IAML - Week 9

Antonio León Villares

December 2021

Contents

1 Artificial Neural Networks

- ANNs are a bad model of the human brain, but good (and powerful) as non-linear models
- Hidden units "learn" features of the input
- Backpropagation is used to solve the optimisation problem, and training ANNs
- CNNs and RNNs can be used to analyse images, or model sequences of data respectively

1.1 The Perceptron

- **What is a perceptron?** Short and sweet intro to the perceptron
  - a simple **linear** binary classifier:

$$\hat{y} = \begin{cases} 1, & \underline{w}^T \underline{x} + w_0 \geq 0 \\ -1, & otherwise \end{cases}$$

  - it uses a step function to determine the output of the classification
  - the step functions outputs +1 and -1 for convenience when training the perceptron
  - the idea is to create a linear model (like logistic regression) but with a simpler function than the sigmoid (we can think of the step function as the limit of the sigmoid as its slope goes to infinity)

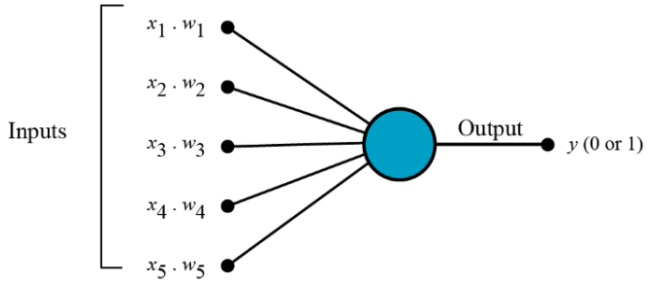


Figure 1: It is useful to think of the perceptron ike in this diagram

- **How can we train a perceptron?**
  - a logistic regression classifier could be trained using **gradient descent**, since computing the derivative was straightforward

- the step-function does not have a well behaved derivative
- the algorithm, due to Rosenblatt, iterates until the perceptron is able to correctly classify **all** data instances:

repeat
  for i in $1, 2, \ldots n$
    $\hat{y} \leftarrow \text{sign}[\mathbf{w}^T \mathbf{x}_i]$
    if $\hat{y} \neq y_i$
      $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$
until all training examples correctly classified

Figure 2: Note, the above algorithm does **not** include the bias; if it did we would also update the bias. Whilst convergence is not yet reached, we iterate over each data instance. For each instance, we predict the label. If the label is correct, we ignore. Otherwise, adjust the weights.

Algorithm 1: Perceptron Learning Algorithm

Input: Training examples $\{\mathbf{x}_i, y_i\}_{i=1}^m$.

Initialize $\mathbf{w}$ and b randomly.

while *not converged* **do**

  # # # Loop through the examples.

  for $j = 1, m$ **do**

    # # # Compare the true label and the prediction.

    $error = y_j$ - $\sigma(\mathbf{w}^T \mathbf{x}_j + b)$

    ### If the model wrongly predicts the class, we update the weights and bias.

    if *error != 0* **then**

      ### Update the weights.

      $\mathbf{w} = \mathbf{w} + error \times x_j$

      ### Update the bias.

      $b = b + error$

  Test for convergence

Output: Set of weights $\mathbf{w}$ and bias b for the perceptron.

Figure 3: Same algorithm, but updating the bias.

- **How does the algorithm work?**
  - General development of perceptrons

  - Intuition behind why the training converges
  - it might not seem immediately obvious how this simple algorithm can lead to a working linear classifier
  - if the prediction is correct, we don't touch the weights: that's easy
  - consider the case in which $\hat{y} = 1$ but $y_i = -1$. Then, the weights will be modified by **decreasing** them (since we set $\underline{w} \leftarrow \underline{w} - \underline{x}_i$)
  - that is, the algorithm is pushing to make the next output smaller, so as to predict -1 next time

- **How powerful is the perceptron as a classifier?**
  - if data is linearly separable, the perceptron will converge to a $\underline{w}$, such that it **always** separates the data
  - if data is **not** linearly separable, the algorithm won't converge (workarounds: *averaged perceptron*, *voted perceptron*
  - the issue is that Minsky showed that most real world (i.e interesting data) is **not** linearly separable

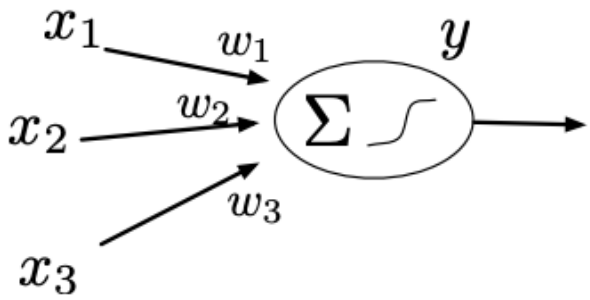
1.2 Defining Artificial Neural Networks

Up to now, all outr tasks have worked around working with linear classifiers/clusters (perhaps using kernels to handle non-linear data). Artificial Neural Networks allow us to build non-linear classifiers/regressors, based on many linear **units**.

	Supervised		Unsupervised	
	Class.	Regr.	Clust.	D. R.
Naive Bayes	✓			
Decision Trees	✓			
k-nearest neighbour	✓			
Linear Regression		✓		
Logistic Regression	✓			
SVMs	✓			
k-means			✓	
Gaussian mixtures			✓	
PCA				✓
Evaluation				
ANNs	✔	✔		

1.2.1 Classification: A Single Neuron

- **What is a unit in ANNs?**
  - a *unit* is the term used to describe an **artificial** or **simulated** neuron in neural networks
  - a *unit* is a **linear** "object", defined by a weight vector and a bias
  - if we combine many units (or neurons), we obtain a **neural network**, which will be **non-linear**
  - whilst units are used as simulated nuerons, there are many nuances and stark differences; we might use them itnerchangeably, but it is more correct to talk in terms of units
- **How does a unit work?**
  - a unit is very similar to a perceptron:



  - the **input** is a vector:

$$\underline{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

  - the unit is defined by a **weight vector**:

$$\underline{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}$$

    and a **bias** w_0

- when the input $\underline{x}$ is fed to the unit, the **activation** is computed:

$$a = \underline{w}^T \underline{x} + w_0$$

- the **output** y of the neuron is the result of applying a function to the **activation**:

$$y = g(a)$$

- **How can a unit be used to binary classification?**
  - if we set the function g to the sigmoid:

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

    the ouput of the unit can be thought as a probability, which we can threshold to give a classification (i.e assign class 1 to $\underline{x}$ if $y = g(a) \geq 0.5$)
  - notice, depending on how we train the unit (i.e what loss function we use), **a single unit is equivalent to a logistic regression classifier**
  - On the relationship between logsitic regression and the unit

1.2.2 Multilayer Networks

- **How can we construct an ANN, by combining units?**
  - the idea is to create a non-linear model, by by using outputs of units as inputs for other units:

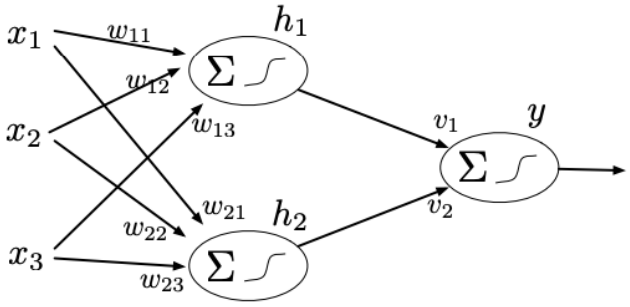


Figure 4: A multilayer ANN

  - the **input** is the same, the vector $\underline{x}$

- **What are hidden units?**
  - this time, there are 2 **hidden units**, h_1, h_2

- the hidden units transform the input, but are not used to give the output: they are "hidden"
- each hidden unit has its own set of weights ($\underline{w}_1, \underline{w}_2$ and biases (w_{10}, w_{20})

- **What is the output unit?**
  - the output of the hidden units is then fed to the **output unit**
  - the output unit has weights $\underline{v}$ and bias v_0, used to handle the ouput of the hideen layer
  - the network can be defined by the following "vector":

$$\langle \underline{w}_1, \underline{w}_2, \underline{v}, w_{10}, w_{20}, v_0 \rangle$$

1.2.3 Binary Classification

To perform classification, consider a function g like the sigmoid:

1. Pass the input through the hidden layer:

$$a_1 = g(\underline{w}_1^T \underline{x} + w_{10})$$

$$a_2 = g(\underline{w}_2^T \underline{x} + w_{20})$$

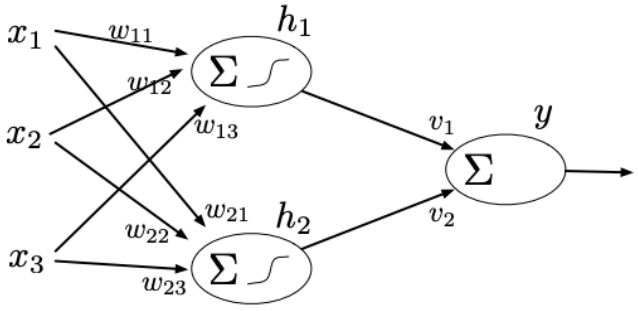
2. Pass the activations of the hidden layer to the output unit:

$$y = g\left(\underline{v}^T \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} + v_0\right)$$

3. Threshold y to output a classification (i.e $f(\underline{x}) = 1$ if $y > 0.5$)

1.2.4 Regression

Regression is similar, but we 2 different functions to non-linearise the output of the units: in the hidden layer, we can use the sigmoid, whilst in the output unit, we can use $g_3(a) = a$ (the linear function):



1. Pass the input through the hidden layer:

$$a_1 = g(\underline{w}_1^T \underline{x} + w_{10})$$

$$a_2 = g(\underline{w}_2^T \underline{x} + w_{20})$$

2. Pass the activations of the hidden layer to the output unit:

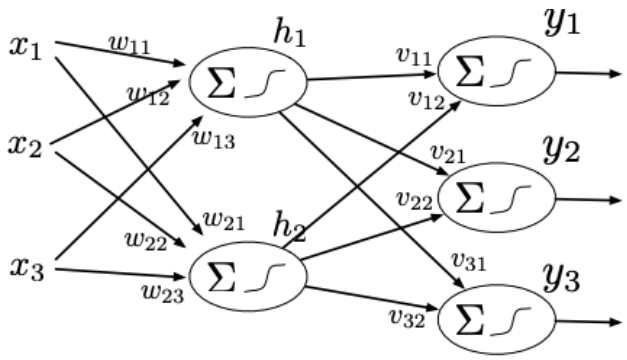
$$y = g_3 \left(\underline{v}^T \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} + v_0 \right)$$

3. Return $f(\underline{x}) = y$ as the regression output

1.2.5 Multiclass Classification

For multiclass classification, we form an output layer, with one output unit per class. The output y_i of the ith output unit gives the probability of input $\underline{x}$ belonging to class i. Again, the function in the output units will change, and we will use the **softmax** function:

$$P(y = i|\underline{x}) = \frac{e^{y_i}}{\sum_{k=1}^{n} e^{y_k}}$$



1. Pass the input through the hidden layer:

$$a_1 = g(\underline{w}_1^T \underline{x} + w_{10})$$

$$a_2 = g(\underline{w}_2^T \underline{x} + w_{20})$$

2. For $i = 1, 2, \ldots, n$, we pass the activations of the hidden layer to the ith output unit:

$$y_i = \underline{v}^T \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} + v_0$$

3. This outputs a vector $\underline{y}$. The classification is given by:

$$f(\underline{x}) = \max_{i \in [1,n]} \{P(y = i|\underline{x}\}$$

where:

$$P(y = i|\underline{x}) = \frac{e^{y_i}}{\sum_{k=1}^{n} e^{y_k}}$$

1.2.6 Modifying Feedforward Neural Networks

- **What is a feedforward neural network?**
  - the type of ANNs we have used up to now
  - they can be thought as a **DAG** (directed acyclic graph)
  - that is, information from layers moves in a given direction, and unit output can't go "back" through the network
  - each layer in the FNN computes a non-linear function of the input
- **What can we modify about a FNN?**
  - the number of hidden units
  - the number of hidden layers
  - the type of activation function g

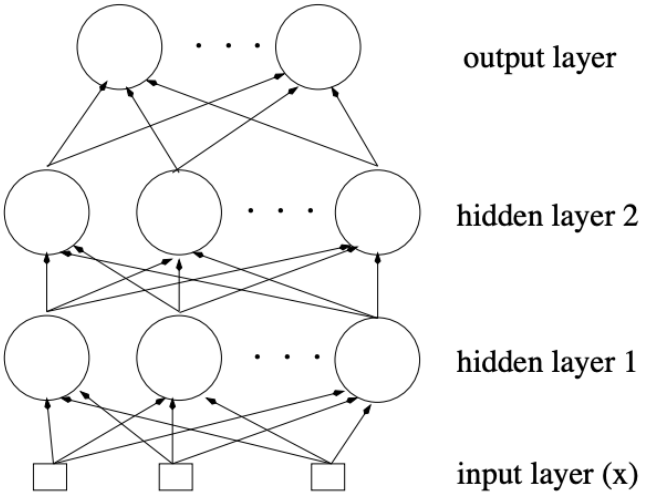


Figure 5: We can use many layers in our FNN

- **Which activation functions can be used in the output layer?**
  - for regression, the **linear** function

- for binary classification, the **sigmoid**

- **Which activation functions can be used in the hidden layers?**
  - the **sigmoid**
  - the **hyperbolic tangent** (tanh)
  - the **linear** function
  - **gaussian density** (radial basis)
  - **step** function:

$$\Theta(a) = \begin{cases} 1, & a \geq 0 \\ -1, & a < 0 \end{cases}$$

1.3 ANNs to Represent Functions

- Boolean functions:
  - Every boolean function can be represented by network with single hidden layer
  - but might require exponentially many (in number of inputs) hidden units
- Continuous functions:
  - Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
  - Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988]. This follows from a famous result of Kolmogorov.
  - Neural Networks are *universal approximators*.
  - But again, if the function is complex, two hidden layers may require an extremely large number of units
- Advanced (non-examinable): For more on this see,
  - F. Girosi and T. Poggio. "Kolmogorov's theorem is irrelevant." Neural Computation, 1(4):465469, 1989.
  - V. Kurkova, "Kolmogorov's Theorem Is Relevant", Neural Computation, 1991, Vol. 3, pp. 617622.

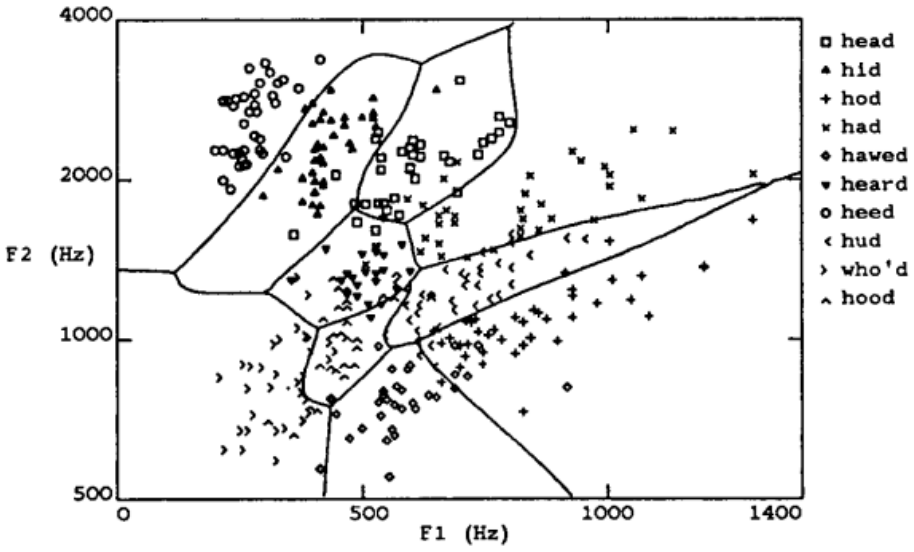


Figure 6: The decision boundary of an ANN when dclasifying vowel sounds, based on frequency information.

1.4 Training ANNs

- **What do we train in ANNs?**
  - for each unit in the network, we seek to find the best set of weights and biases
  - the training problem can be converted into an **optimisation** function, by minimising a loss function
- **Which loss functions can be used?**
  - the general idea is to compare the prediction with the true labe
  - in linear regression, we used the sum of squared errors:

$$E = \sum (y_i - f(\underline{x}_i)^2$$

  - in logistic regression, we use log loss:

$$E = \sum y_i \log f(\underline{x}_i) + (1 - y_i) \log(1 - f(\underline{x}_i))$$

  - others include maximum likelihood, or cross-entropy
  - a regularisation paramer ($\lambda \|\underline{w}\|^2$) can also be used (known as **weight decay** in the context of ANNs)

  - however, unlike with linear or logistic regression, the optimisation problem does not have a single optimum: for example, the same network can be represented by permuting the units; this should not change the accuracy, but techincally changes the network

- **What is backpropagation?**
  - Math behind backpropagation
  - backpropagation is teh algorithm used to apply gradient descent during optimisation
  - for this, we need to compute $\frac{\partial E}{\partial \underline{w}}$, which is non-trivial for a ANN
  - backpropagation employs the chain rule, alongside the layered structure of the network, to determine this (for example, if we want to compute $\frac{\partial E}{\partial v_0}$, we note that this will ultimately depends on the derivative of the set of weight at the hidden layer, like $\frac{\partial E}{\partial w_{11}}$)
  - backpropagation thus uses previous derivatives to compute derivatives further along the network, allowing speedup in training

- **Does backpropagation converge?**
  - yes, but it can do so to local minima
  - this can be alleviated by training the network multiple times, and selecting the best model (or combining models)
  - From slides: Initialize weights near zero; therefore, initial networks are near-linear; Increasingly non-linear functions possible as training progresses]
  - Article on initialising weights

  - Optimize over vector of all weights/biases in a network
  - All methods considered find *local* optima
  - Gradient descent is simple but slow
  - In practice, second-order methods (*conjugate gradients*) are used for batch learning
  - Overfitting can be a problem

1.5 Applications of Neural Networks

- handwritting recognition
- speech recognition
- financial forecasting

1.6 Recent Developments

1.6.1 Convolutional Neural Networks

- **When are CNNs used?**
  - determining if an object is present in the image
  - set a bounding box on an object in an image
  - identify pixels belonging to object in image
- **Why are CNNs used?**
  - when analysing images: vector representations of images are impractical (large, lose of spatial information)
  - CNNs mitigate this problem, by compressing features of the image data, without losing information
- **What is a convolution?**
  - a mathematical operation used to extract feature information from a set of pixels
  - for example, the Sobel operators, to detect edges in pictures
- **How do CNNs detect features?**
  - the hidden layers are made up of a set of convolution operators (these are the **weight matrices** which we train)
  - this means that if we are given the image as a 2D matrix, the output of the hidden layer is a a 3D cube, resulting from applying different convolution kernels on the image
  - this turns the original image into a set of matrices which focus on different features. These matrices (known as **feature maps**) will be lower dimensional

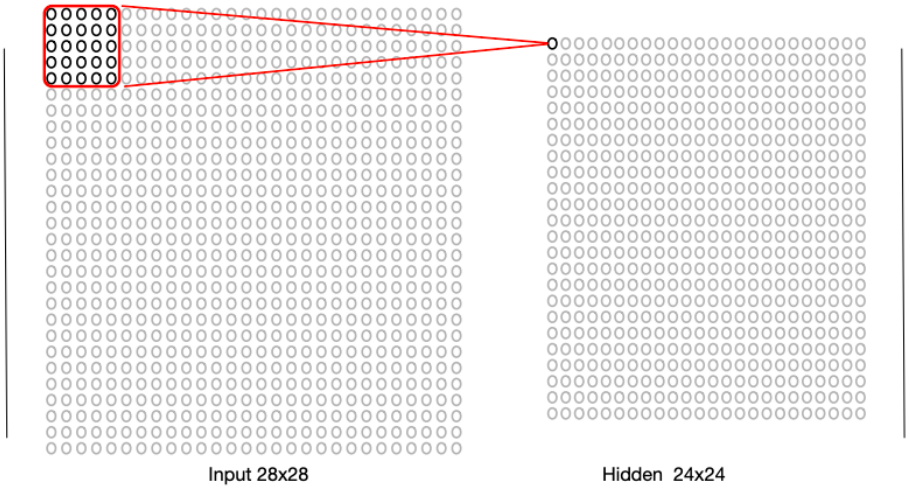


Figure 7: Applying a convolution to the image reduces it in size. If we apply different convolutions, we obtain matrices which represent different features of the image.

- Constrain each hidden unit $h_{i,j}$ to extract the feature by sharing weights across the receptive fields
- For hidden unit $h_{i,j}$ and $m \times m$ weight matrix W

$$h_{i,j} = \text{sigmoid}\left(\sum_{k=0}^{m-1}\sum_{\ell=0}^{m-1} w_{k,\ell} x_{i+k,j+\ell} + b\right)$$

- The output is a *feature map*
- Many different weight matrices can be used, to produce multiple feature maps

Figure 8: The formula shows how the hidden unit $h_{i,j}$ (the jth unit of the ith layer) considers an $m \times m$ pixel region, and applies the weights and bias of the operator on the region. Notice, the set of weights and biases will be the same over all the units in the layer. Applying this on the image it reduces each $m \times m$ set of pixels into a single value, whicha re then combined to form the feature map.

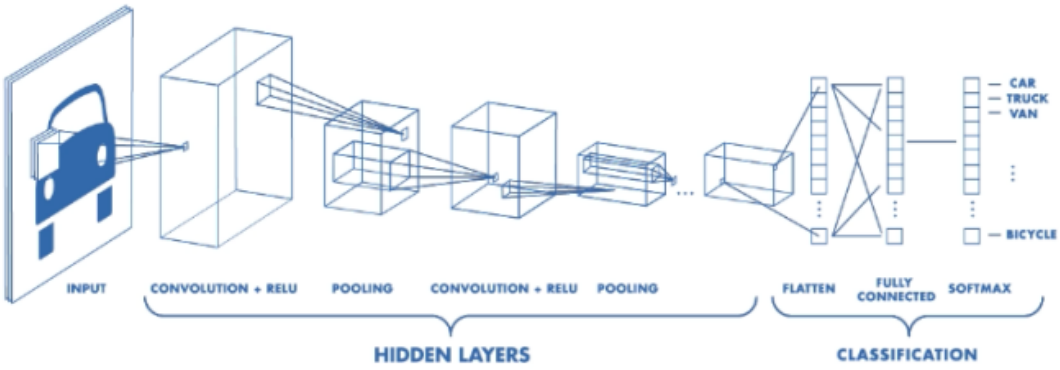


Figure 9: The general structure of a CNN works in 2 steps. The first step focuses on feature extraction: each hidden layer is composed of a set of convolutions (to extract feature maps) and pooling (to reduce the size of the feature maps, by picking certain feature values). This is repeated, squashing the feature maps more and more. At the second step, the squashed feature maps are then converted into vectors (which are more manageable in size than if we took the whole image and turned it into a vector). These vectors encode the underlying structures that define the image. The vectors are then passed through a standard classifier (connected NN, with its standard wieghts and biases). Finally, softmax can be used to output a classification (for example).

1.6.2 Recurrent Neural Networks

- **When are RNNs used?**
  - when we have to model data which comes in sequences

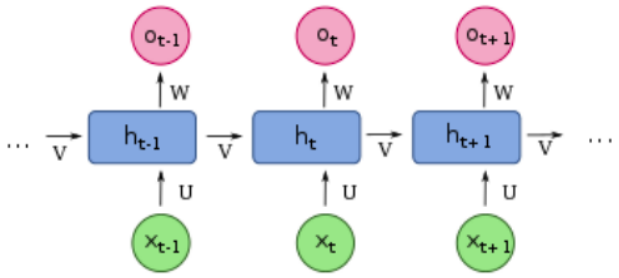


Figure 10: The inputs are the $\ldots, x_{t-1}, x_t, x_{t+1}, \ldots$. The input to the hidden units is not only the previous hidden units ($h_{t-1} \to h_t$), but also part of the data input (that is h_t gives an output o_t given what has previously happened [represented using the hidden unit h_{t-1}], and what is happening [represented using the input x_t]).

- used to for example predict an event,given all previous events
- predict the next word in a sentence
- translate sentences
- predict weather
- annotate motion capture data (label the action being performed, like running or walking)
- tag speech (i.e adjective, verb, noun) in NLP
- LSTM (long-short term memory) is a complex RNN