Deep learning

Leonardo Hügens, Pedro Guedes

Faculdade de Ciências, Universidade do Porto

October 4, 2023

Paper

This presentation is based on the paper `Deep learning` by `LeCun, Bengio, Hinton`, a review article from the journal `Nature`.

REVIEW

doi:10.1038/nature14539

Deep learning

Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.

Outline

1 Introduction

2 Neural Networks

3 Unsupervised learning

4 CNN's

5 Implementation

6 RNN's

7 Future

Introduction

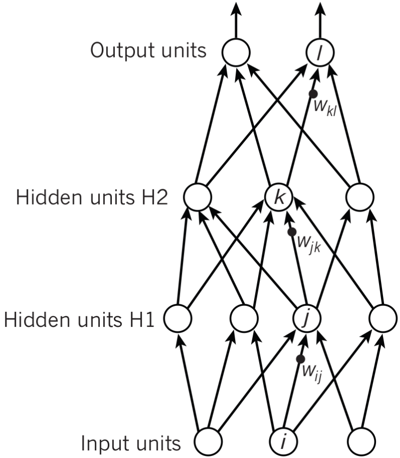
- Deep learning allows computation models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction.

Introduction

- Deep learning allows computation models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction.
- Suitable for:

    Speech recognition
    Visual object recognition
    Object detection
    Etc.

Neural Networks



$$y_l = f(z_l)$$

$$z_l = \sum_{k \,\varepsilon\, H2} w_{kl}\, y_k$$

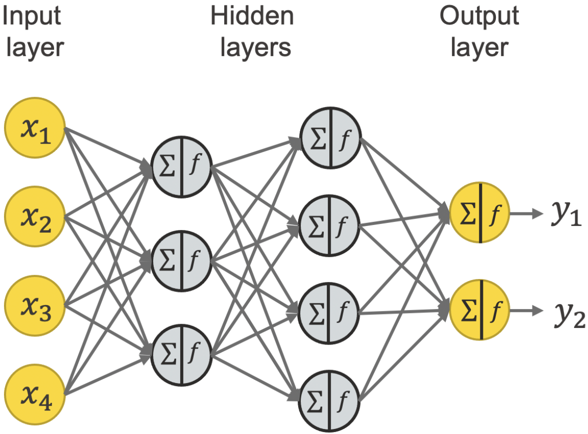
$$y_k = f(z_k)$$

$$z_k = \sum_{j \,\varepsilon\, H1} w_{jk}\, y_j$$

$$y_j = f(z_j)$$

$$z_j = \sum_{i \,\varepsilon\, \text{Input}} w_{ij}\, x_i$$

Neural Networks



Examples of Non-linear activation functions (f's)

- **ReLU** - Rectified Linear Unit, $f(x) = \max(0, x)$
- **Sigmoids**, such as Hyperbolic tangent, $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- **Logistic function**, $f(x) = \frac{1}{1+e^{-x}}$

Examples of Loss functions (E's)

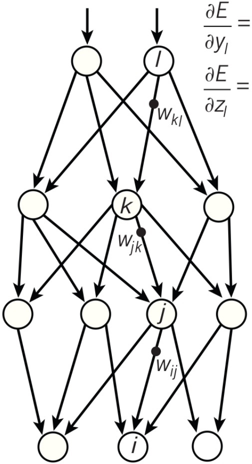
- **Cross-Entropy**

  Binary CE, -(ylog$(p) + (1 - y)\log(1 - p))$)

  Multiclass CE, -$\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$

- **Mean Squared Error**, $\sum_{i=1}^{D}(y_i - p_i)^2$

Backpropagation



$$\frac{\partial E}{\partial y_l} = y_l - t_l$$

$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l} \frac{\partial y_l}{\partial z_l}$$

$$\frac{\partial E}{\partial y_k} = \sum_{l \, \varepsilon \, \text{out}} w_{kl} \frac{\partial E}{\partial z_l}$$

$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_k}$$

$$\frac{\partial E}{\partial y_j} = \sum_{k \, \varepsilon \, \text{H2}} w_{jk} \frac{\partial E}{\partial z_k}$$

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j}$$

Backpropagation

SGD - Stochastic Gradient Descent

- Using a `batch` of few examples, computing the outputs and the error, the average gradient and adjusting the weights accordingly.

Backpropagation

SGD - Stochastic Gradient Descent

- Using a `batch` of few examples, computing the outputs and the error, the average gradient and adjusting the weights accordingly.

- Offers:

  Faster convergence
  Better generalization (gets stuck in local minima less often)
  Memory efficient

Unsupervised learning

- Can create layers of feature detectors without requiring labelled data.

- Tries to mimic the input data, finding patterns, structure, or relationships.

Unsupervised learning

- Can create layers of feature detectors without requiring labelled data.

- Tries to mimic the input data, finding patterns, structure, or relationships.

- `Clustering`: In clustering, the algorithm groups similar data points together into clusters or groups. One popular algorithm for clustering is K-Means, which assigns each data point to one of K clusters, with K being a predefined number.

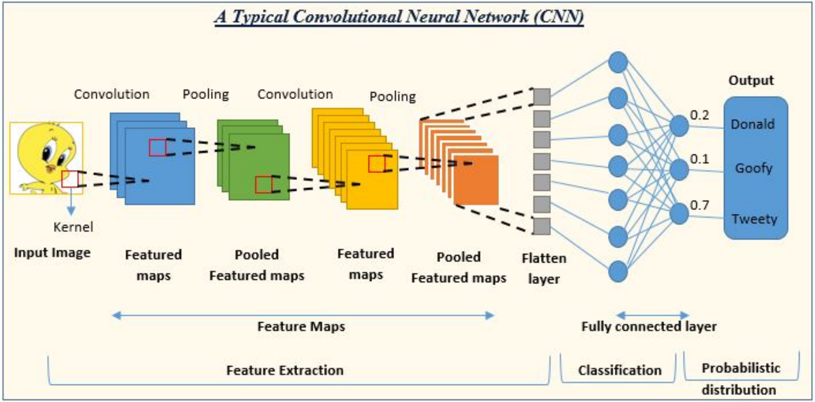
Convolutions

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

Pooling

$$\text{Max}\left[\begin{array}{cc|cc} 3 & 1 & 1 & 3 \\ 2 & 5 & 0 & 2 \\ \hline 1 & 4 & 2 & 1 \\ 4 & 7 & 2 & 4 \end{array}\right] = \begin{array}{|c|c|} \hline 5 & 3 \\ \hline 7 & 4 \\ \hline \end{array}$$

CNN's - Convolutional Neural Networks



Packages Versions

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import scipy

print("TensorFlow version: ", tf.__version__)
print("SciPy version: ", scipy.__version__)
```

```
TensorFlow version:  2.13.0
SciPy version:  1.11.2
```

Model definition

```python
# Define the CNN model
model = keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    layers.MaxPooling2D(2, 2),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(1, activation='sigmoid')  # Binary classification (cats vs. dogs)
])
```

Model definition

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2  (None, 74, 74, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 36, 36, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 17, 17, 128)       0
 g2D)

 flatten (Flatten)           (None, 36992)             0

 dense (Dense)               (None, 128)               4735104

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 4828481 (18.42 MB)
Trainable params: 4828481 (18.42 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Model training

```
# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=200,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=50
)
```

```
Epoch 1/10
200/200 [==============================] - 82s 408ms/step - loss: 0.6836 - accuracy: 0.5409
Epoch 2/10
200/200 [==============================] - 103s 513ms/step - loss: 0.6494 - accuracy: 0.6167
Epoch 3/10
200/200 [==============================] - 108s 538ms/step - loss: 0.6333 - accuracy: 0.6411
Epoch 4/10
200/200 [==============================] - 107s 535ms/step - loss: 0.6059 - accuracy: 0.6594
Epoch 5/10
200/200 [==============================] - 108s 541ms/step - loss: 0.5860 - accuracy: 0.6870
Epoch 6/10
200/200 [==============================] - 111s 554ms/step - loss: 0.5629 - accuracy: 0.7047
Epoch 7/10
200/200 [==============================] - 111s 552ms/step - loss: 0.5638 - accuracy: 0.7038
Epoch 8/10
200/200 [==============================] - 108s 539ms/step - loss: 0.5426 - accuracy: 0.7198
Epoch 9/10
200/200 [==============================] - 109s 543ms/step - loss: 0.5409 - accuracy: 0.7280
Epoch 10/10
200/200 [==============================] - 107s 535ms/step - loss: 0.5462 - accuracy: 0.7222
```

Model predictions - Joint Confusion Matrix

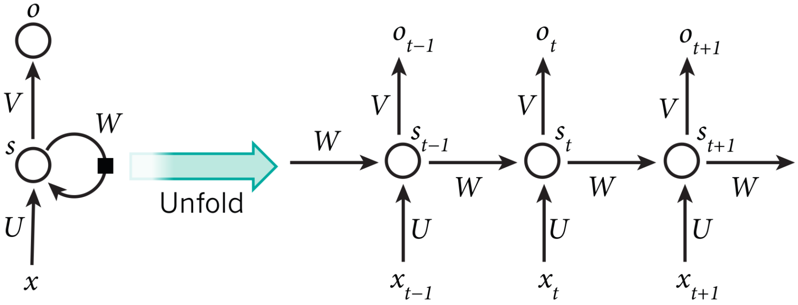
Dog, prob. 0.51

Cat, prob. 0.23

Dog, prob. 0.74

Cat, prob. 0.48

RNN's - Recurrent Neural Networks



RNN's - Recurrent Neural Networks

- Better for sequential data, i.e. language.

- Problem: long sequences cause backpropagated gradients to either explode of vanish.

RNN's - Recurrent Neural Networks

- Better for sequential data, i.e. language.

- Problem: long sequences cause backpropagated gradients to either explode of vanish.

- Possible solution: **LSTM** - Long short-term memory:

  A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

The future of deep learning

Expectations in 2015:

- Unsupervised learning to become far more important in the longer term, it is what human intelligence does.

The future of deep learning

Expectations in 2015:

- Unsupervised learning to become far more important in the longer term, it is what human intelligence does.
- Future progress in vision to come from combinations of CNN's and RNN's that use Reinforcement Learning to decide where to look.

The future of deep learning

Present reallity:

- A new architecture called the `Transformer`, introduced in the paper `Attention is All You Need, 2017`, that uses a mechanism called `Attention`, that brings benefits like:

  Capturing long-term dependencies
  Parallelization
  Scalability