Deep Neural Network (DNN) for Images and Text

IADS Summer-School 2021

Dr Haider Raza
Lecturer in AI for Decision-Making
University of Essex July 26-27, 2021

About

- We will try to get a practical understanding of neural networks
  - The topic is massive
- The field has changed names a number of times
  - Connectionist systems
  - Neural networks
  - Deep learning
- They are all the same stuff, different name
  - Re-branding

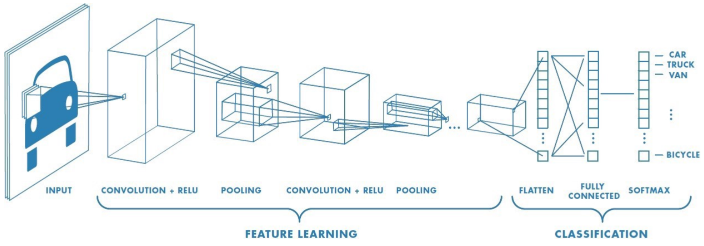
What inspired Convolutional Networks

CNNs are biologically-inspired models inspired by research by D. H. Hubel and T. N. Wiesel. They proposed an explanation for the way in which mammals visually perceive the world around them using a layered architecture of neurons in the brain.

It inspired “Yann LeCun” and his team to attempt to develop similar pattern recognition mechanisms in computer vision

The architecture of Convolutional Neural Networks (CNNs)_

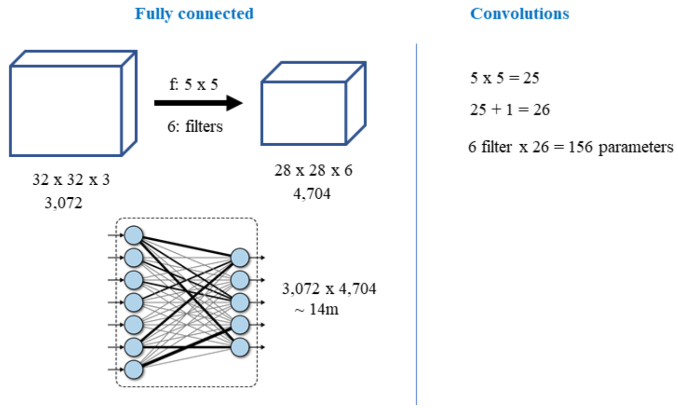
1. Local connections (i.e. cluster of neurons) [1]
2. Layering (i.e. hierarchy of features learned)
3. Spatial invariance (i.e. if you see car it's a car. View don't matters). Invariant of size, contrast, rotation, etc.



[1] https://medium.com/@RaghavPrabhu/

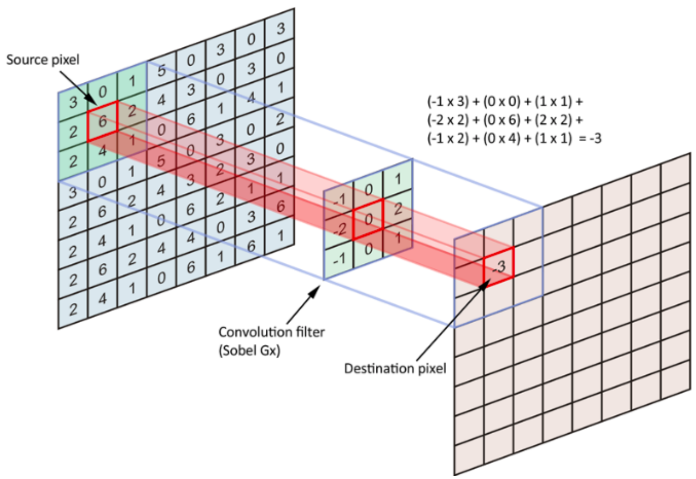
Why convolution neural networks?

1. Parameter sharing
2. Sparsity of connections



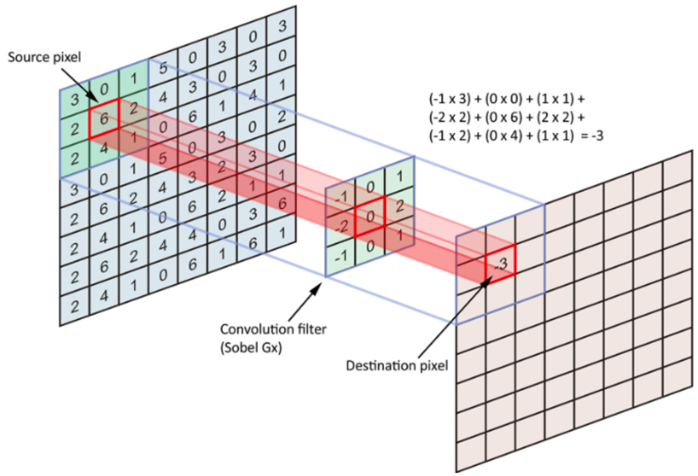
Parameter sharing

A feature detector (such as a vertical edge detector) that's useful in one park of the image is possibility useful in another part of image



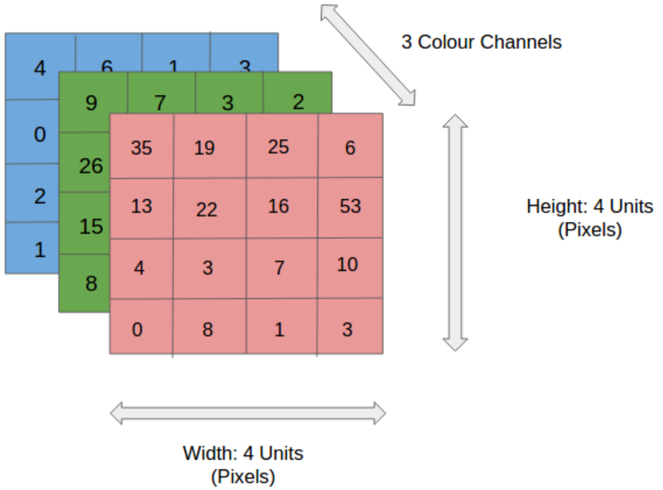
Sparsity of connections

In each layer, each output value depends only on a small number of inputs



Translation invariance property is possible because of this

Step 1: Prepare a dataset of images



Step 1: Prepare a dataset of images

1. Every image is a matrix of pixel values
2. The range of values that can be encoded in each pixel depends upon its bit size
3. Most commonly, we have 8 bit or 1 Byte-sized pixels. Thus the possible range of values a single pixel can represent is [0, 255].
4. However, with coloured images, particularly RGB (Red, Green, Blue)-based images, the presence of separate colour channels (3 in the case of RGB images) introduces an additional 'depth' field to the data, making the input 3-dimensional.
5. Hence, for a given RGB image of size, say 255×255 (Width x Height) pixels, we'll have 3 matrices associated with each image, one for each of the colour channels.
6. Thus the image in it's entirety, constitutes a 3-dimensional structure called the Input Volume (255x255x3).

Popular datasets for images

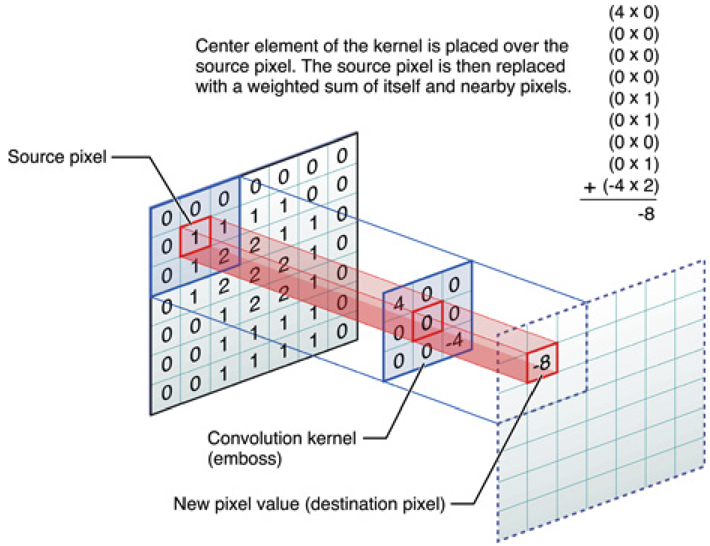
- MNIST: handwritten digits (http://yann.lecun.com/exdb/mnist/)
- NIST: similar to MNIST, but larger
- Perturbed NIST: a dataset developed in Yoshua's class (NIST with tons of deformations)
- CIFAR10 / CIFAR100: 32×32 natural image dataset with 10/100 categories (http://www.cs.utoronto.ca/~kriz/cifar.html)
- Caltech 101: pictures of objects belonging to 101 categories (http://www.vision.caltech.edu/Image_Datasets/Caltech101/)
- Caltech 256: pictures of objects belonging to 256 categories (http://www.vision.caltech.edu/Image_Datasets/Caltech256/)
- Imagenet: image database organized according to the WordNethierarchy (http://www.image-net.org/)

Step 2: Convolution

Convolution is a mathematical operation on two functions (f and g) to produce a third function that expresses how the shape of one is modified by the other (Source: Wiki)

$$\underbrace{\begin{array}{|c|c|c|c|c|c|c|}\hline 0&1&1&1&0&0&0\\\hline 0&0&1&1&1&0&0\\\hline 0&0&0&1&1&1&0\\\hline 0&0&0&1&1&0&0\\\hline 0&0&1&1&0&0&0\\\hline 0&1&1&0&0&0&0\\\hline 1&1&0&0&0&0&0\\\hline\end{array}}_{\mathbf{I}} * \underbrace{\begin{array}{|c|c|c|}\hline 1&0&1\\\hline 0&1&0\\\hline 1&0&1\\\hline\end{array}}_{\mathbf{K}} = \underbrace{\begin{array}{|c|c|c|c|c|}\hline 1&4&3&4&1\\\hline 1&2&4&3&3\\\hline 1&2&3&4&1\\\hline 1&3&3&1&1\\\hline 3&3&1&1&0\\\hline\end{array}}_{\mathbf{I}*\mathbf{K}}$$

Step 2: Convolution. . .



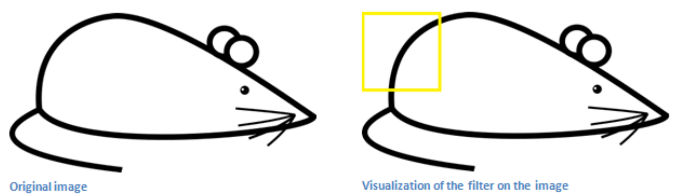
Step 2: Convolution. . .

- A convolution is an orderly procedure where two sources of information are intertwined
- A kernel or filter is a smaller-sized matrix in comparison to the input dimensions of the image, that consists of real valued entries
- Kernels are then convolved with the input volume to obtain so-called 'activation maps' (also called feature maps) [2]

[2] http://cs231n.github.io/convolutional-networks/

Step 2: Convolution. . .

Filter at the top left corner of image [3]



[3] https://adeshpande3.github.io

Step 2: Convolution. . .

Creating model using keras

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32,(3,3), activation='relu', input_shape=(28,28,1)))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(64,(3,3), activation='relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(64,(3,3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

Step 2: Convolution...

Multiply the values in the filter with the original pixel values of the image [4]

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

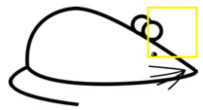
Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

[4] https://adeshpande3.github.io

Step 2: Convolution...

What happens when we move our filter? [5]



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

[5] https://adeshpande3.github.io

Step 3: Pooling

What happens when we move our filter? [6]

4	6	1	3
0	8	12	9
2	3	16	100
1	46	74	27

⇒

8	12
46	100

(i)

35	19	25	6
13	22	16	63
4	3	7	10
9	8	1	3

⇒

35	63
9	10

(iii)

9	7	3	2
26	37	14	1
15	29	16	0
8	6	54	2

⇒

37	14
29	54

(ii)

35	19	25	6
13	22	16	63
4	3	7	10
9	8	1	3

⇒

35	25	63
22	22	63
9	8	10

(iv)

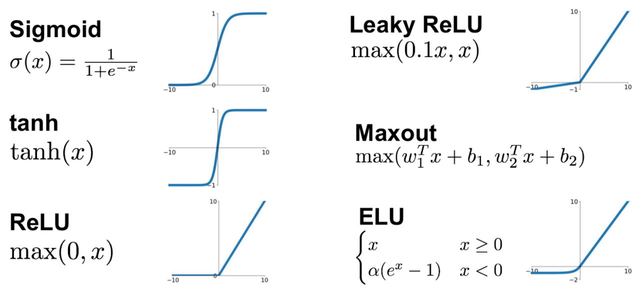
[6] https://en.wikipedia.org/wiki/Convolution

Step 3: Pooling....

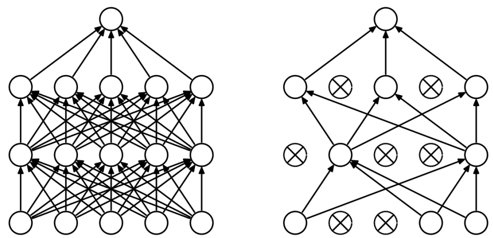
1. Pooling reducing the spatial dimensions (Width x Height) of the Input for next layer
2. The transformation is either performed by taking the
   - Maximum value from the values observable in the window (called 'max pooling')
   - Average of the values
   - Max pooling has been favoured over others due to its better performance characteristics
3. Pooling also called down-sampling

Step 4: Activation function (ReLU)

It is a non linear transformation that we do over the input signal. This transformed output is then send to the next layer of neurons as input



Step 5: Regularization (Dropout)



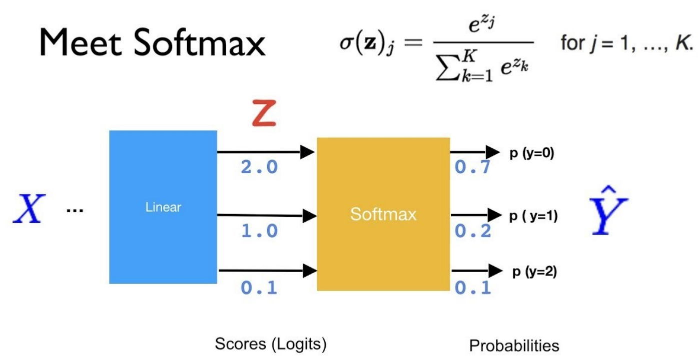
(a) Standard Neural Net (b) After applying dropout.

Step 5: Regularization (Dropout)

1. Dropout forces an artificial neural network to learn multiple independent representations of the same data by alternately randomly disabling neurons in the learning phase
2. Dropout is a vital feature in almost every state-of-the-art neural network implementation
3. To perform dropout on a layer, you randomly set some of the layer's values to 0 during forward propagation

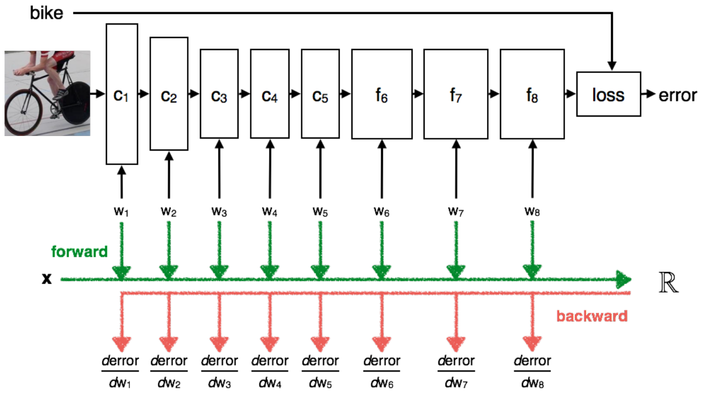
Step 6: Probability Conversion. . . .

At the end of the network, apply a softmax function to convert the outputs to probability values for each class and Choose most likely label (max probability value)



We went forward. How we are going to learn?

We can learn features and weight values through backpropagation



Step 6: Get Error and Update Weights

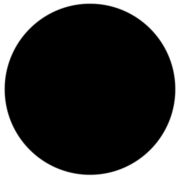
- Loss function
  - **For Classification**
    - Softmax Function $y_j = \frac{e^{\mathbf{x}^T \mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^T \mathbf{w}_k}}$ with negative log likelihood $-\sum_{i=1}^{K} t_i \log y_i$
  - **For Regression**
    - Mean squared error $\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(t_i - y_i)^2$
- Weight Update

$$\omega_i \leftarrow \omega_i - \eta \frac{\partial E}{\partial w_i} + \alpha \omega_i - \lambda \eta \omega_i$$

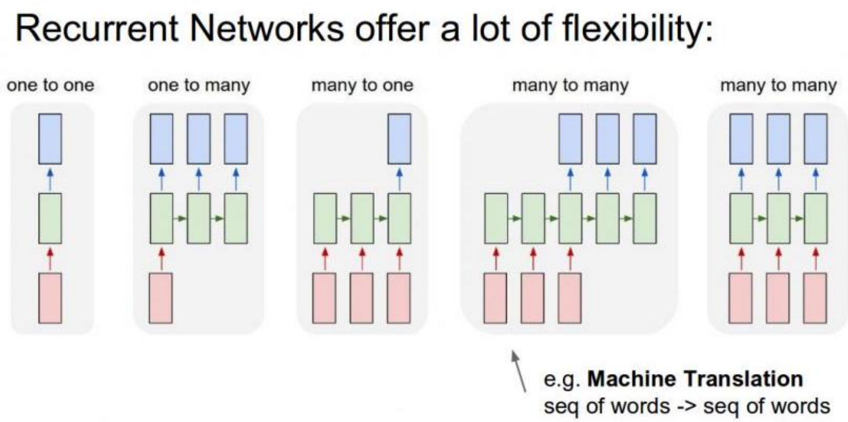
where η - learning rate,
α - momentum,
λ - weight decay

Deep Learning for Text

Do you know where this circle will move next?



Deep Sequence Modelling



what is sequence processing

- One-to-One: Binary Classification: Will I pass this class?: Yes/No
- Many-to-One: Sentiment Classification: Tweet Classification
- One-to-Many: Image Captioning: Show single-image to get the caption
- Many-to-Many: Machine Translation: English to any language such as French, Hindi, Urdu, German.

Why RNN?

Unlike other algorithms, NN can also encode useful and obvious relationship in the data domain

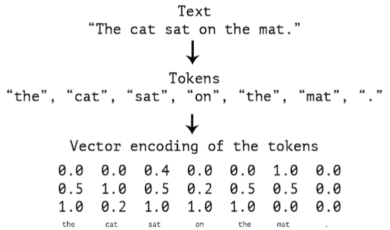
1. Local spatial dependencies (computer vision: CNN)
2. Time dependencies (language and speech: RNN)

Note: Keep in mind throughout that none of deep-learning models truly understand text in a human sense; rather, these models can map the statistical structure of written language, which is sufficient to solve many simple textual tasks

Vectorizing & Tokenization

Deep-learning models don’t take as input raw text: they only work with numeric tensors. Vectorizing text is the process of transforming text into numeric tensors. This can be done in multiple ways

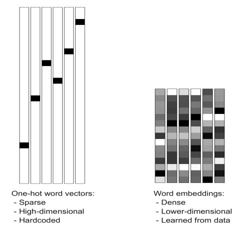
1. Segment text into Words and transform to vector
2. Segment text into Characters and transform character into vector
3. Extract n-grams of words or characters, and transform each n-gram into a vector.



N-Grams example

```
{"The", "The cat", "cat", "cat sat", "The cat sat",
  "sat", "sat on", "on", "cat sat on", "on the", "the",
  "sat on the", "the mat", "mat", "on the mat"}
```

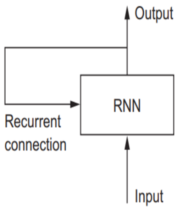

Word vector vs Word Embeddings



- the vectors obtained through one-hot encoding are binary, sparse (mostly made of zeros), and very high-dimensional (same dimensionality as the number of words in the vocabulary)
- word embeddings are low dimensional floating-point vectors (that is, dense vectors, as opposed to sparse vectors)
- So, word embeddings pack more information into far fewer dimensions

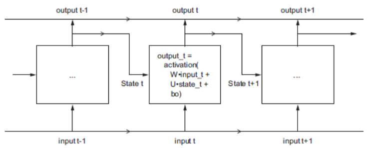
Recurrent Neural Network

It processes sequences by iterating through the sequence elements and maintaining a state containing information relative to what it has seen so far. RNN is a type of neural network that has an internal loop.

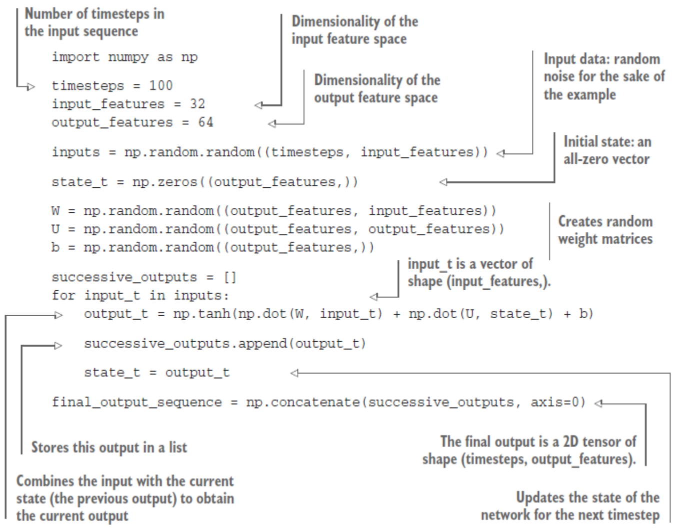


RNN using NumPy

1. RNN takes as input a sequence of vectors, which you'll encode as a 2D tensor of size `(timesteps, input_features)`
2. It loops over timesteps, and at each timestep, it considers its current state at `t` and the input at `t` (of shape `(input_features,)`, and combines them to obtain the output at `t`
3. You'll then set the state for the next step to be this previous output. For the first timestep, the previous output isn't defined; hence, there is no current state. So, you'll initialize the state as an all-zero vector called the initial state of the network



RNN implementation NumPy



Recurrent layer in Keras

1. The process you just naively implemented in Numpy corresponds to an actual Keras layer—the `SimpleRNN` layer:

```
from keras.layers import SimpleRNN
```

2. The only difference is in Keras layers, not a single sequence as in the Numpy example. This means it takes inputs of shape `(batch_size, timesteps, input_features)`, rather than `(timesteps, input_features)`

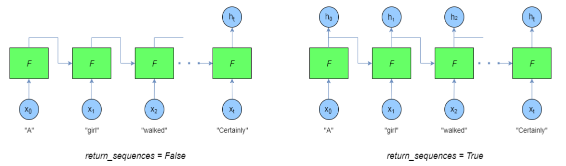
Simple RNN

SimpleRNN can be run in two different modes

1. It can return either the full sequences of successive outputs for each `timestep` (a 3D tensor of shape `(batch_size, timesteps, output_features)`)
2. the last output for each input sequence (a 2D tensor of shape `(batch_size, output_features)`)

These two modes are controlled by the `return_sequences` constructor argument

Return Sequence in RNN



1. False: Return sequences refer to return the hidden state and by `default: False` . The last hidden state output captures an abstract representation of the input sequence. In some case, it is all we need, such as a classification or regression model where the RNN is followed by the Dense layer(s) to generate logits or softmax probabilities
2. True: two primary situations

- RNN layer or layers can have the full sequence as input
- Such as speech recognition or OCR sequence modelling with CTC

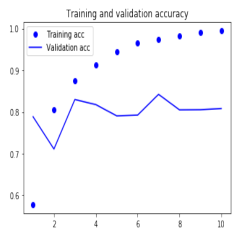
RNN for IMDB review dataset

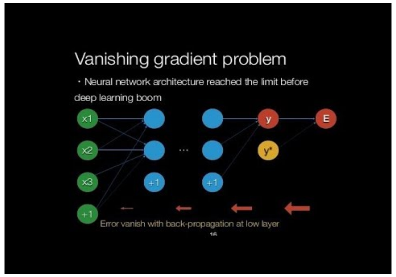
```
from keras.layers import Dense
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(input_train, y_train,
epochs=10,
batch_size=128,
validation_split=0.2)
```

Validation accuracy 82.26%, reason: Inputs only consider the first 500 words, rather than full sequences—hence, the RNN has access to less information

Understanding the LSTM and GRU layers

- SimpleRNN has a major issue: `vanishing gradient problem`: Model able to retain at time `t` information about inputs seen many timesteps before, in practice, such long-term dependencies are impossible to learn



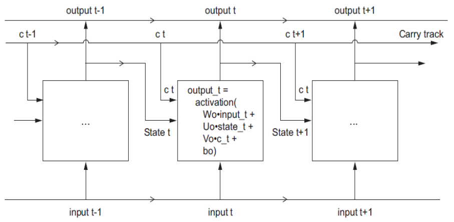


- LSTM and GRU layers are designed to solve this problem

LSTM (Long Short-Term Memory)

- This layer is a variant of the SimpleRNN layer you already know about
- It adds a way to carry information across many timesteps (such as a conveyor belt running parallel to sequence)
- The information can jump onto the belt at any point, transported to later timestep, and jump off, intact when you need it.

Summary: it saves information for later, this preventing older signals from `gradually vanishing` during processing



LSTM for IMDB review dataset

```
from keras.layers import LSTM

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=['acc'])
history = model.fit(input_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

Achieve up to 89% validation accuracy. Not bad: certainly much better than the SimpleRNN network

How to improve LSTM

1. No effort to tune hyperparameters such as the embeddings dimensionality or the LSTM output dimensionality
2. Another may be lack of regularization
3. The primary reason is that analyzing the global, long-term structure of the reviews (LSTM is not helpful for sentiment analysis problem). It is good for `question-answering` and `machine translation`

Conclusion

- We have touched upon neural networks
- It's a really hot topic right now
- Requires a bit of dedication
- New algorithms are coming out every day