CO Summer School Central: Introduction to neural networks with Python II

Erik Spence

SciNet HPC Consortium

27 June 2018

SciNet compute • calcul CANADA

Material for this class

The slides, code and data for this class can be found here:

`https://scinet.courses/451`

All the material for the Summer School can be found here:

`https://scinet.courses/438`

This is also the link to confirm your attendance for this class. Make sure that you confirm within 15 minutes of the end of the class; note that you can only confirm if you are registered for the class.

SciNet compute • calcul CANADA

Ask for help!

The team at SciNet is here to help you advance your research, whether you use SciNet systems or not.

courses@scinet.utoronto.ca

support@scinet.utoronto.ca

Ask for help! Even if it's six months from now!

SciNet compute • calcul CANADA

About this class

The purpose of this class is to continue our introduction of neural network programming in Python. Some notes about the class, in case you missed the first:

- The material is transitioning from introductory to intermediate. If you are already using neural networks it's likely you are already familiar with much of this content.
- We will be using Python 3.X.
- We will be using the Keras neural-network programming framework. You may use whichever back end you like.
- You will need the usual machine learning packages: numpy, matplotlib, scikit-learn.
- You'll obviously also need a back end (Theano, Tensorflow, CNTK, *etc.*) and Keras installed.

Ask questions!

SciNet compute • calcul CANADA

A review of last class

What did we cover last class?

- We discussed the "neuron", which takes input data, multiplies it by some weights, adds a bias, and processes the result through an activation function (sigmoid, tanh, relu, *etc.*).
- We discueed the "neural network", which is simply a collection of neurons, organized in layers, which pass data from one layer to another, and are used to categorize the input data.
- We introduced the backpropagation algorithm, which is used, along with stochastic gradient descent, to train the neural network.
- We introduced neural network programming frameworks, which are used to simplify the development of neural networks. We switched to programming our networks in the Keras NN framework.

These concepts and tools form the foundation of neural network programming.

SciNet compute • calcul CANADA

The status of our MNIST problem

We were working on the MNIST (hand-written digit) data set problem. Our story so far:

- We flattened the input data from (28 x 28) to (1 x 784).
- The output data was one-hot-encoded into a 10-element vector.
- We built a neural network, using Keras, consisting of an input layer, a hidden layer, and an output layer, to classify the MNIST data:
  - We used the tanh function as the activation for the hidden layer.
  - We used the softmax function as the activation for the output layer.
- We used cross entropy as the cost function, and stochastic gradient descent (Adam) as the optimization algorithm.
- We trained on the full data set, and achieved an accuracy of 93% on the test data.

We can do better, but it requires a different approach.

SciNet compute • calcul CANADA

What next?

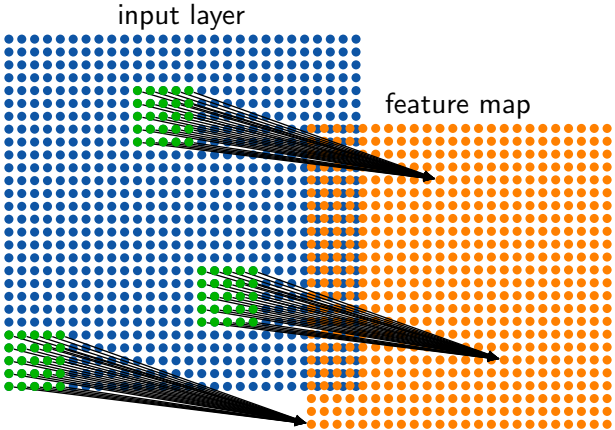
What we've done so far is pretty good, but it's not going to scale well.

- These are small images, and only black-and-white.
- Imagine we had a more-typical image size (200 x 200) and 3 colours? Now we're up to 120,000 input parameters.
- We need an approach that is more efficient.
- A good place to start would be an approach that doesn't throw away all of the spatial information.
- The data is (28 x 28), not (1 x 784).
- We should redesign our network to account for the spatial information. How do we do that?
- The first step called a Convolutional Layer. This is the bread-and-butter of all neural network image analysis.

SciNet compute • calcul CANADA

Convolutional layers: feature maps

Create a set of neurons that, instead of using all of the data as input, only takes input from a small area of the image. This set of neurons is called a "feature map".



Feature maps

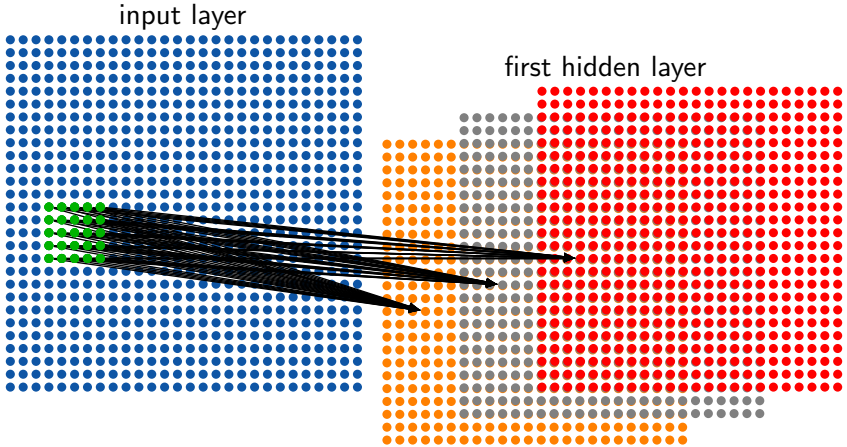
Some notes about feature maps.

- Notice that the feature map is smaller (24 x 24) than the input layer (28 x 28).
- The size of the feature map is partially set by the 'stride', meaning the number of pixels we shift to use as the input to the next neuron. In this case I've used a stride of 1.
- The **weights and biases are shared by all the neurons in the feature map**.
- Why? The goal is to train the feature map to recognize a single feature in the input, regardless of its location in the image.
- Consequently, it makes no sense to have a single feature map as the first hidden layer. Rather, multiple feature maps are used as the first layer.
- Feature maps are also called "filters" and "kernels".

SciNet compute • calcul CANADA

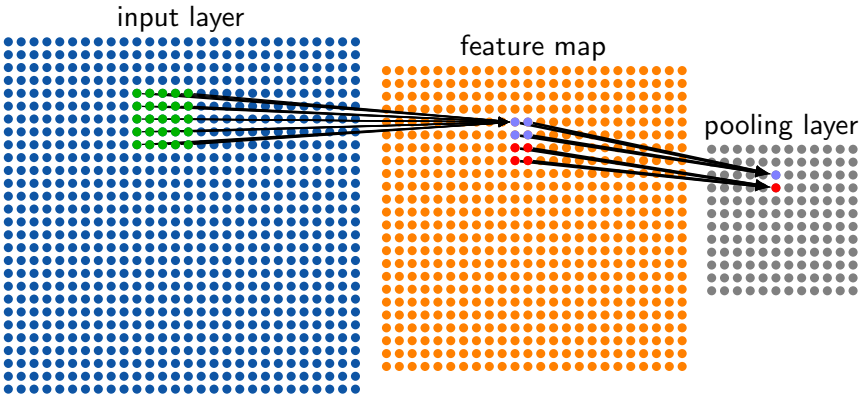
Convolutional layers, continued more

The first hidden layer, a "convolutional layer", consists of multiple feature maps. The same inputs are fed to the neurons in different feature maps.



Pooling layers

Each feature map is often followed by a "pooling layer".



In this case, 2 x 2 feature map neurons are mapped to a single pooling layer neuron.

Pooling layers, continued

Some notes about pooling layers.

- The purpose of a pooling layer is to reduce the size of the data, and thus the number of free parameters in the network.
- The reduction in data also helps with over-fitting.
- Rather than use one of the activation functions we've already discussed, pooling layers use other functions.
- These functions do not have free parameters (weights and biases) in them which need to be fit. They are merely functions which operate on the input.
- The most common function used is 'max', simply taking the maximum input value.
- Other functions are sometimes used, average pooling, L2-norm pooling.

SciNet compute • calcul CANADA

Re-prepping the data

First let us gather the data again, and one-hot-encode the target data.

```
In [1]:
In [1]: from keras.datasets import mnist
Using Theano backend.
In [2]:
In [2]: import keras.utils as ku
In [2]:
In [2]: (x_train, y_train), (x_test, y_test) =
        mnist.load_data()
In [3]:
In [3]: x_train.shape
Out[3]: (60000, 28, 28)
In [4]:
In [4]: y_train = ku.to_categorical(y_train, 10)
In [5]: y_test = ku.to_categorical(y_test, 10)
In [6]:
```

2D data formatting

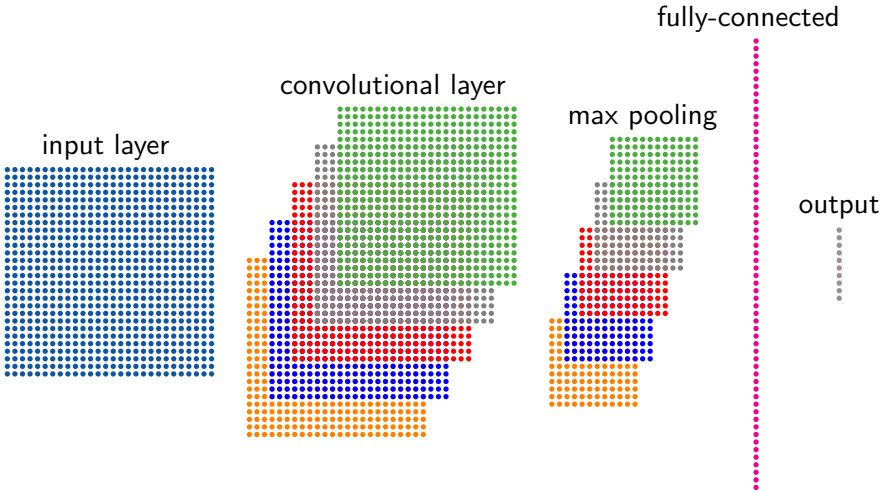
Generally, 2D images are actually 3D, to deal with colours. We need to add this third dimension to the data, since the Convolutional layers are expecting it.

Where the third dimension (the "channels") shows up in the dimensionality is given by the "image_data_format" function, which is part of keras.backend.

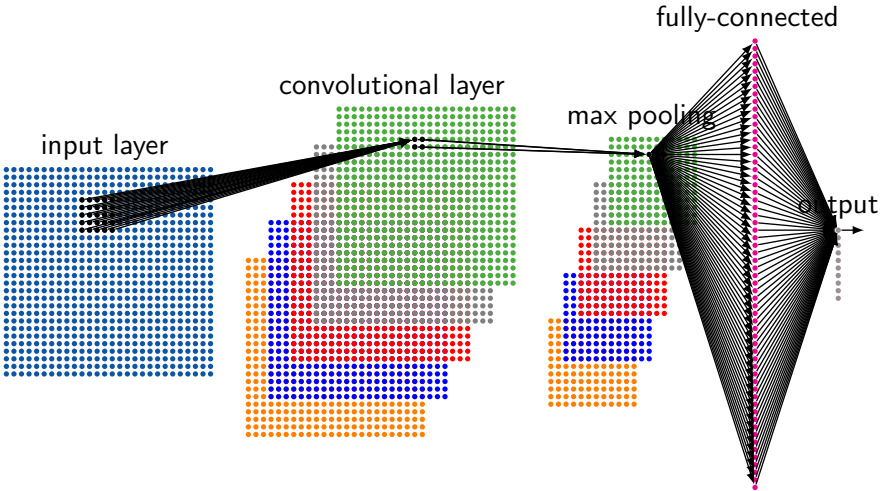
Having the channels at the end seems to have become standard.

```
In [6]:
In [6]: import keras.backend as K
In [7]:
In [7]: K.image_data_format()
Out[7]: 'channels_last'
In [8]:
In [8]: x_train.shape
Out[8]: (60000, 28, 28)
In [9]:
In [9]: x_train = x_train.reshape(60000, 28, 28, 1)
In [10]: x_test = x_test.reshape(10000, 28, 28, 1)
In [11]:
In [11]: x_train.shape
Out[11]: (60000, 28, 28, 1)
In [12]:
```

Our network, latest version



Our network, latest version



SciNet compute • calcul CANADA

Our network revisited again

```
# model7.py
import keras.models as km, keras.layers as kl

def get_model(numfm, numnodes):

  model = km.Sequential()
  model.add(kl.Conv2D(numfm, kernel_size = (5, 5),
    input_shape = (28, 28, 1), activation = "relu"))

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

  model.add(kl.Flatten())
  model.add(kl.Dense(numnodes, activation = "tanh"))
  model.add(kl.Dense(10, activation = "softmax"))
  return model
```

```
In [12]:
In [12]: import model7 as m7
In [13]:
In [13]: model = m7.get_model(20, 100)
In [14]:
```

The "Flatten" layer converts the 2D output to 1D, so that the fully-connected layer can handle it.

Our network revisited again, continued

```
In [14]: model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 20)        520
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 20)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2880)              0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               288100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params:  289,630
Trainable params:  289,630
Non-trainable params:  0
_________________________________________________________________
```

Our network revisited again, more

```
In [15]:
In [15]: model.compile(loss = "categorical_crossentropy", optimizer = "sgd",
...:           metrics = ['accuracy'])
In [16]:
In [16]: fit = model.fit(x_train, y_train, epochs = 30, batch_size = 100, verbose = 2)
Epoch 1/30
25s - loss:  0.4992 - acc:  0.8638
Epoch 2/30
25s - loss:  0.1973 - acc:  0.9466
⋮
Epoch 30/30
24s - loss:  0.0321 - acc:  0.9911
In [17]:
```

Our network revisited again, some more

Now check against the test data.

98.35%! Only 165 / 10000 wrong!

Not bad!

You can improve this even more by adding another convolutional layer-max pooling layer after the first pair.

```
In [17]:
In [17]: score = model.evaluate(x_test, y_test)
In [18]:
In [18]: score
Out[18]: [0.053592409740015862, 0.98350000000000004]
In [19]:
```

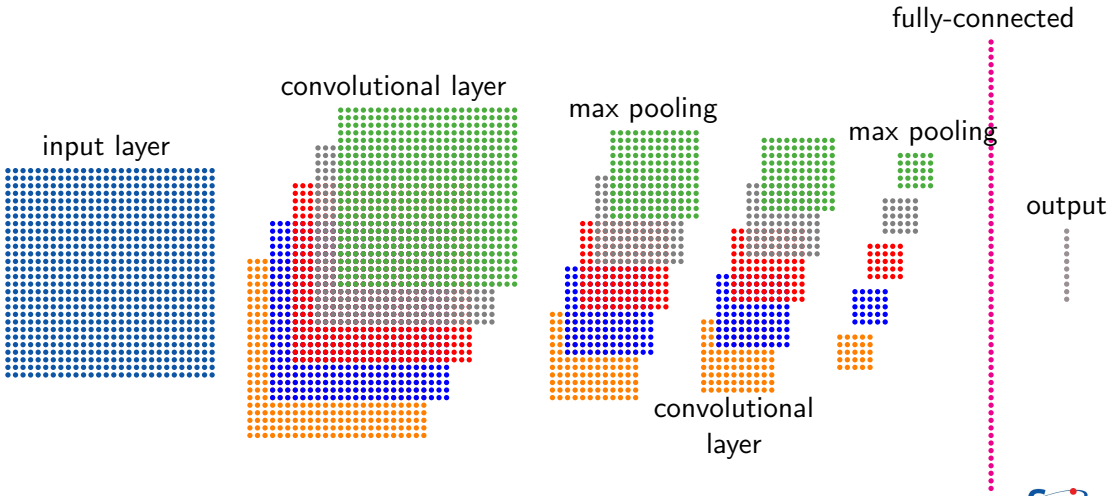
Notes on Convolutional Networks

The previous network is called a Convolutional Neural Network (CNN), and is quite common in image analysis.

- Often more than a single convolutional layer-pooling layer combination will be used.
- This will lead to improved performance, in this case.
- In practice people come up with all manner of combinations of convolutional, pooling and fully-connected layers in their networks.
- Trial-and-error is a good starting point. Again, hyperparameter optimization techniques should be considered. Reviewing the literature, you will find themes, but also much art.

SciNet compute • calcul CANADA

Our latest network, version 2



Our latest network, version 2, continued

```
# model8.py
import keras.models as km, keras.layers as kl

def get_model(numfm, numnodes, input_shape = (28, 28, 1)):

  model = km.Sequential()
  model.add(kl.Conv2D(numfm, kernel_size = (5, 5), input_shape = input_shape, activation = "relu"))
  model.add(kl.MaxPooling2D(pool_size = (2, 2), strides = (2, 2)))

  model.add(kl.Conv2D(2 * numfm, kernel_size = (3, 3), activation = "relu"))
  model.add(kl.MaxPooling2D(pool_size = (2, 2), strides = (2, 2)))

  model.add(kl.Flatten())
  model.add(kl.Dense(numnodes, activation = "tanh"))
  model.add(kl.Dense(10, activation = "softmax"))

  return model
```

Our latest network, version 2, summary

```
In [19]: import model8 as m8
In [20]: model = m8.get_model(20, 100)
In [21]:
In [21]: model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 20)        520
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 20)        0
conv2d_2 (Conv2D)            (None, 10, 10, 40)        7240
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 40)          0
flatten_1 (Flatten)          (None, 1000)              0
dense_1 (Dense)              (None, 100)               100100
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params:  108,870
Trainable params:  108,870
Non-trainable params:  0
```

Understanding successive Convolutional Layers

As you noticed, we previously had to change the input data to have dimension (28, 28, 1) rather than (28, 28).

- All convolutional layers assume that its input has a 'channel', which is a third dimension.
- For colour images the three colours of the image (RGB) are the three channels.
- The channel is usually put in the third dimension, though sometimes in the first.
- When the output of one convolutional layer is fed into another layer, the feature maps are the channels.
- How are the channels read by the feature maps?
- If the filter size is, say (3 x 3), and there 20 channels, as in this example, then the number of weights in a given feature map will be (3 x 3) x 20, plus 1 bias.
- Thus, the number of trainable parameters in the second convolutional layer is $(((3 \times 3) \times 20) + 1) \times 40 = 7240$.

SciNet compute • calcul CANADA

Our latest network, version 2, more

```
In [22]:
In [22]: model.compile(loss = "categorical_crossentropy", optimizer = "sgd",
...:           metrics = ['accuracy'])
In [23]:
In [23]: fit = model.fit(x_train, y_train, epochs = 100, batch_size = 100, verbose = 2)
Epoch 1/100
33s - loss:  0.7378 - acc:  0.7966
Epoch 2/100
33s - loss:  0.2010 - acc:  0.9486
⋮
Epoch 99/100
33s - loss:  0.0028 - acc:  0.9996
Epoch 100/100
32s - loss:  0.0028 - acc:  0.9996
In [24]:
```

Our latest network, version 2, even more

Now check against the test data.

99.1%! Only 88 / 10000 wrong! Not bad!

```
In [24]:
In [24]: score = model.evaluate(x_test, y_test)
In [25]:
In [25]: score
Out[25]: [0.028576645734044722, 0.99119999999999997]
In [26]:
```

Using GPUs

An important note. Graphical Processing Units (GPUs) are particularly good at running NN-training calculations.

data size	CPU only		CPU-GPU	
	epoch time	total time	epoch time	total time
50000	41 s	21 min 4 s	4 s	2 min 43s
250000	198 s	100 min	26 s	15 min

These numbers are for today's first network. These were run on a Power 8 CPU, and a P100 GPU.

Multi-GPU functionality is available in Keras running on TensorFlow, though it can be a bit of work to set up.

Deep Learning

What is Deep Learning?

- Quite simply: a neural network with many hidden layers.
- Our last network qualified as Deep Learning.
- Up until the mid-2000s neural network research was dominated by "shallow" networks, networks with only 1 or 2 hidden layers.
- The breakthrough came in discovering that it was practical to train networks with a larger number of hidden layers.
- But it only became practical with the advent of sufficient computing power (GPUs) and easily-accessible huge data sets.
- State-of-the-art networks today can contain dozens of layers.

SciNet compute • calcul CANADA

Dealing with sequential data

So far we've focussed on solving problems that involve getting the input data all at once, such as images.

But suppose we are given information that is sequential instead?

- Timeseries data, predicting future trends.
- Natural Language Processing (NLP), voice recognition, language translation.
- Next-word predictions, question answering.
- Handwriting generation.

Generally these data are processed as the data arrives, or generate an output based on a sequence of inputs, rather than getting the data all at once. This requires a different sort of network.

SciNet compute • calcul CANADA

Dealing with sequential data, continued

Sequential data is complicated by the long-term relationships that exist between data points.

Consider the following sentence:

> *I live in Canada. I speak english and ...*

We can all guess what the next word in the sentence probably is. But the information which we use to determine that word is given in the sentence before.

For a neural network to be able to predict the next word, it must remember that we're talking about Canada. This information must stay in the network somehow. The network needs to 'remember'.

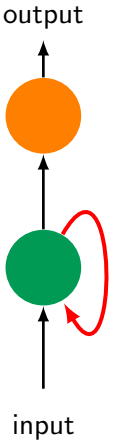
SciNet compute • calcul CANADA

Recurrent neural networks

In most applications dealing with sequential data, the network needs a means of "remembering" previous data.

To this end, the output of a node is fed back into the network, as part of the input. These are called 'recurrent' neural networks. (Not to be confused with 'recursive' neural networks.)

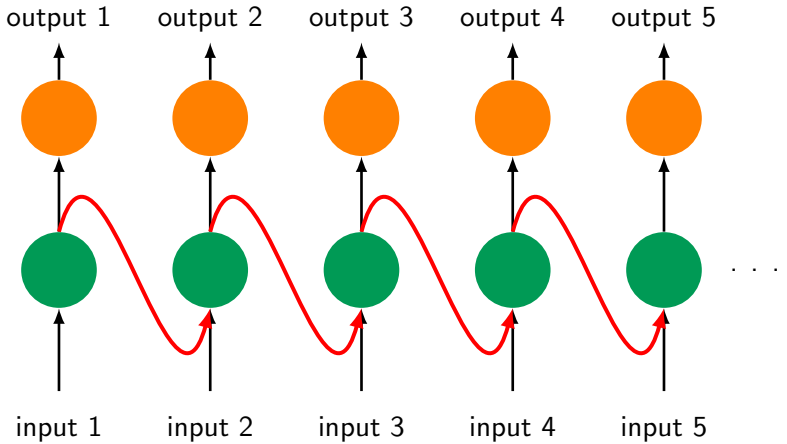
This allows the network to have 'memory', in a sense.



Recurrent neural networks, continued

This is what the previous network looks like when it's "unrolled". All the orange (and green, respectively) nodes are the same node.

The chain-like structure of these networks naturally lends itself to dealing with sequences and lists.



Backpropagation through time

How do you perform backpropagation on such a network?

- Recall that when we use Stochastic Gradient Descent to train our network, we need the derivatives of the cost function with respect to the weights and biases.
- The obvious problem is that the hidden layer references itself, and thus the references in the partial derivatives go backward forever in time, as seen in the last slide.
- While this is true, one must observe that, if my input sequence is of length n, then the unrolled version of the network only needs $n + 1$ layers to calculate the gradient.
- So while it may look scary at first, this is actually fairly straightforward.

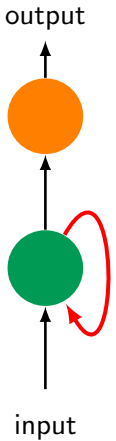
As with all such problems, backpropagation through time is done automatically in Keras.

SciNet compute • calcul CANADA

RNNs, continued more

Simple recurrent neural networks suffer from an instability.

- Because of the feedback on itself, it naturally acts like an amplifier.
- Depending on the activation function, you either get the vanishing gradient problem (sigmoid), or the exploding gradient problem (rectifier linear units).
- Regularization can help in these cases.
- However, the more-common approach is to switch to a different recurrent network architecture, one which is capable of actively suppressing the instability.
- The most common of these is known as Long Short Term Memory networks (LSTMs), though others are also used.
- LSTMs have been trained to do some amazing things.

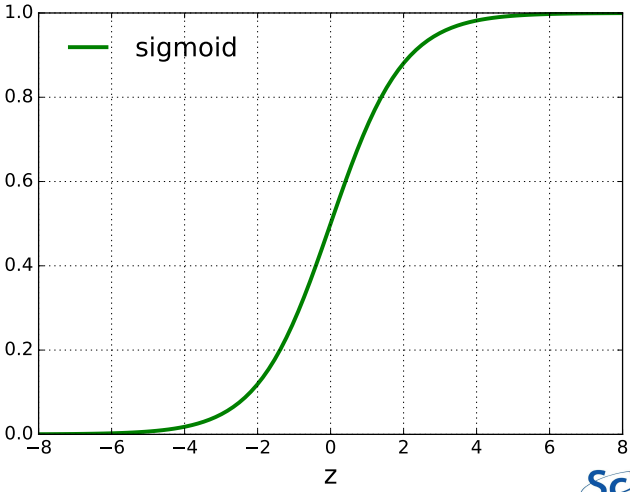


SciNet compute • calcul CANADA

Recall the sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

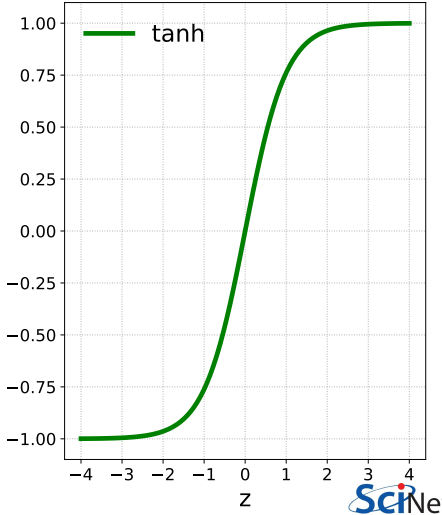
This function is ideal for 'gates'.



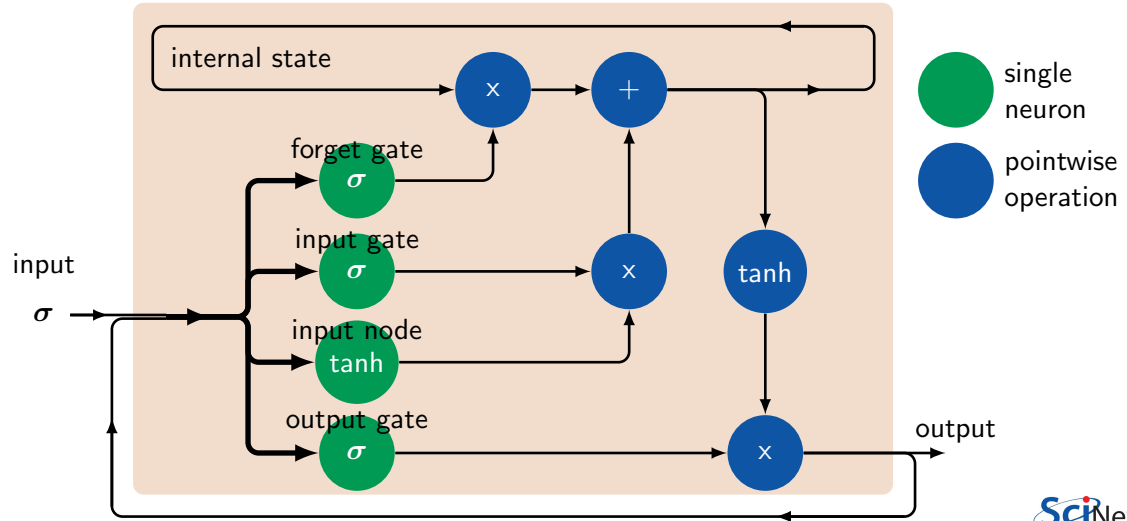
Recall the tanh function

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

In LSTMs these are used to scale the output to between -1 and 1.



Long Short Term Memory networks, memory cells



Notes about LSTM memory cells

Some notes about these memory cells.

- The 'input node' is a standard input node. These typically use a tanh activation function, though others can be used.
- How much of the input is added to the 'internal state' is controlled by the 'input gate.'
- The 'forget gate' controls how much of the internal state we're keeping, based on the input.
- The 'output gate' controls how much of the internal state is output.
- The internal state is put through a tanh function before output. This is optional, and is only done to put the output in the same range (-1 to 1) as a typical hidden layer. Some implementations use other functions such as rectifier linear units.

SciNet compute • calcul CANADA

Notes about LSTMs

Some notes about LSTMs in networks.

- Each 'memory cell' is treated like a single neuron in a hidden layer. Typically there are many such cells in such a layer.
- In the Keras implementation of LSTMs, not only is the output of a single LSTM cell concatenated to its input, the output of all the LSTM cells in the layer are concatenated to the input.
- These networks are trained in the usual way, using Stochastic Gradient Descent and Backpropagation, as with other neural networks.
- These have been used in language translation, voice recognition, handwriting analysis, next-letter prediction, and many many other applications.

SciNet compute • calcul CANADA

LSTM example

One common application of LSTMs is text prediction. Let's use an LSTM network to create a recipe.

- We will use the recipe data set, which is a text file containing 4869 recipes.
- We take the recipe data set, as a single file, and analyse it to find all unique words.
- We then one-hot-encode the words in the data set using our word list.
- We then break the data set into 50-word one-hot-encoded chunks ("sentences").
- We will then train the network:
  - the input will be the 50-word-encoded chunks.
  - the target will be the next word in the data set.
- Once the network is trained we can feed the network a random sentence as a seed, and it will use that sentence to generate new words, until we have a new recipe.

SciNet compute • calcul CANADA

LSTM example, the data

```
ejspence@mycomp ~> head -24 allrecipes.txt
                      Almond Liqueur

  Amount  Measure       Ingredient -- Preparation Method
--------  ------------  --------------------------------
       3           cup  sugar
   2 1/4           cup  water
       3                lemons; the rind -- finely grated
       1         quart  vodka
       3    tablespoon  almond extract
       2    tablespoon  vanilla extract

Combine first 3 ingredients in a Dutch oven; bring to a boil. Reduce
heat and simmer 5 minutes, stirring occasionally; cool
completely. Stir in remaining ingredients; store in airtight
containers.

Yield: about 6 1/2 cups.

----------------------------------------------------------

                      Cafe Mexicano
  Amount  Measure       Ingredient -- Preparation Method
--------  ------------  --------------------------------
```

One-hot encoding

One way of portraying sentences is one-hot encoding. In this representation, all words are given an index in a vector of length num_words. The word gets a '1' when the word occurs and a '0' when it doesn't. The sentence then consists of an array of sentence_length rows and num_words columns.

Consider the sentence "The dog is in the dog crate."

The number of unique words is 5. Each word gets its own index: {the: 0, dog: 1, is: 2, in: 3, crate: 4}.

The sentence above can then be represented by the matrix to the right, with dimensions (sentence_length, num_words).

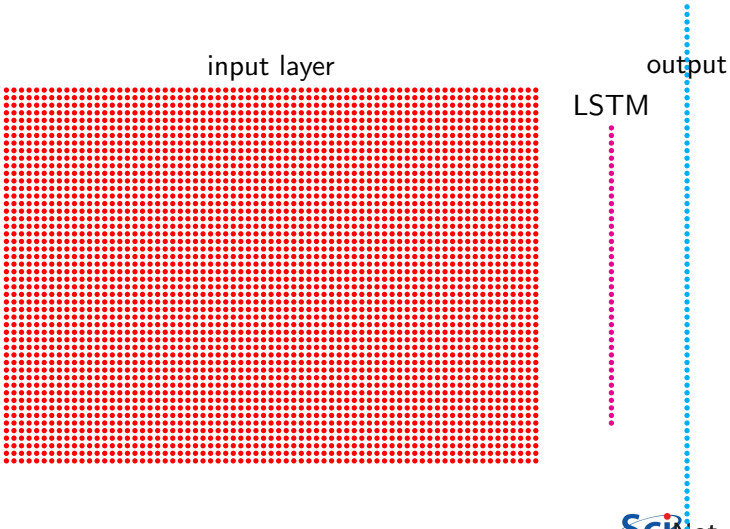
$$\begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

SciNet compute • calcul CANADA

Our LSTM network

The network is simple:

- The input has dimensions (sentence_length, n_words)
- sentence_length = 50
- n_words = number of unique words in the data.
- The LSTM layer has 256 nodes.
- The output layer is fully-connected, of length n_words.



LSTM example, data preprocessing

Before being used, the data needs to be preprocessed so that the network has an easier time learning. How is it preprocessed?

- Put spaces around the punctuation, so that "word!" becomes "word !" This is done so that "word" and "word!" are not counted as two distinct words.
- Do the same with new line symbols.
- Treat multiple dash combinations as words, put spaces around single dashes.
- Change all entries to lower case.
- Split on spaces.
- Separate all new line characters from words, so that "word" and "word\n" are not considered distinct words.
- Remove all spaces from the data (this was key).
- Remove all words that show up less than 5 times.

SciNet compute • calcul CANADA

LSTM example, learning code

```
# Learn_Recipes.py
import numpy as np; import shelve
import keras.models as km, keras.layers as kl

# Read the data set, preprocess (not shown).
f = open('allrecipes.txt')
corpus = f.read();     f.close()

# Create the list of words.
words = sorted(list(set(corpus)))
n_words = len(words)

# Create word-index encodings.
encoding = {w: i for i, w in enumerate(words)}
decoding = {i: w for i, w in enumerate(words)}

# Initialize some parameters.
sentence_len = 50;   xdata = [];   ydata = []
```

```
# Break up the corpus into sentences.
for i in range(len(corpus) - sentence_len):
  sentence = corpus[i: i + sentence_len]
  next_word = corpus[i + sentence_len]
  xdata.append([encoding[w] for w in sentence])
  ydata.append(encoding[next_word])

# The one-hot-encoded variables.
n_sentences = len(xdata)
x = np.zeros((n_sentences, sentence_len, \
  n_words), dtype = bool)
y = np.zeros((n_sentences, n_words), dtype = bool)

# Populate the variables.
for i, sentence in enumerate(xdata):
  for t, encoded_word in enumerate(sentence):
    x[i, t, encoded_word] = 1
  y[i, ydata[i]] = 1
```

LSTM example, learning code, continued

```
# Learn_Recipes.py, continued

# Save the metadata.
g = shelve.open("data/recipes.shelve")
g["sentence_len"] = sentence_len
g["n_words"] = n_words
g["encoding"] = encoding
g["decoding"] = decoding
g.close()

# Create the NN.
model = km.Sequential()

# A layer of LSTMs.
model.add(kl.LSTM(256,
  input_shape = (sentence_len, n_words)))
```

```
# Add a fully-connected output layer.
model.add(kl.Dense(n_words, activation = 'softmax'))

# The usual compilation.
model.compile(loss = 'categorical_crossentropy',
  optimizer = 'sgd', metrics = ['accuracy'])

# Run the fit.
fit = model.fit(x, y, epochs = 200,
  batch_size = 128, verbose = 2)

# Save the model.
model.save('data/recipes.model.h5')
```

LSTM example, running

Do not run this. And don't even think of running it without a GPU.

```
ejspence@mycomp ~>
ejspence@mycomp ~> python Learn_Recipes.py
Using Tensorflow backend.
Epoch 1/200
- 540s - loss:  6.8757 - acc:  0.1458
Epoch 2/200
- 541s - loss:  5.5554 - acc:  0.1462
⋮
Epoch 198/200
- 540s - loss:  0.2978 - acc:  0.9198
Epoch 199/200
- 540s - loss:  0.2932 - acc:  0.9210
Epoch 200/200
- 541s - loss:  0.2873 - acc:  0.9223
ejspence@mycomp ~>
```

LSTM example, generating code

```
# Generate_Recipe.py
import shelve, numpy as np
import keras.models as km
import random

# Read the parameters.
g = shelve.open("data/recipes.shelve")
sentence_len = g["sentence_len"]
n_words = g["n_words"]
encoding = g["encoding"]
decoding = g["decoding"];   g.close()

# Create a random seed sentence.
seed = []
for i in range(sentence_len):
  seed.append(decoding[random.randint(0,
    n_words - 1)])
```

```
# Get the model.
model = km.load_model(data/recipes.model.h5')

# Create and populate the x data.
x = np.zeros((1, sentence_len, n_words), dtype = bool)
for i, w in enumerate(seed): x[0, i, encoding[w]] = 1

text = ""

for i in range(1000):
  pred = np.argmax(model.predict(x, verbose = 0))
  text += decoding[pred] + " "
  next_word = np.zeros((1, 1, n_words), dtype = np.bool)
  next_word[0, 0, pred] = 1
  x = np.concatenate((x[:, 1:, :], next_word), axis = 1)

print("Our recipe:")
print(text)
```

LSTM example, prediction

```
ejspence@mycomp ~> python Generate_Recipe.py

sour cream and horseradish whip squares

amount measure ingredient -- preparation method
-------- ------------ --------------------------------
1 3/4 pounds top flour -- frozen
1/2 cup olive oil
1/4 cup lemon juice
1 teaspoon garlic -- finely minced
1 teaspoon lemon rind -- finely grated
1 teaspoon vanilla extract

prepare the baking dish in a bowl , make crust , with the topping .  set aside .  add all
dry ingredients , blend well with an electric mixer .  beat the egg whites with a mixer until
blended and bake at 350f , for 15 minutes .  remove from firm , and carefully pour over margarine
.  bake until tester is well blended , 8 to 10 minutes with small spatula , ; sprinkle with
confectioner's sugar .  sprinkle chopped pecans over peaches .  spread immediately .
```

LSTM example, notes

Some notes about this example.

- The model only gets 93% training accuracy so far, hence some of the 'features' of the generated recipe.
- Getting it this far took over 16 hours of training on a GPU.
- Note the things that it gets correct:
  - It creates a title.
  - It correctly lays out the amount/measure/ingredients table.
  - It lays out ingredients with sensible amounts.
  - The instructions are more-or-less sensible.
- The things it gets wrong:
  - The instructions reference ingredients which are not in the ingredients list.

Further training should improve this. A larger dataset would improve it even more.

SciNet compute • calcul CANADA

LSTM example, notes continued

Some more notes about this example.

- Examine the code if you want to see how the data was preprocessed.
- In particular, the many blank spaces, which created the nice formatting of the recipes, were removed, since they were dominating the fit.
- You need a large amount of data to train this model (there are a large number of parameters). Consequently it takes a long time. Use a GPU!
- The field of Natural Language Processing is heavily based on techniques similar to this. However, much more efficient techniques are used.
- In particular, if you need to go down this road, you should explore the 'word embedding' packages word2vec (`https://code.google.com/p/word2vec`) and GloVe (`http://nlp.stanford.edu/projects/glove`). These create low-dimensional vectors with semantic meaning; similar words have similar vectors.

SciNet compute • calcul CANADA

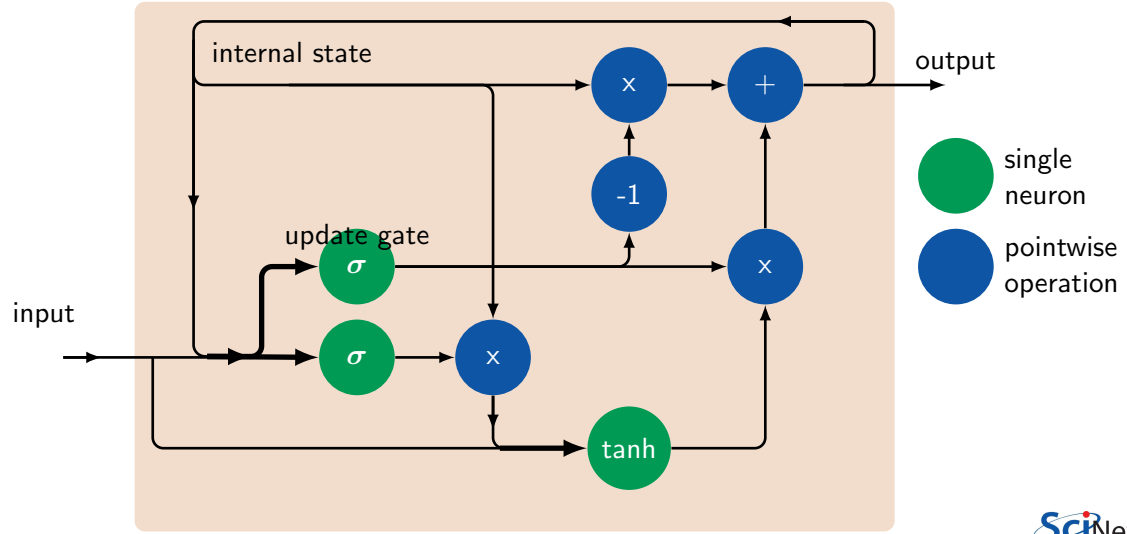
Other types of RNNs

There are many types of RNNs out there.

- Gated Recurrent Units (GRUs). These are similar to LSTMs, and are the leading competitor to them. We'll look at these in more detail.
- Bidirectional RNNs. Sometimes your output depends not just on data points that came previously, but also on future data points. These networks are just two RNNs glued together, reading the input from opposite directions.
- Fully-recurrent neural networks. In these, every node is connected to every node in the network, including itself.
- Variations on LSTMs. There are many: LSTMs with 'peep holes', LSTMs with combined input and forget gates, and many others.

SciNet compute • calcul CANADA

Gated Recurrent Units



Gated Recurrent Units, notes

Some notes about gated recurrent units (GRUs).

- Are GRUs better than LSTMs? In many situations they give similar results.
- But because there are fewer trainable neurons in a GRU it will train faster than an LSTM.
- Side note: only LSTMs, GRUs and SimpleRNNs are implemented in Keras. SimpleRNNs are essentially the nodes described at the beginning of class. They are supplied for educational purposes only; do not use them.

SciNet compute • calcul CANADA

Other neural network results

Neural networks have been applied to all manner of situations. There are too many other areas to cover in detail. We will finish the class by reviewing some areas which have been tackled, and the current state of the art.

- Image classification.
- Object detection.
- Image segmentation.
- Generative networks.
- Style transfer.

There are many many other applications which have been developed.

SciNet compute • calcul CANADA

Other neural network results: image classification

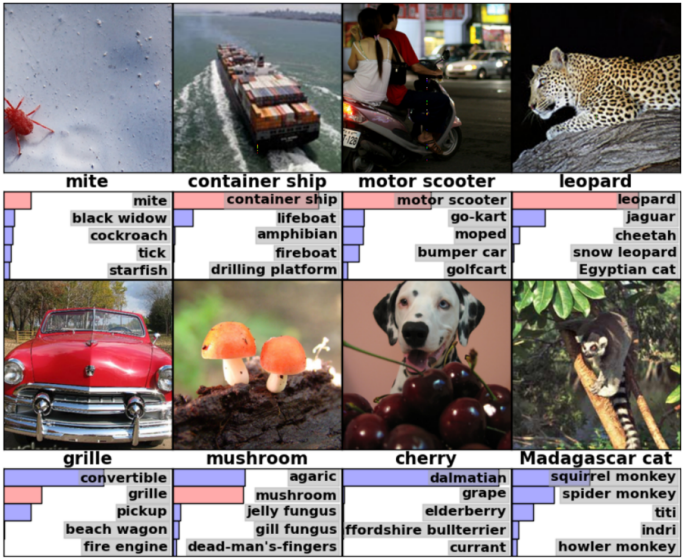
Neural networks have revolutionized image classification.

- This is the problem of: given a photo, what is in it?
- This is similar to working with MNIST data, but there are important differences:
  - the images are much bigger,
  - the images are colour,
  - there are thousands of categories.
- The networks which solve these problems are deep, and often contain a number of architectural innovations.

Early in the deep-learning revolution this was a very active area of research, strongly driven by the ILSVRC competition.

SciNet compute • calcul CANADA

Image classification, top-5 example



ILSVRC

From 2010 - 2017 the most pre-eminent image classification competition was ILSVRC.

- Stands for ImageNet Large Scale Visual Recognition Challenge (the "ImageNet Competition").
- Millions of training images, 1000 categories.
- More-recent competitions have also had object-detection challenges, which involves also locating objects within images, and now also within videos.
- The classification competitions allowed the top-5 classifications to be submitted for any given image, along with the associated bounding boxes for each object.
- In 2017, 29 of 38 teams had greater than 95% accuracy. This is probably why the challenge does not appear to be running anymore.

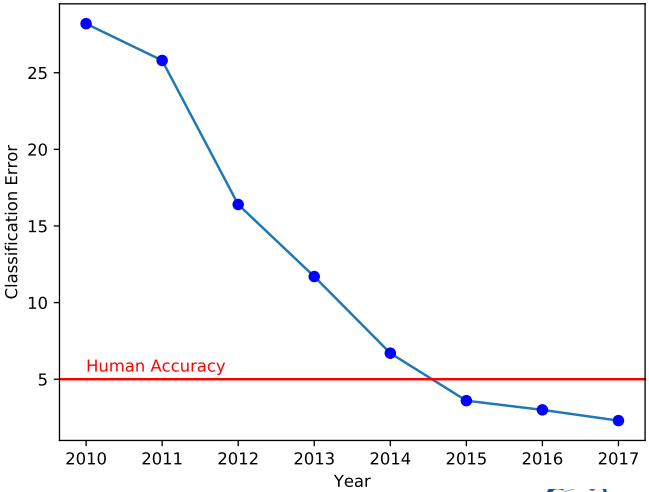
This competition saw significant innovations in neural network architectures.

SciNet compute • calcul CANADA

ILSVRC, continued

ILSVRC winners introduced new ideas to image classification.

- AlexNet (2012): first network to use successive convolutional layers in the competition.
- GoogLeNet (2014): introduced the "Inception Module".
- ResNet (2015): introduced the "ResNet Module".



Other neural network results: object detection

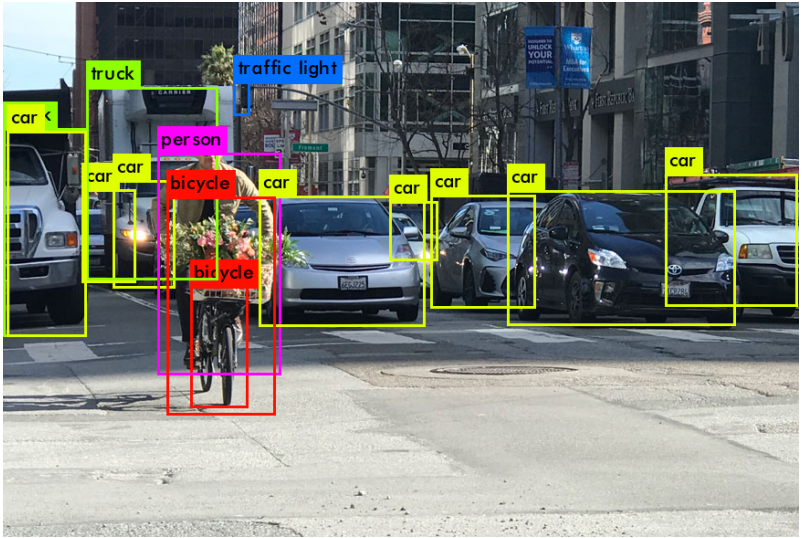
The use of neural networks in object detection is an active area of research.

- This is the problem of: given a photo, what is in it, and WHERE is it?
- This is more complicated than simple image classification, obviously:
  - the images usually have multiple objects within them,
  - a bounding box must be put around each object,
  - each object must simultaneously be classified,
  - sometimes the whole object is not visible in the image.
- The networks which solve these problems are deep, and often contain a number of architectural innovations.
- The networks are also optimized for speed, since real-time object detection is necessary for self-driving cars.

This is not a solved problem on the most-difficult data sets.

SciNet compute • calcul CANADA

Object detection, example



Other neural network results: image segmentation

Obviously, not all applications of object detection have their needs sufficiently met with bounding boxes. This leads to one of the hardest image-analysis problems: image segmentation.

- This is the problem of: given a photo, what is in it, and draw an outline around the boundaries of the objects.
- This is more complicated than simple object detection, obviously:
  - the images usually have multiple objects within them,
  - the object's 'mask' must outline each object,
  - each object must simultaneously be classified,
  - distinct objects, even of the same type, must be uniquely identified.
- The use of such networks in medicine, especially radiology, is a very active area of research.

These networks are moving toward the localizing of tumours, cancer, etc. in diagnostic images.

SciNet compute • calcul CANADA

Image segmentation, example

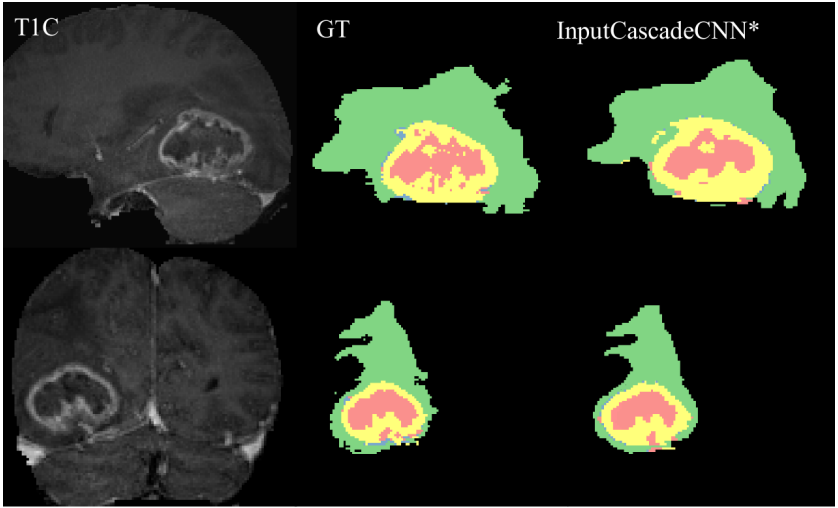


Image stolen from `https://arxiv.org/abs/1505.03540`

Discriminative versus generative networks

Let's examine a distinction we haven't yet made: discriminative versus generative networks.

- A discriminative network is trained to detect whether some input data is a member of a given class. Examples include the standard networks we've come to know and love, such as fully-connected and CNNs.
- In probabilistic terms, given the input data $\mathbf{x}$, and a desired label $\boldsymbol{y}$, the discriminative network calculates the conditional probability $\boldsymbol{P}(\boldsymbol{y}|\mathbf{x})$.
- In contrast, a generative network is trained to calculate the probability of the data directly, $\boldsymbol{P}(\mathbf{x})$.
- Once we have $\boldsymbol{P}(\mathbf{x})$, we can sample from this distribution to create new data.

The ability to generate fake, authentic looking data has a number of applications.

Generative networks

There are several types of generative networks you may run into.

- PixelCNN: an auto-regressive model, the conditional distribution of each pixel is modeled given the left and above pixels.
- Variational Autoencoders (2014): one network (the encoder) casts the input data into a lower-dimensional representation; a second network (the decoder) reconstructs the input from the low-D representation.
- Generative Adversarial networks (2014): two networks are trained simultaneously, one to generate fake data, and one to identify the fake data, when compared to real data.
- Boltzmann Machines, Fully Visible Belief Networks, Generative Stochastic Networks, and others.

Today we won't go into these in detail, but you need to know they exist.

SciNet compute • calcul CANADA

GANs can do amazing things

https://thispersondoesnotexist.com

Style transfer (2015)

As a change of pace, we'll finish today with an interesting application of neural networks.

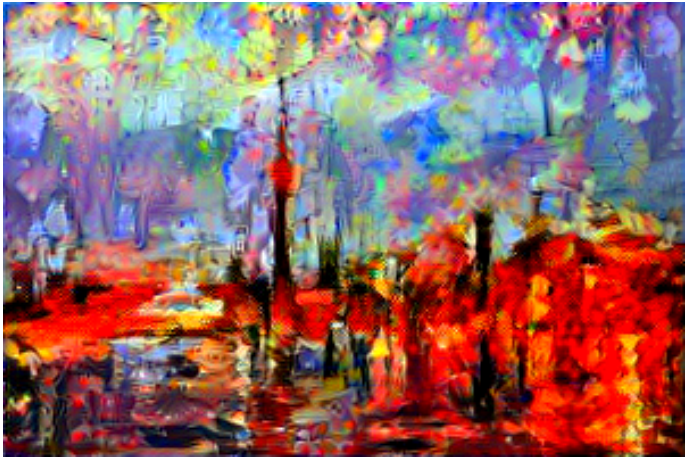
This application leverages Convolutional Neural Network's strengths in visual pattern recognition and object localization.

Interestingly, CNNs can also distinguish between "content" of an image, and "style". This can be used to impose the style of one image onto another. This is known as "style transfer".

SciNet compute • calcul CANADA

Style transfer, an example

Style transfer, an example, continued



Style transfer, continued

So how does it work? The technique proceeds as follows.

- Start with two images, the 'content' image (the photo), and the 'style' image (the painting).
- We create a loss function, which measures how close the 'content' of the generated image is to the 'content' image.
- We create a second loss function, which measures how close the 'style' of the generated image is to the 'style' image.
- We combine these two loss functions into a single loss function.
- We use scipy, rather than Keras, to minimize the combined loss function, in the process creating our generated image.
- Note that the minimization is done, not by adjusting the weights and biases of a neural network, but rather by choosing the input to the network which minimizes the loss function.

SciNet compute • calcul CANADA

Image transformation

This is part of a broader area of research known as "image transformation". In this field, CNNs are used to perform a number of different applications:

- super-resolution,
- colourization,
- surface-normal prediction,
- depth prediction,
- style transfer.

This is an active area of research.

SciNet compute • calcul CANADA

Style transfer + GANs

Linky goodness: CNNs

Convolutional neural networks:

- http://scs.ryerson.ca/~aharley/vis/conv/flat.html
- http://www.cs.utoronto.ca/~fidler/teaching/2015/CSC2523.html
- https://cs231n.github.io/convolutional-networks
- http://deeplearning.net/tutorial/lenet.html
- https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb

SciNet compute • calcul CANADA

Linky goodness: RNNs

RNNs and LSTMs:

- https://karpathy.github.io/2015/05/21/rnn-effectiveness
- http://colah.github.io/posts/2015-08-Understanding-LSTMs
- https://deeplearning4j.org/lstm.html
- http://www.deeplearningbook.org/contents/rnn.html
- http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns

SciNet compute • calcul CANADA

Linky goodness: image classification

ILSVRC:

- `http://www.image-net.org/challenges/LSVRC`

Image classification networks:

- AlexNet: `http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf`
- ResNet: `https://arxiv.org/abs/1512.03385`
- GoogLeNet: `https://arxiv.org/abs/1409.4842`
- Xception: `https://arxiv.org/abs/1610.02357`

SciNet compute • calcul CANADA

Linky goodness: GANs

GANs:

- https://arxiv.org/abs/1701.00160
- https://blog.openai.com/generative-models
- https://deephunt.in/the-gan-zoo-79597dc8c347
- http://arxiv.org/abs/1511.06434
- https://medium.com/towards-data-science/gan-by-example-using-keras-on-tensorflow-backend-1a6d515a60d0
- https://arxiv.org/abs/1606.03498

WGAN:

- https://arxiv.org/abs/1701.07875 (original WGAN paper)
- https://arxiv.org/abs/1704.00028
- http://www.alexirpan.com/2017/02/22/wasserstein-gan.html

SciNet compute • calcul CANADA

Linky goodness: style transfer

Style transfer:

- https://arxiv.org/abs/1508.06576 (the original paper)
- https://medium.com/mlreview/making-ai-art-with-style-transfer-using-keras-8bb5fa44b216
- https://github.com/titu1994/Neural-Style-Transfer
- https://chrisrodley.com/2017/06/19/dinosaur-flowers

Photo style transfer:

- https://arxiv.org/abs/1703.07511

SciNet compute • calcul CANADA