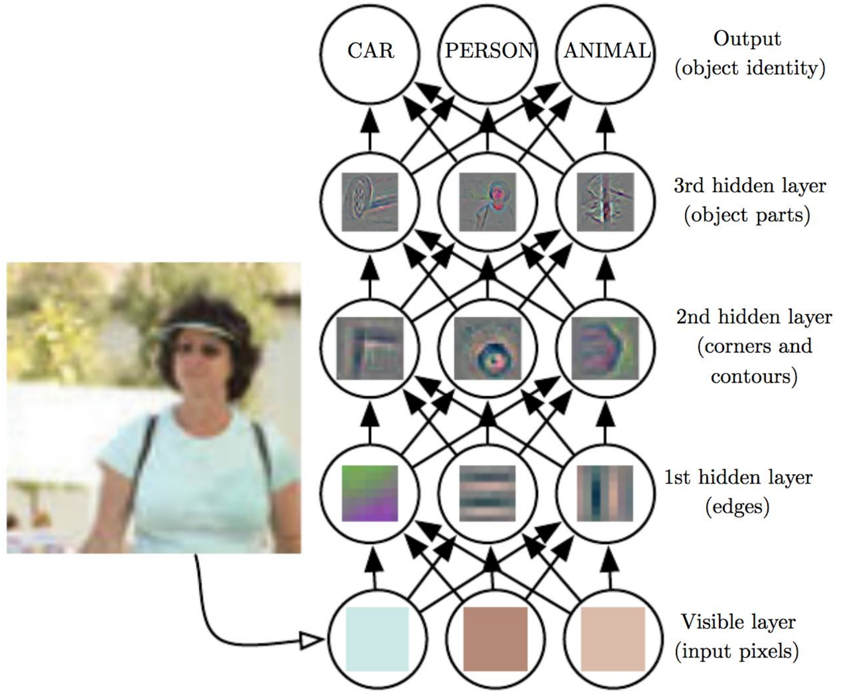
Representation learning

Deep learning presents us with a unique philosophy for machine learning. Instead of learning the mapping from our features to an output we are also learning the best representation of our features.

Each layer of a neural network introduces feature representations that are expressed in terms of other more simple representations.

The process has multiple steps, with each layer building on the representations created by the previous layer.

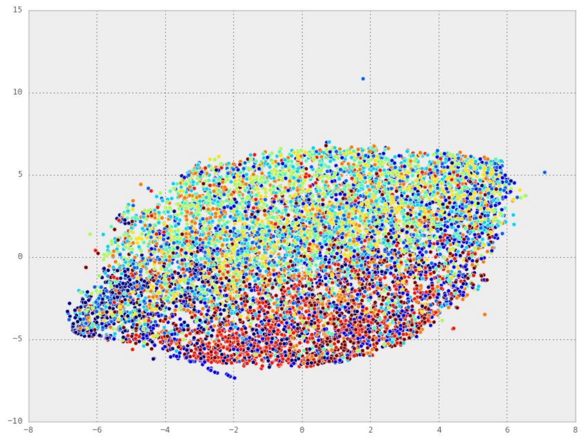
Representation Learning

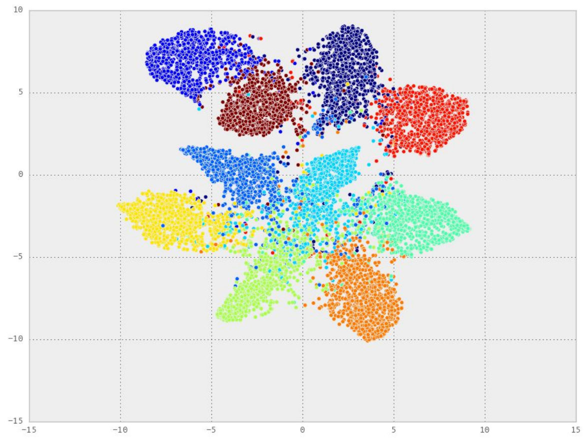

CAR
PERSON
ANIMAL
Output (object identity)
3rd hidden layer (object parts)
2nd hidden layer (corners and contours)
1st hidden layer (edges)
Visible layer (input pixels)

Deep Learning Book (2016)

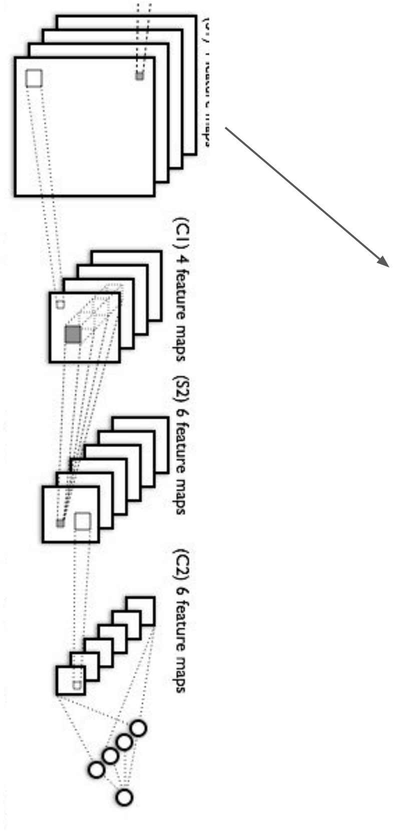
Representation Learning

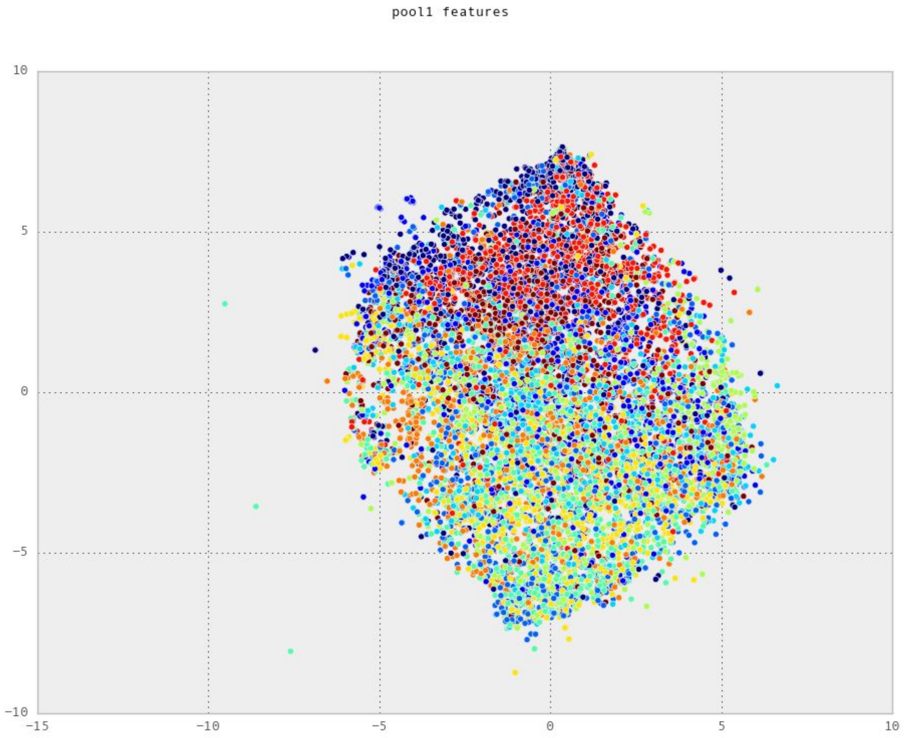
Here is a test with natural images. Unsupervised clustering on the raw images and then again on the features extracted from the final layer of a deep neural network.



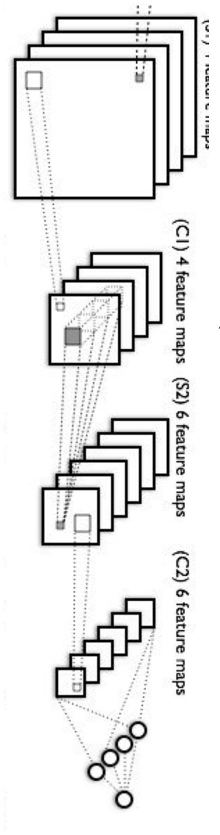


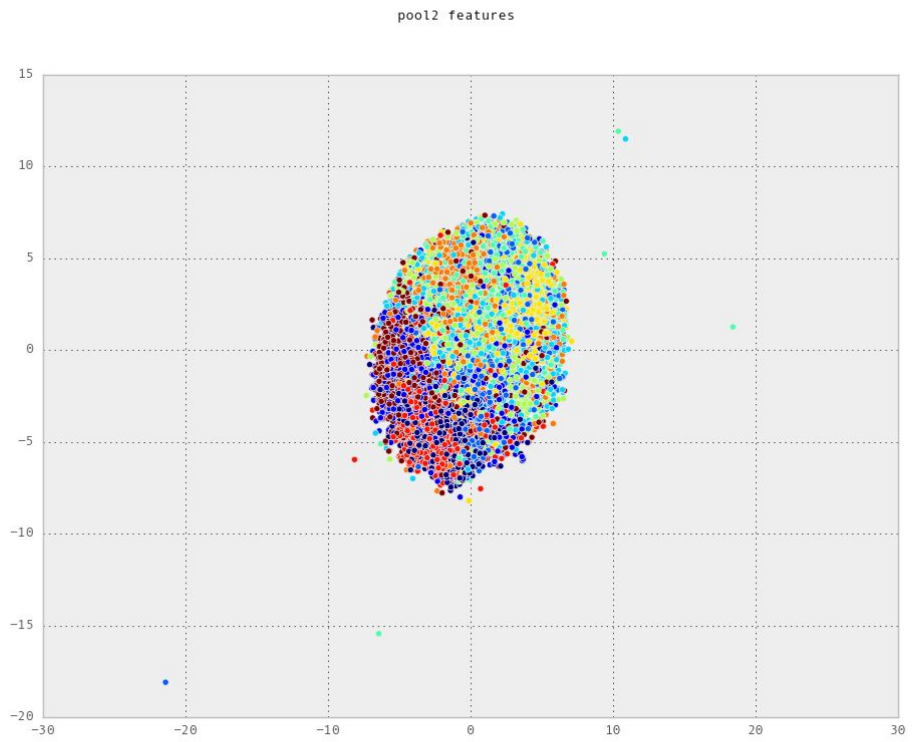
Representation Learning



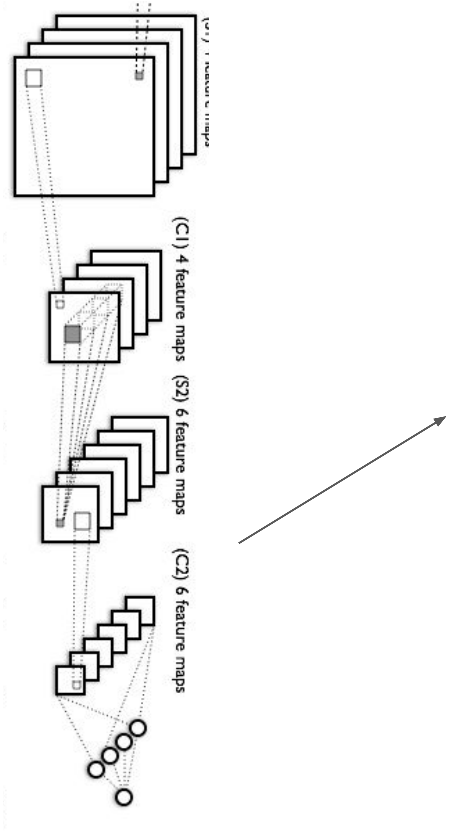


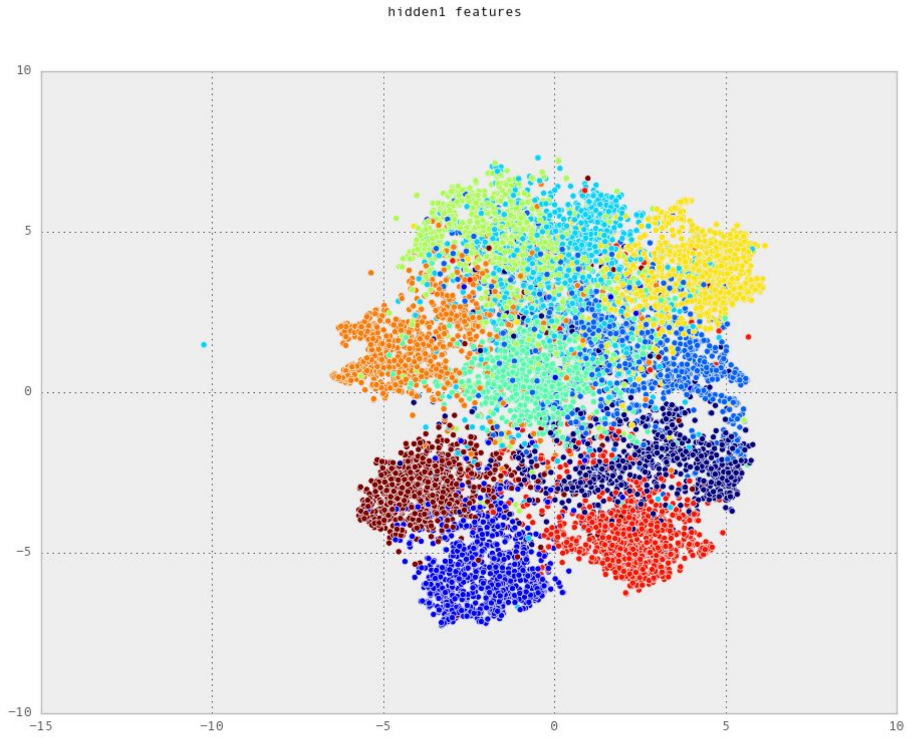
Representation Learning



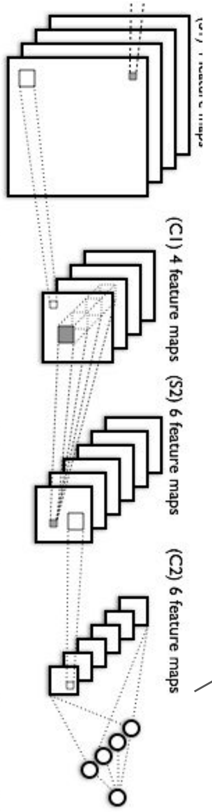


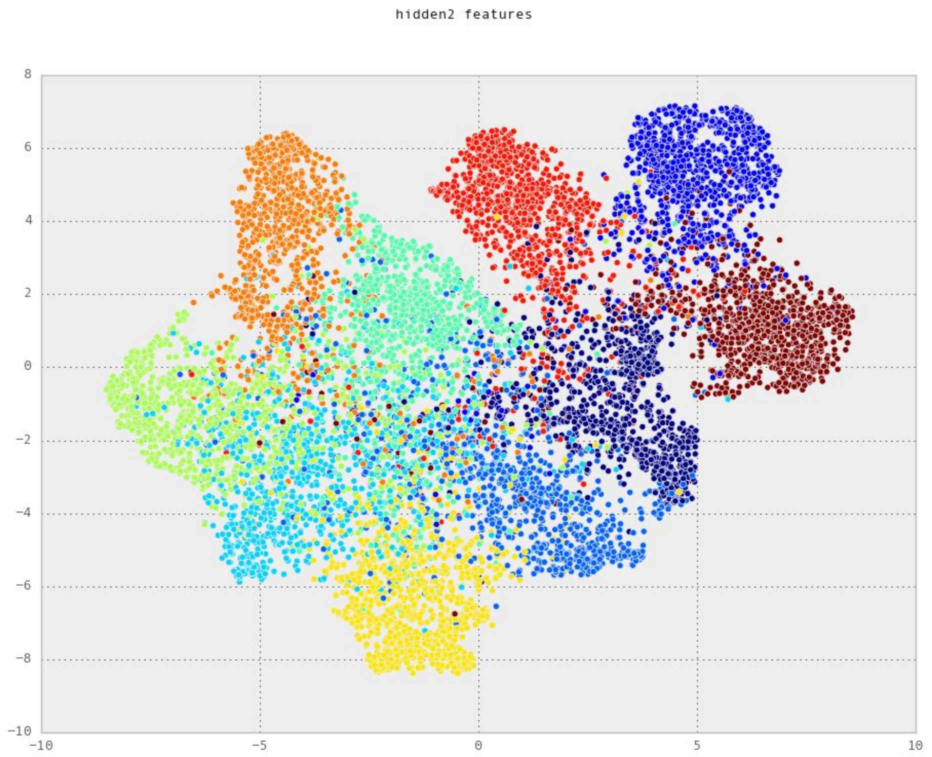
Representation Learning





Representation Learning





Representation learning

This is an interesting observation!

The ‘deep’ part of this neural network is essentially working to try and make the problem easier.

We will leverage this property to very quickly and effectively make use of the power of deep learning without the headache of training the networks from scratch.

Applied Deep Learning

Most applied deep learning work revolves around two strategies. Both of which utilize neural networks that have already been trained

1. We can take a pretrained network and finetune it on our specific problem.
    a. Ex. A computer vision model already knows how to ‘see’ we just need to finetune it to see whatever our specific problem requires.
2. Using a pretrained network and extract its internal representations of a dataset to use as features for a model.

Fine Tuning a Network

To finetune a network we typically choose a network that was already trained on a similar problem to our own.

We then train the network in much the same way that it was originally trained with a few differences.

1. Use a much smaller learning rate
2. Only train for a handful of iterations

Feature Extraction

We will use these networks as feature extractors to represent our data as dense information rich vectors.

We can vectorize the following quite easily:

1. Images
   a. Using Convolutional Neural Networks trained to classify images
2. Words
   a. Using neural networks trained to predict words given their context
3. Sentences
   a. Using neural networks that are trained to reconstruct sentences given their context

Applied Deep Learning Practice

In the notebooks today we will do:

1. Reverse Image search
2. Semantic Document Matching
3. Image Classification

Code Introduction

Theano and Tensorflow

Theano and TensorFlow are the two main frameworks in python for creating deep learning systems.

Theano came first, from U.Montreal, and it should be thought of as a symbolic math library with some properties that make it very useful for deep learning (automatic differentiation and gpu support for matrix algebra).

TensorFlow, from Google, is much newer but is very similar to Theano. It’s still a symbolic math library but learned a bit from some of Theano’s problems.

Both libraries are very low level, need to define every function yourself. Makes work tedious.

Keras

“Keras is a minimalist, highly modular neural networks library, written in Python and capable of running on top of either TensorFlow or Theano. It 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.

Use Keras if you need a deep learning library that:

Allows for easy and fast prototyping (through total modularity, minimalism, and extensibility).

Supports both convolutional networks and recurrent networks, as well as combinations of the two.

Supports arbitrary connectivity schemes (including multi-input and multi-output training).

Runs seamlessly on CPU and GPU.”

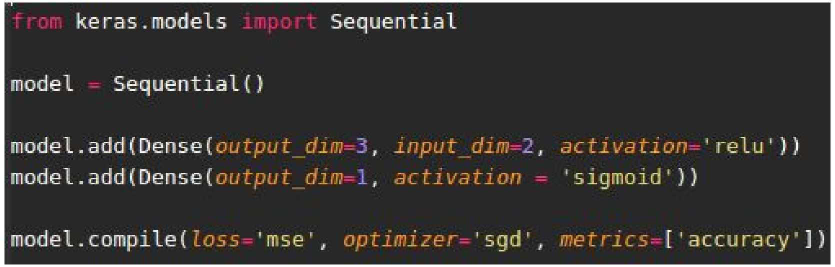
Source: https://keras.io/

Keras

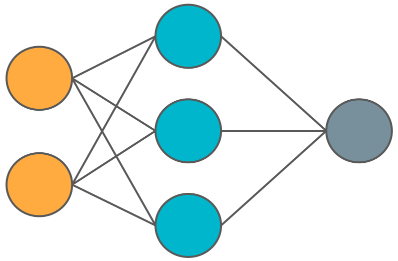
Sequential api?

Functional api?

Sequential Models



```
from keras.models import Sequential

model = Sequential()

model.add(Dense(output_dim=3, input_dim=2, activation='relu'))
model.add(Dense(output_dim=1, activation = 'sigmoid'))

model.compile(loss='mse', optimizer='sgd', metrics=['accuracy'])
```



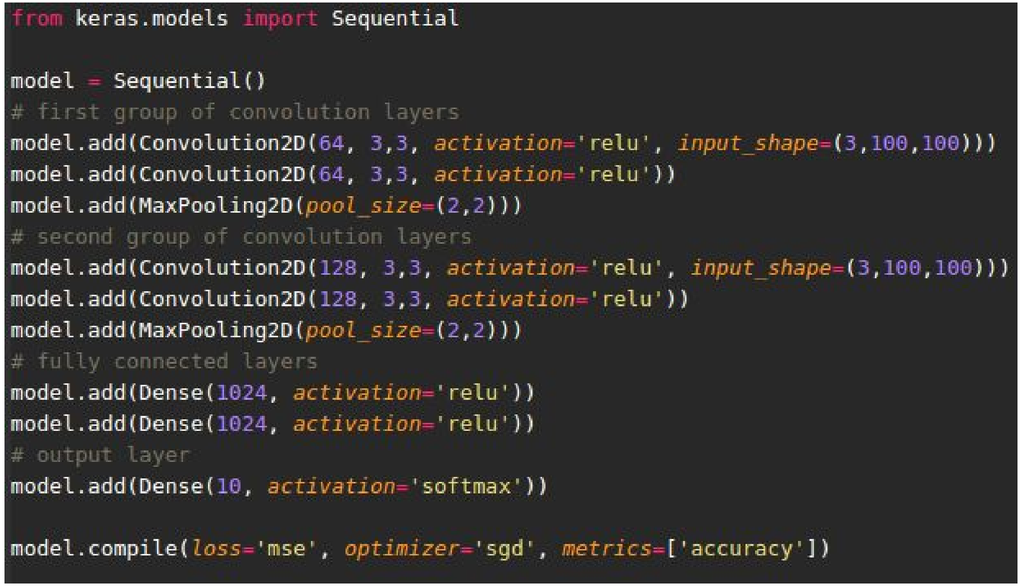
With the sequential model you start by declaring your model as a sequential. That gives you a blank canvas to start working.

From there you can add anything you want using model.add(). The data will flow through the model in the order that you declare each layer.

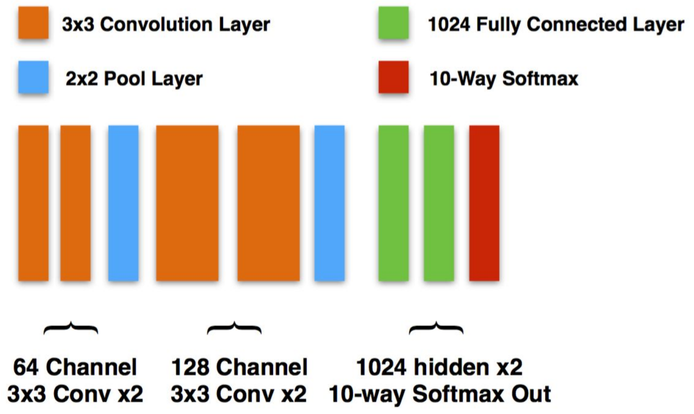
The example on the left builds the small neural network that we looked at the in the beginning.

Sequential Models

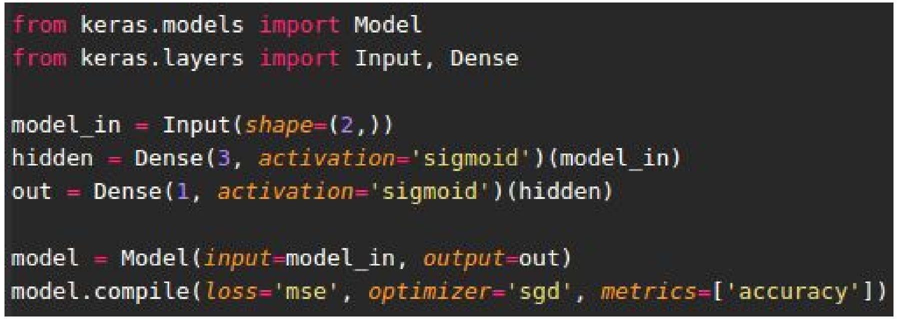
A slightly more complicated and modern looking ConvNet.



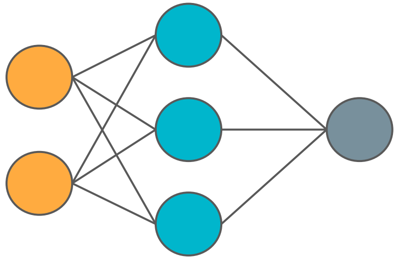
```
from keras.models import Sequential

model = Sequential()
# first group of convolution layers
model.add(Convolution2D(64, 3,3, activation='relu', input_shape=(3,100,100)))
model.add(Convolution2D(64, 3,3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
# second group of convolution layers
model.add(Convolution2D(128, 3,3, activation='relu', input_shape=(3,100,100)))
model.add(Convolution2D(128, 3,3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
# fully connected layers
model.add(Dense(1024, activation='relu'))
model.add(Dense(1024, activation='relu'))
# output layer
model.add(Dense(10, activation='softmax'))

model.compile(loss='mse', optimizer='sgd', metrics=['accuracy'])
```



Functional Models



```
from keras.models import Model
from keras.layers import Input, Dense

model_in = Input(shape=(2,))
hidden = Dense(3, activation='sigmoid')(model_in)
out = Dense(1, activation='sigmoid')(hidden)

model = Model(input=model_in, output=out)
model.compile(loss='mse', optimizer='sgd', metrics=['accuracy'])
```

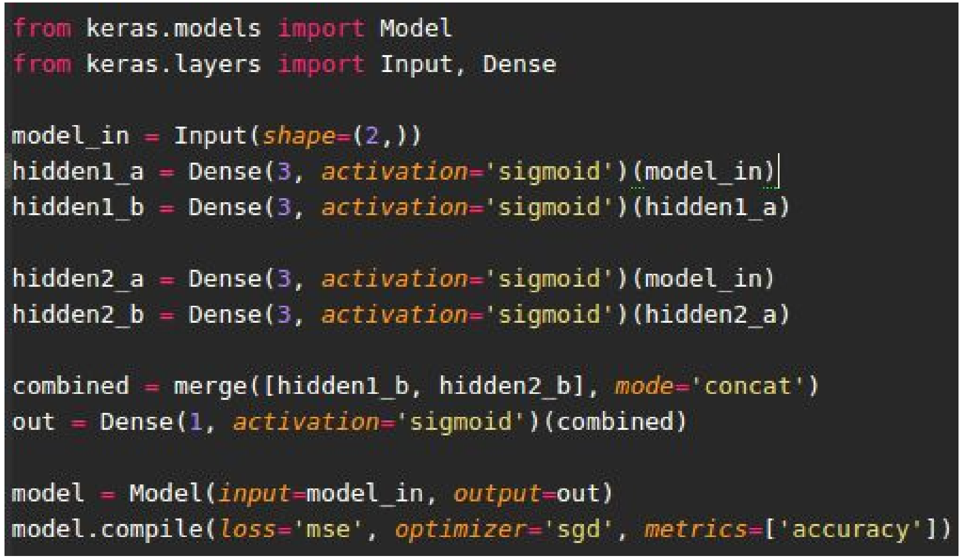


With the functional model you give each computational layer a name instead of just stacking them up with model.add().

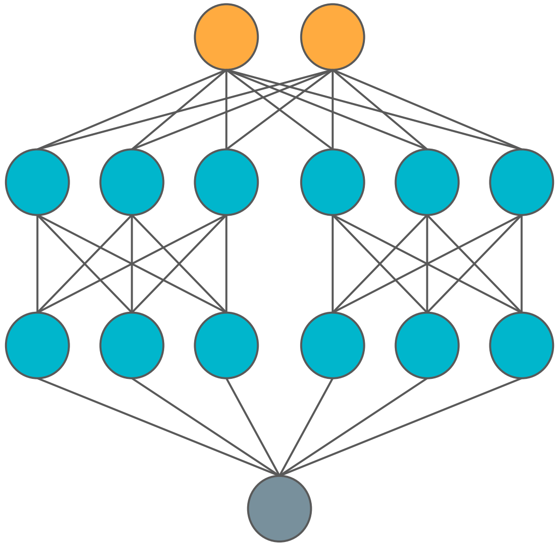
This is key, because you also explicitly provide the input to each layer. This allows you more freedom to try some different architectures with no extra effort to hack the sequential models.

For example some architectures could have multiple independent branches of neurons coming off of the input. This method makes setting that up a breeze.

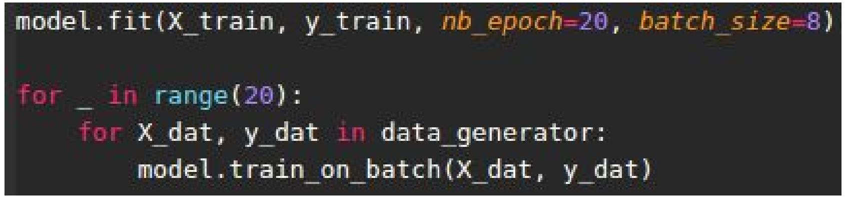
Functional Models



```
from keras.models import Model
from keras.layers import Input, Dense

model_in = Input(shape=(2,))
hidden1_a = Dense(3, activation='sigmoid')(model_in)
hidden1_b = Dense(3, activation='sigmoid')(hidden1_a)

hidden2_a = Dense(3, activation='sigmoid')(model_in)
hidden2_b = Dense(3, activation='sigmoid')(hidden2_a)

combined = merge([hidden1_b, hidden2_b], mode='concat')
out = Dense(1, activation='sigmoid')(combined)

model = Model(input=model_in, output=out)
model.compile(loss='mse', optimizer='sgd', metrics=['accuracy'])
```



Using Models

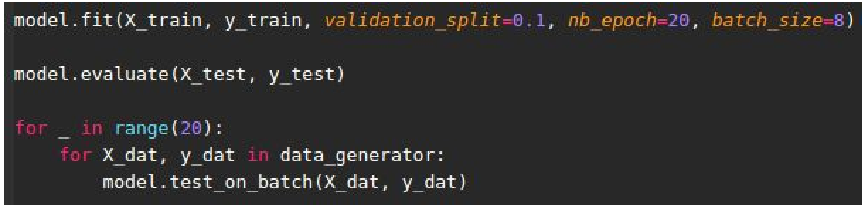


```
model.fit(X_train, y_train, nb_epoch=20, batch_size=8)

for _ in range(20):
    for X_dat, y_dat in data_generator:
        model.train_on_batch(X_dat, y_dat)
```

Training a network is an iterative process, so basically we'll be creating loops that stream the data into each model.

We can use Keras built in methods or create our own. The 'data_generator' would just be some function that would loop through the data, dividing it up into batches to stream into the model.

Using Models

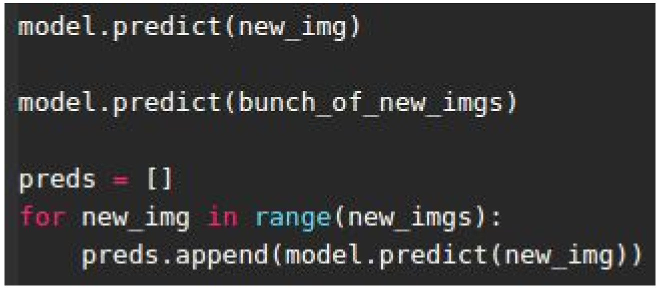


```
model.fit(X_train, y_train, validation_split=0.1, nb_epoch=20, batch_size=8)

model.evaluate(X_test, y_test)

for _ in range(20):
    for X_dat, y_dat in data_generator:
        model.test_on_batch(X_dat, y_dat)
```

With .fit() we don't have to split the data into training and testing before using it. Just tell Keras what percentage of data we want to use to evaluate the model with.

With .evaluate() we split the data before training and use it the same way that we use .fit(). Except this time it won't update parameters it will just return evaluation metrics.

Using Models



```
model.predict(new_img)

model.predict(bunch_of_new_imgs)

preds = []
for new_img in range(new_imgs):
    preds.append(model.predict(new_img))
```

Each model has a .predict(), it can be used in a variety of ways. Similar to how training and evaluating had a variety of methods, except each one of these will be called with .predict().

You can pass a single new point of data, a numpy array of new data.

You could also loop through the data and make a prediction on each data point.

Other Frameworks

Torch (Lua)

http://torch.ch

MXNet (everything?)

https://github.com/dmlc/mxnet

Theano (Python)

http://deeplearning.net/software/theano/

TensorFlow (Python, C++)

https://www.tensorflow.org