slidenumbers: true
footer: A. Ferrara. **Language models**. *Part 5: Neural Language Models*. email, corse website, slack, github

Information Retrieval

[fit]Language models

Part 5: Neural Language Models. Prof. Alfio Ferrara

Master Degree in Computer Science

Master Degree in Data Science and Economics

[fit] Recap on the goal of language models

We know that the goal of language models is to estimate

$$P(w_i \mid w_1, \ldots, w_{i-1})$$

When a sequence of words is too much long for making it possible to perform a robust estimation, we just approximate the task by

$$P(w_i \mid w_{i-n+1}, \ldots w_{i-1}) \approx P(w_i \mid w_1, \ldots, w_{i-1})$$

The idea of neural language models is just to train a neural network to perform such an estimation

---

(Very fast) Introduction to neural networks

Neural Networks are composed by two or more **layers of nodes**, where each layer output provides an input for the subsequent layer. Such an input is combined with **weights** associated with the node connections before feeding the subsequent layer. In general terms, a NN can be seen as a tool for computing a combination of linear and non linear transformations of the form

$$\hat{\mathbf{y}} = \phi(W\mathbf{x} + b)$$

where $\hat{\mathbf{y}}$ ia vector representing the **network prediction**, $\mathbf{x}$ is the **input vector** (the data), W is a **matrix of weights** (that the network aims at learning), and $\phi(\cdot)$ is a (potentially) non linear transformation applied to data before the final prediction is returned, often called **activation function** [1].

Learning

The NN main goal is to find (**learn**) the value of its **parameters** Θ (the weights W and the bias b) that make it possible to obtain a prediction $\hat{y}$ as much close as possible to the real output y, which is known from the examples available in the **training set** (**supervised learning**).

To perform learning, several **iterations** are tried starting with random values for the weigths. For each iteration we update weights and we base the subsequent iteration on the feedback provided by the **error in the prediction**, with the goal of **minimizing the error**. A crucial component of NN is thus to have a function that measures the error, called **loss function**. Given t as one of the iterations, η being the learning rate, $L(\cdot)$ the loss function, and X represents input data The general form of the learning step is

$W^{t+1} \leftarrow W^t + \eta L(X)X$

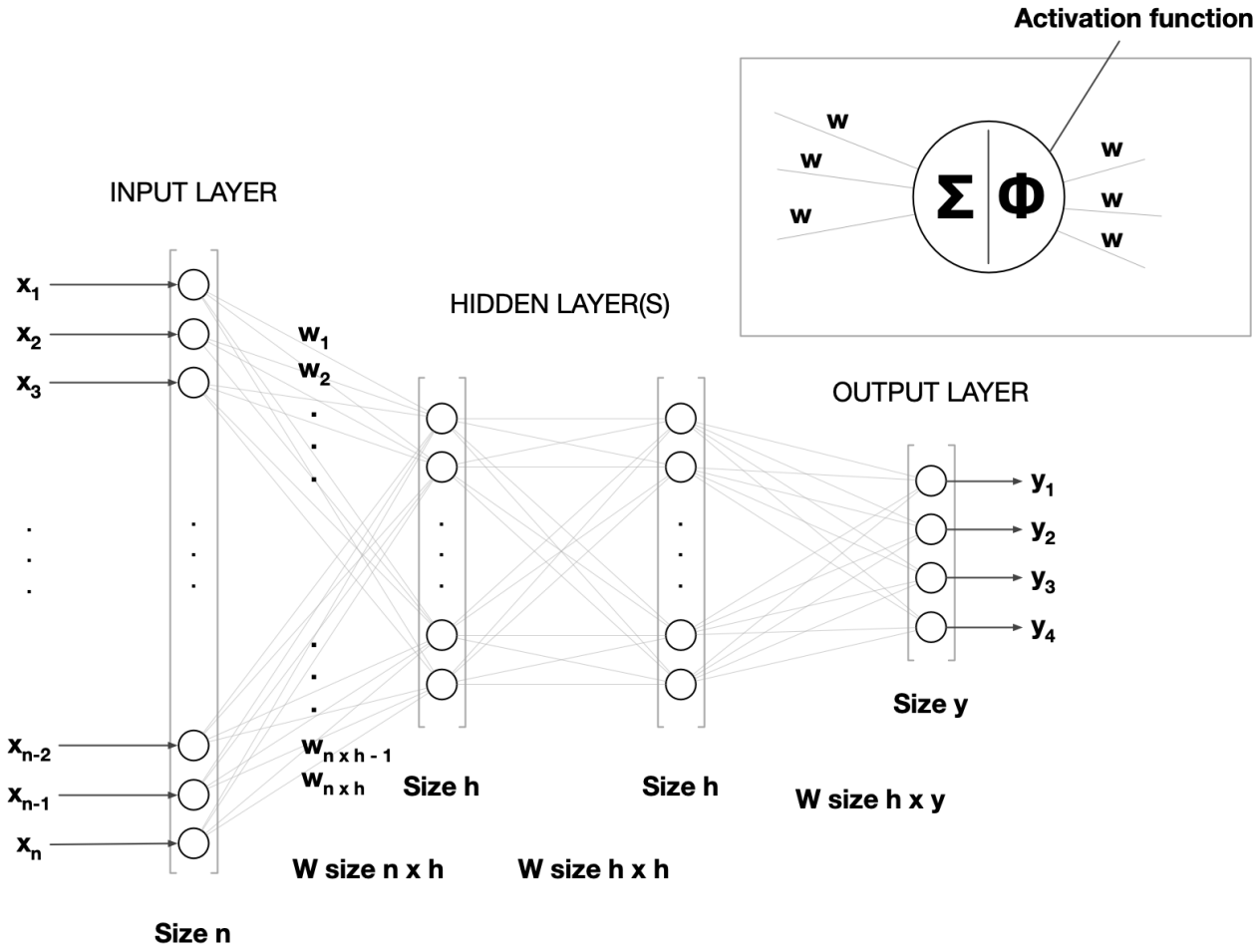
Overview

When neural networks are used for **textual data**, usually we have:

Input is **word vectors**, either sparse or dense

Dense vectors for words can be given as input (i.e., used a pre-trained models, such as **word embeddings**, **LDA**) or can be calculated by the network itself as part of the training process, strating from a sparse word representation (such as **one-hot encoding**, **co-occurrence count**, **n-grams matrix**)

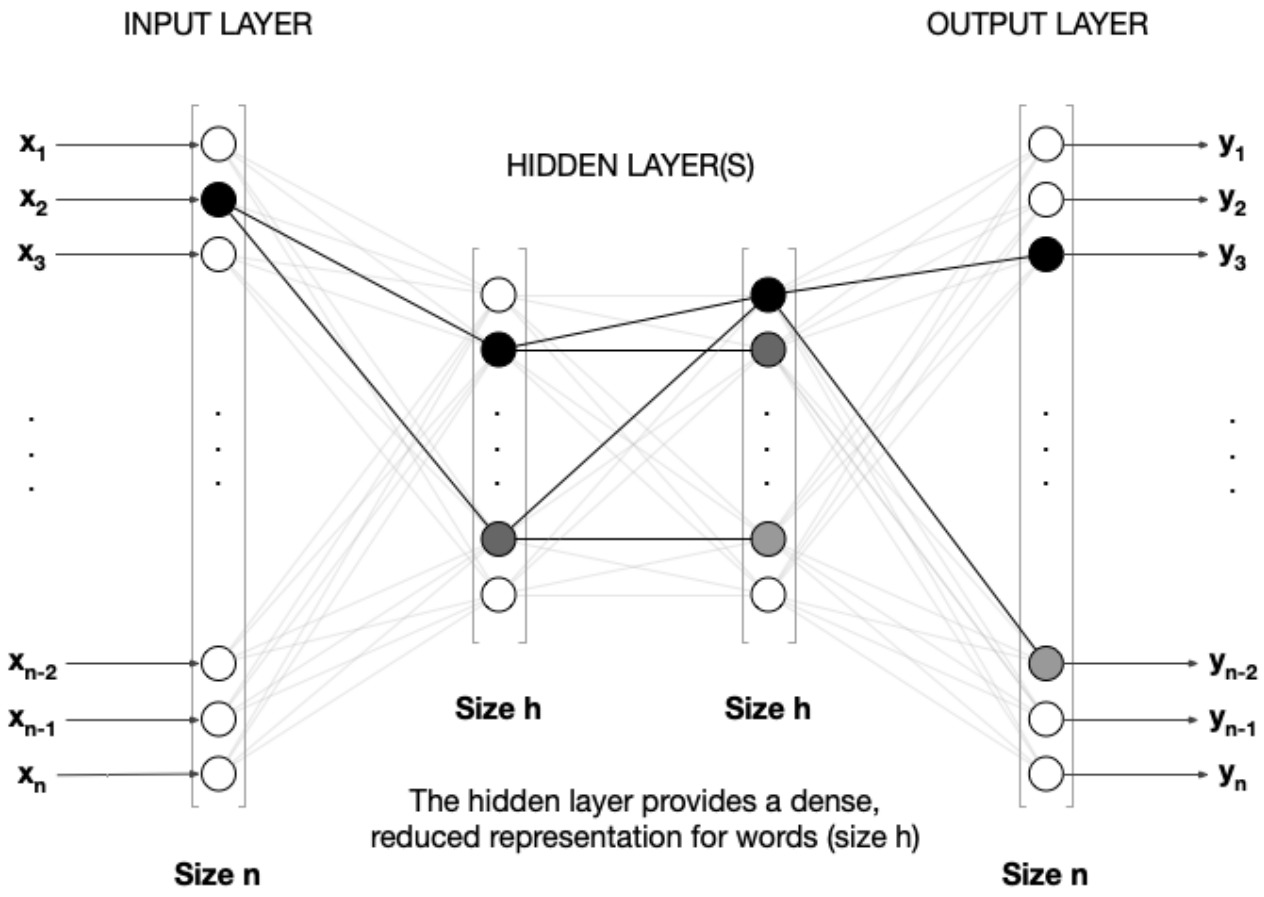
The output depends on the task (**multi-class** or **multi-label classification**, **autoencoders**)



Autoencoder

Autoencoders are special network architectures where an input (represented by an input layer of size n) is mapped onto itself (an output layer of size n), such as when a network is used to map words on other words

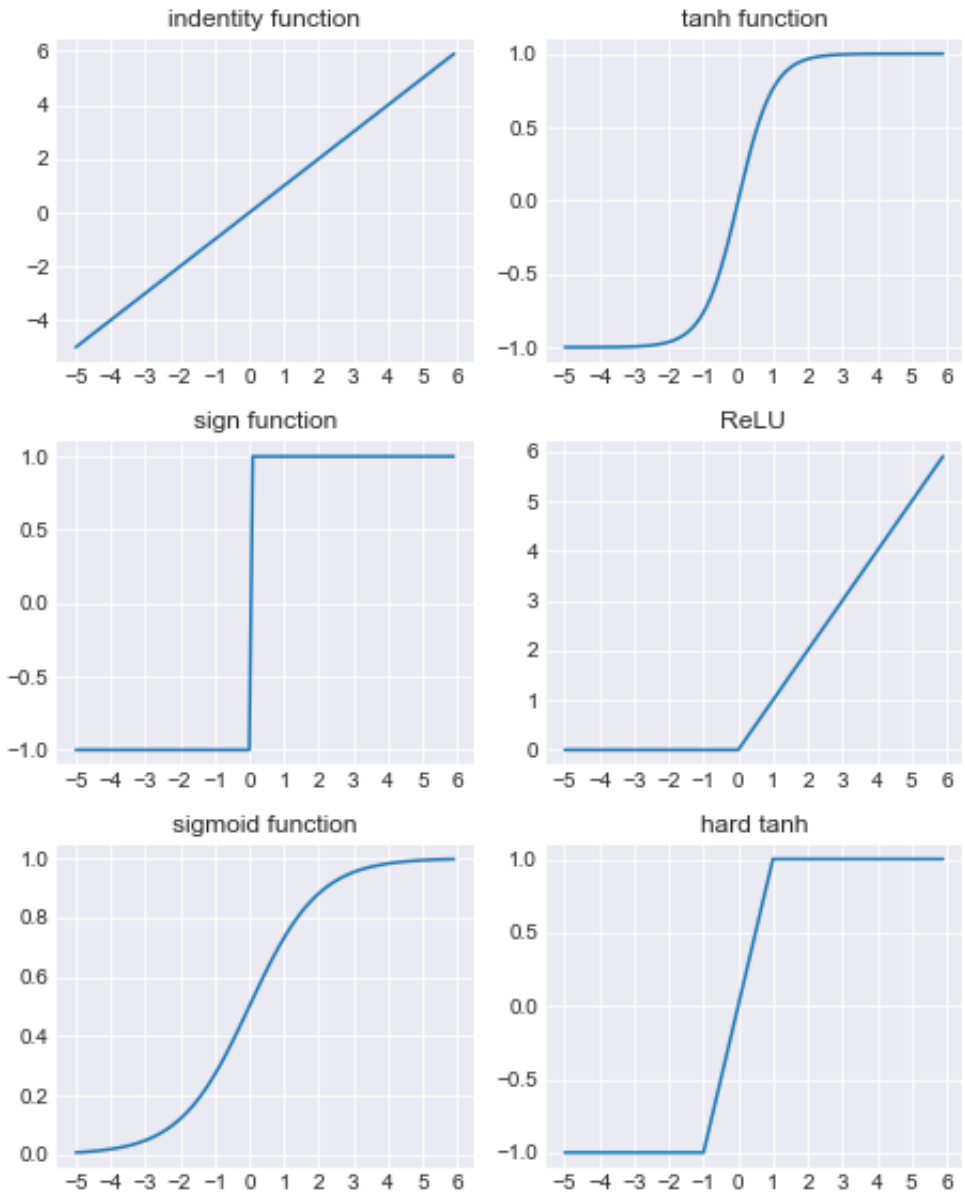
An example that we have seen is Word2vec. An interesting side-effect of this architecture is that we can take the hidden layer as a reduced dense represetation of the input (as in word embeddings)



Activation function

There are many options for the choice of the activation function $\phi(\cdot)$ having that $\hat{y} = \phi(Wx)$

- $\phi(x) = x$ (indentity)
- $\phi(x) = x^+$ (sign)
- $\phi(x) = \frac{1}{1+\epsilon^{-x}}$ (sigmoid)
- $\phi(x) = \max\{x, 0\}$ (**Re**ctified **L**inear **U**nit)
- $\phi(x) = \frac{\epsilon^{2x}-1}{\epsilon^{2x}+1}$ (tanh)
- $\phi(x) = \max\{\min\{x, 1\}, -1\}$ (hard tanh)



Output nodes

When we need to predict one (or more) outputs among multiple options, such as in case of multi-class (or multi-label) classification or with autoencoders, a very common choice is to model the output layer as a **softmax** layer, in order to enforce a probabilistic interpretation of the results.

Having an output with n dimensions:

$$\phi(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp x_j} \; \forall i \in \{1, \dots, n\}$$

Loss functions

The use of *softmax* produces a probabilistic output, which requires also a some constraints on the choice of the loss function

Binary target (logistic regression):

$L = \log(1 + \exp(-y \cdot \hat{y}))$

Categorical targets (cross-entropy loss): given $\hat{y}_1, \ldots, \hat{y}_k$ as the probabilities predicted for each of the k targets, assume that $r \in \{1, \ldots, k\}$ is the correct target according to the training set for the instance under evaluation. Then the cross-entropy loss is

$L = -\log(\hat{y}_r)$

Multilayer networks (I)

When a netwoork has multiple hidden layers, you can see it as a composition of functions of the form $h_n(\ldots h_1(f(x)))$, where each function h_i corresponds to the ith hidden layer. However, this makes training more difficult, because the error gradient has to be backpropagated through the layers.

The algorithm performing training is articulated in two phases, a *forward phase* and a *backward phase*.

Multilayer networks (II)

Forward: The input produces a forward cascade of computation across the layers. The final output is compared with the training set expected output and the **derivative of the loss function is computed**.

Backward: Denote w_{h_{r-1},h_r} as the weights of the transition between layer h_{r-1} and h_r and consider to have $h_1, \ldots, h_k$ hidden layers before the output layer o. The Loss function derivative is decomposed along the path from h_1 to h_k as follows

$$\frac{\partial L}{\partial w_{h_{r-1},h_r}} = \frac{\partial L}{\partial o}\left[\frac{\partial o}{\partial h_k}\prod_{i=r}^{k-1}\frac{\partial h_{i+1}}{\partial h_i}\right]\frac{\partial h_r}{\partial w_{h_{r-1},h_r}} \quad \forall r \in 1 \ldots k,$$

Where the component $\frac{\partial o}{\partial h_k}\prod_{i=r}^{k-1}\frac{\partial h_{i+1}}{\partial h_i}$ has to be aggregated (summed) for each pattern of nodes connecting h_r to o.

Naive examples

Two very simple examples of how to implement a neural network using `PyTorch` are provided in the GitHub repository of the course:

- *Spelling correction*: very simple example on how to use character-based words processing for implementing spelling correction. Introduces some basics about PyTorch (link)
- *Language model*: Implements a neural language model, showing how to exploit pre-trained word vectors and by comparing different network architectures (link)

[fit]Recurrent Neural Networks (RNN)

When dealing with *sequence-to-sequence* learning, we aim at predicting the value s_i in a sequence (e.g., the ith word in a text) given the previous $s_{i-n+1}, \ldots, s_{i-1}$ sequence elements (e.g., the previous words).

The issue here is that with a feedforward network, each prediction $\hat{s}_i$ may be based on data about the previous elements in the sequence (such as for n-gram models), but it is **independent** from the previous predictions of the network itself.

The idea of **Recurrent Neural Networks (RNN)** is instead to use the output $\hat{s}_i$ of the network on the instance s_i as an input (together with data) for the subsequent prediction(s) $\hat{s}_{i+j}$

Architecture of RNN

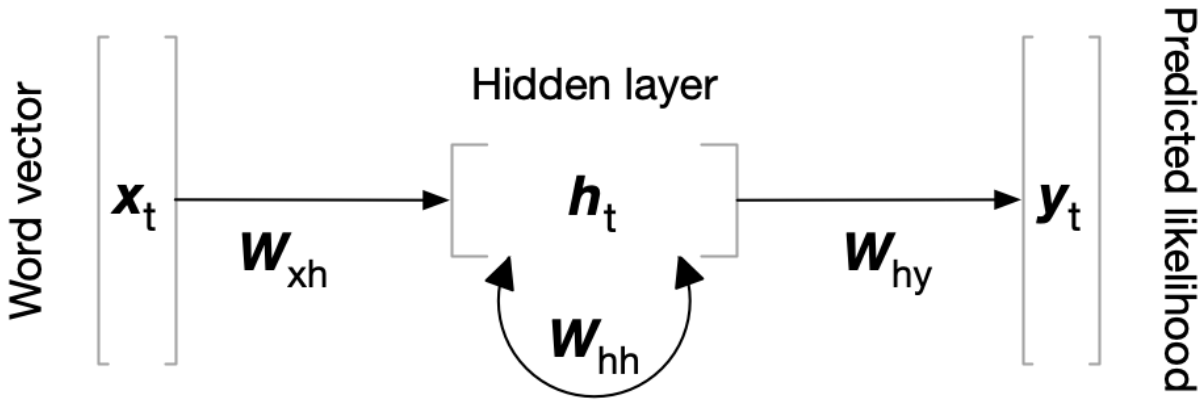
The basic idea is that the state of the hidden layer h_t at time t if a function of the form

$h_t = f(h_{t-1}, x_t)$

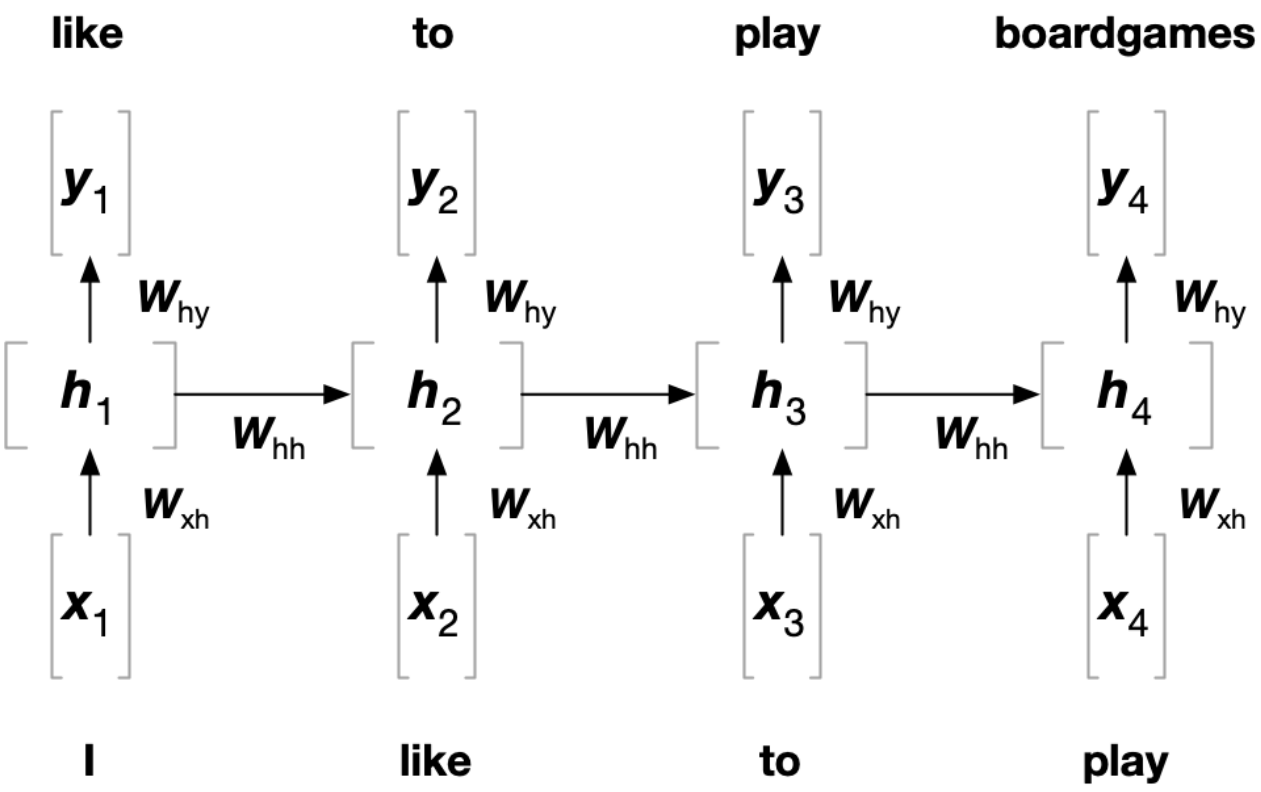
Given d and the data dimensions (e.g., the vocabulary for language models) and p as the dimension of the hidden layers, we will have to work with three matrices of weights: $W_{xh} \in \mathbb{R}^{p \times d}$, $W_{hh} \in \mathbb{R}^{p \times p}$, and $W_{hy} \in \mathbb{R}^{d \times p}$ for input to hidden layer, hidden layer to hidden layer, and hidden layer to output. Thus we will have:

$h_t = \tanh(W_{xh} x_t + W_{hh} h_{t-1}), \quad y_t = W_{hy} h_t$

RNN general architecture



RNN in time



RNN training

The softmax probabilities of the correct words at various time-stamps are aggregated to create the loss function.

The backpropagation algorithm is updated to **backpropagation through time (BPTT)**.

1. running the input sequentially in the forward direction through time and computing the error/loss at each time-stamp (same as BP)

2. computing the changes in edge weights in the backwards direction on the network without any regard for the fact that weights in different time layers are shared (same as BP)
3. adding all the changes in the (shared) weights corresponding to different instantiations of an edge in time (BPTT specific)

Intuition of Long Short Term Memory networks (LSTM)

In RNN a common problem is that successive multiplication by the weight matrix is highly unstable (*vanishing/exploding gradients* problem).

This problem is an issue expecially for long sequences, requiring the network to have a *long memory*.

To address the problem, in LSTM we introduce a new hidden vector of p dimensions, referred to as the **cell state**. The cell state is a kind of long-term memory that retains at least a part of the information in earlier hidden states by using a combination of partial *forgetting* and *increment* operations on previous cell states.

See further details in Section 10.7.7.1 of Aggarwal, C. (2018). *Machine learning for text* (pp. 3121-3124). Cham: Springer International Publishing. pdf.

Applications

Language models can be used for addressing several different tasks. Main applications include:

- Text generation
- Text classification
- Sequence learning
  - Text tagging
  - Automatic translation

Text generation

The process of text generation can be implemented directly on the capability of the model to evaluate $P(w_i \mid w_{i-n+1}, \ldots, w_{i-1})$. Suppose to have a 2-gram model M. Let's denote $\hat{\mathbf{y}}_\mathbf{j}$ the distribution of probabilities over the vocabulary V predicted for the w_{j+1} word. Text can be generated as

```
text = ['#S']
while text[-1] != '#E':
  y_text[-1] = model.predict(text[-1])
  new = sample from V with probability y_text[-1]
  text.append(new)
```

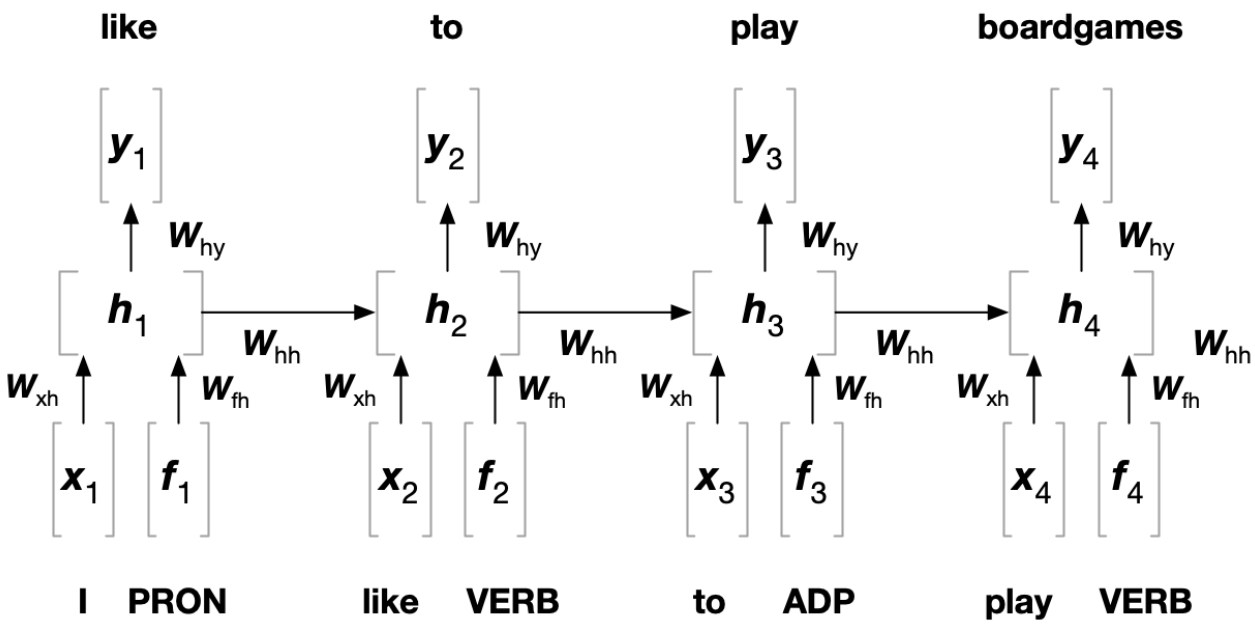
See an example on inforet

Incorporating linguistic features (such as POS)

Linguistic features may be incorporated in text generation (and also in other tasks) in two ways:

Naive: instead of working with words, we can word with tokens that include the linguistic information, such as `cat_NOUN`. This is the approach used in out example.

Special network (try this as an exercise)



Text classification

Given a set of K of classes (also called target labels) and a corpus of documents D, text classification is the task of learning a mapping $\mathcal{M}(D) \to K$. The classification task may be

Binary classification: $|K| = 1$ (determine whether a document is mapped on the class or not)

Multiclass: $|K| > 1$ and $|\{k \in K : \mathcal{M}(d) \to k\}| = 1 \; \forall d \in D$ ($\mathcal{M}(D)$ is a partition of the corpus)

Multilabel: $|K| > 1$ and $|\{k \in K : \mathcal{M}(d) \to k\}| \geq 1 \; \forall d \in D$ (a document can be associated with more than one class)

Text classification

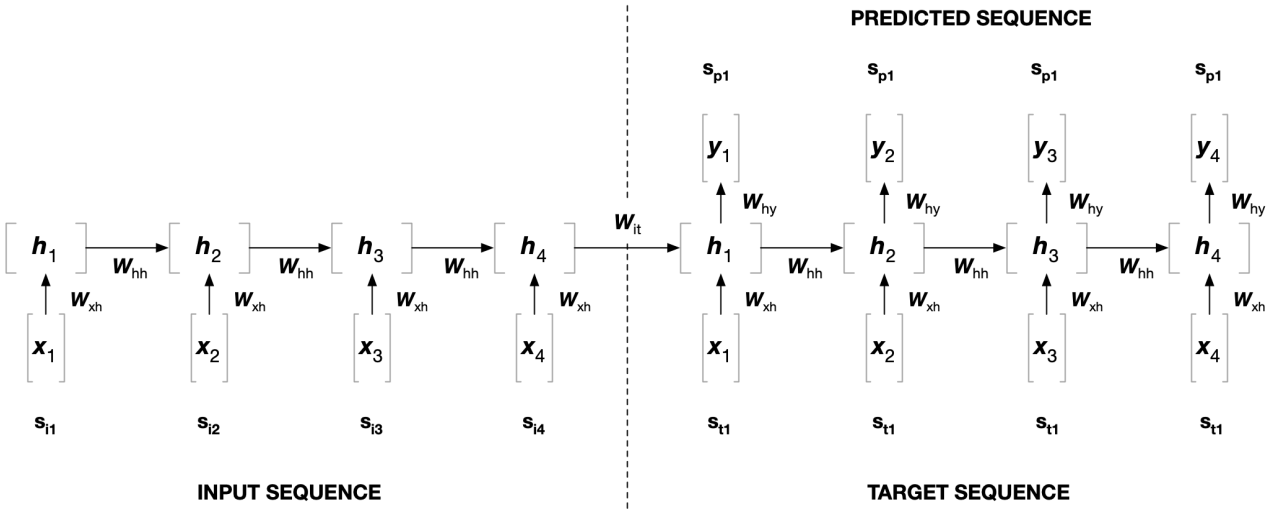
Language models can be used to address a classification task in two ways:

- Estimate $P(k \mid w_1, \ldots, w_n)$ (see example on inforet)
- Define a language model L_k for each class k and than use it to estimate the probability of a document d to be in k as

$$P(k \mid d) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-n+1}, \ldots, w_{i-1})$$

Sequence learning

RNN and LSTM can be used to map a sequence s_i onto another target sequence s_t. The general idea is to combine two language models as in figure



Examples

Sequence-to-sequence learning can be used for many tasks

- **POS tagging**: `[I, play, boardgames] => [PRON, VERB, NOUN]` (see inforet)
- **Entity recognition**: `[Michael, Stipe, was, born, in, Decatur, in, 1960] => [PERSON, PERSON, NONE, NONE, NONE, PLACE, NONE, DATE]`
- **Machine translation**: `[Michael, Stipe, was, born, in, Decatur, in, 1960] => [Michael, Stipe, è, nato, a, Decatur, nel, 1960]`
- **Dialogue generation and question answering**: `[Hi, my, name, is, Jack] => [Nice, to, meet, you]` (see inforet)

1. For references on this lecture see Aggarwal, C. (2018). *Machine learning for text* (pp. 3121-3124). Cham: Springer International Publishing. pdf. Some examples and figures in these slides are taken from the book. ↩