Representation Learning

11-785 / Spring 2019 / Recitation 10-a

Raphael Olivier

Introduction

In a Machine Learning task, you need to learn :

- A good set of features to represent your data
- A classifier on those features

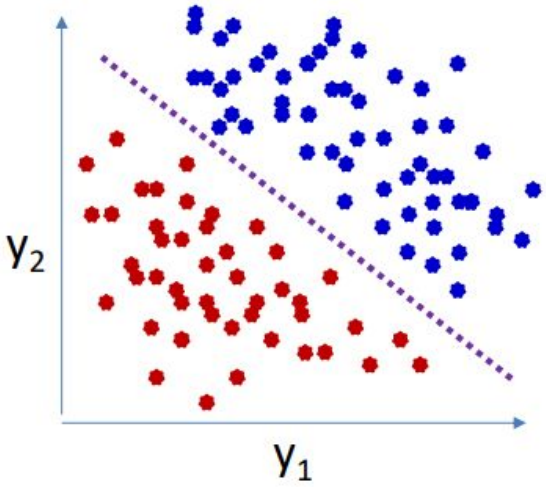
Today we are interested in the former

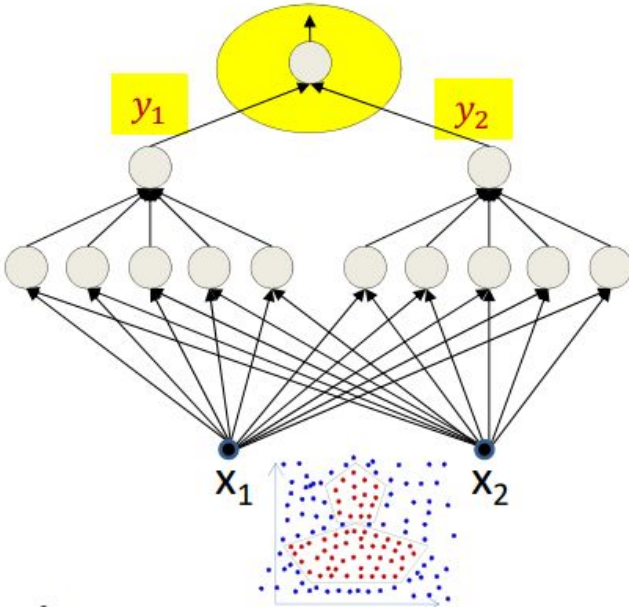
If you design the features manually, it is called Feature Engineering (very important in SVM, decision trees, NLP algorithms, etc)

If you do it automatically, it is **Feature Learning** or **Representation Learning**.

Recap

In the lectures, you have seen all layers but the last one actually just learn a separable representation of your data, and the last one classifies…



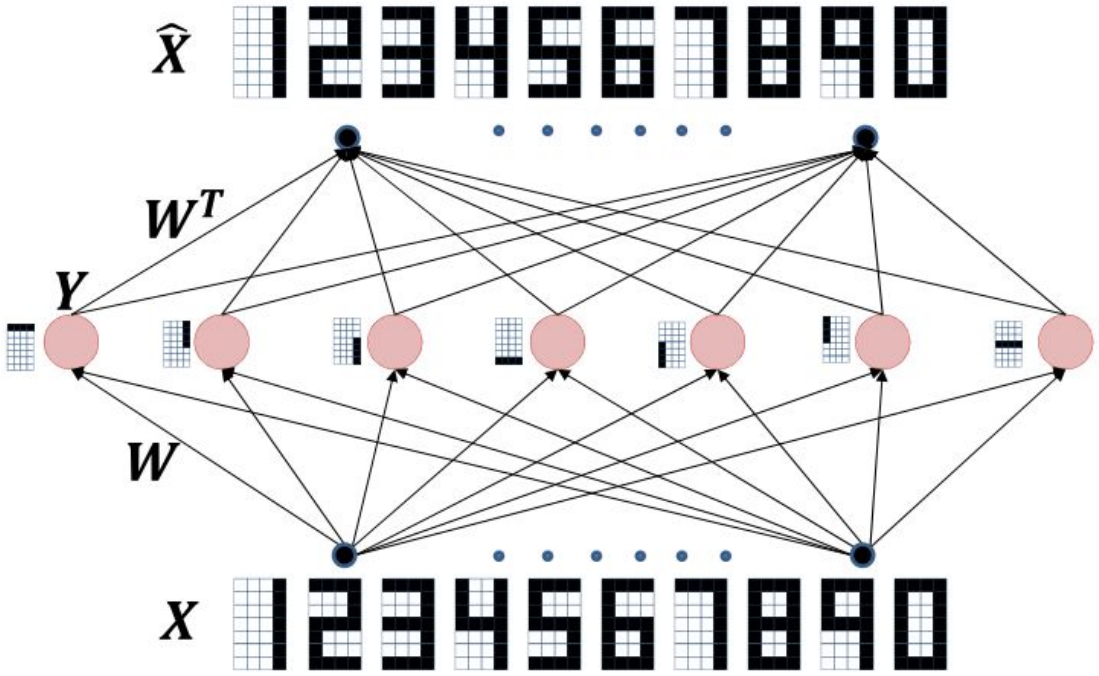


… i.e. Deep Learning merges representation and classification into a unique task !

Recap

But with neural nets you can also learn representations “by themselves”, i.e. in an unsupervised setting.

For that we use **Autoencoders** (among other things).



This is pretty much a **non-linear PCA**, trained with Reconstruction

Plan for today

For this (half) recitation we will go deeper into representation learning. That half has no code but some practical advice (and abstract advice too)

This could be useful for a large number of projects (like all recitations from now on).

The second half will be about VAEs, which is a different topic (although linked)

Plan

- “Think” in representations
- Unimodal representations
- Multimodal representations

Representation learning is a mindset

Q : How to know the “right” architecture for a given task ?
(Asked on Piazza ~60 times)
“A” : You have only two things to consider :

- Optimization : can my model learn by SGD and generalize well
  (→ depth, skip-connections, dropout, batchnorm,etc)

- Representation : can my model capture the “inner structure” of the data ?

These two are partly complementary, partly supplementary.

Representation learning is a mindset

Why are CNNs good for images ?
→ because features you need in image tasks tend to be translation-invariant
Why are RNNs good for sequences ?
→ because features you need in language tasks have long-term dependencies
Why is attention useful in Seq2Seq ?
→ Because a decoded word's representation should be correlated to some specific input words.

Rather than thinking in "tricks" or academic recipes, think of what design makes sense w.r.t the properties you seek in your representation.
Ex : for sentiment analysis, CNNs are often better than RNNs

How to learn a representation

- End-to-end (what you usually do)
- In an unsupervised fashion (autoencoders)
- On an alternate task
- Use a pretrained model (Ex: pretrained word embeddings)

If you use a representation learned one way and move on to the task you're really interested in, you can :

- Fine-tune the representation
- Fix it, and add deeper layers

Again, think about what makes most sense based on the data you have and the task you're interested in.

Some examples

Transfer learning

Train a neural network on an easy-to-train task where you have a lot of data.

Then, change only the final layer fine-tune it on a harder task, or one where you have less data.

Ex : **HW2p2**, if you train a network on classification then fine-tune it with triplet loss for verification. The first task's representation is probably useful for the second.

Some examples

Semi-supervised learning

Train both an unsupervised model and a supervised one, with or without shared parameters.

Example : you want to learn to translate from English to French. You have a lot of text in French, a lot of text in English, **but** a limited amount of aligned French-English data.

A seq2seq model with attention could only learns on the aligned data.

What do you do ?

Some examples

A possible solution :

- Train the encoder as a language model in English
- Train the decoder as a language model in French
- Train the attention weights (1 or 2 matrices only) on the translation task.

Some examples

A more complex solution :

- Use the encoder as the encoding part of a English-English autoencoder
- Use the decoder as the decoding part of a French-French autoencoder

(Semi-Supervised Learning for Neural Machine Translation, Cheng et al., ACL 2016)

$$
\begin{aligned}
& J(\overrightarrow{\boldsymbol{\theta}}, \overleftarrow{\boldsymbol{\theta}}) \\
= & \underbrace{\sum_{n=1}^{N} \log P(\mathbf{y}^{(n)}|\mathbf{x}^{(n)}; \overrightarrow{\boldsymbol{\theta}})}_{\textit{source-to-target likelihood}} \\
& + \underbrace{\sum_{n=1}^{N} \log P(\mathbf{x}^{(n)}|\mathbf{y}^{(n)}; \overleftarrow{\boldsymbol{\theta}})}_{\textit{target-to-source likelihood}} \\
& + \lambda_1 \underbrace{\sum_{t=1}^{T} \log P(\mathbf{y}'|\mathbf{y}^{(t)}; \overrightarrow{\boldsymbol{\theta}}, \overleftarrow{\boldsymbol{\theta}})}_{\textit{target autoencoder}} \\
& + \lambda_2 \underbrace{\sum_{s=1}^{S} \log P(\mathbf{x}'|\mathbf{x}^{(s)}; \overrightarrow{\boldsymbol{\theta}}, \overleftarrow{\boldsymbol{\theta}})}_{\textit{source autoencoder}},
\end{aligned}
$$

Unimodal representations

Q : What are some good pretrained, general-purpose representations I can use ?

A : Depends on your **modality** : Image, Text, Audio, Video….

Image representations

Very often people use models trained on **ImageNet**.
Ex : all the pretrained models you find in the torchvision package

Recipe :
If you want to train a domain-specific classifier, consider loading the pretraining model then fine-tuning on your dataset.
If you want to train another general task (ex: Visual Question Answering), you can simply encode all your dataset with the pretrained model at once, then learn a network that takes that representation as input

Pay attention to the specific format of images !

Text representations

These are the famous **Word embeddings** : word-level representations trained on some objective.

Some are strictly word-level, i.e. they are just a matrix for a given vocabulary (Ex: Word2Vec trained on google news)
Others are contextual, i.e. each word representation depends in real time of the other words in the sentence (Ex: Elmo, FastText, BERT, MS-DNN)... Essentially those are language models.

Recipe : the package AllenNLP, built on Pytorch, contains Elmo which is good. Great package, look it up. If you want state-of-the-art (and only then) check BERT or MSDNN.

Audio representations

Those are less standard.

VGGish is a pretrained sound encoder (https://github.com/tensorflow/models/tree/master/research/audioset) in tensorflow. You can consider using it like ImageNet models, to encode your entire dataset beforehand.

Video representations

Use separate representations for audio and visual.

For audio you can use VGGish.

For the visual modality, the simplest (and most standard) way to do it is by separating images : take frames separated by a fix time interval from the video and encode each with an ImageNet model.

Then you can either take the mean or keep the sequence.

Multimodal representations

Say that I have a task involving 2 or more modalities.

Ex : Visual Question Answering. Given an image and a question about it, find the answer.

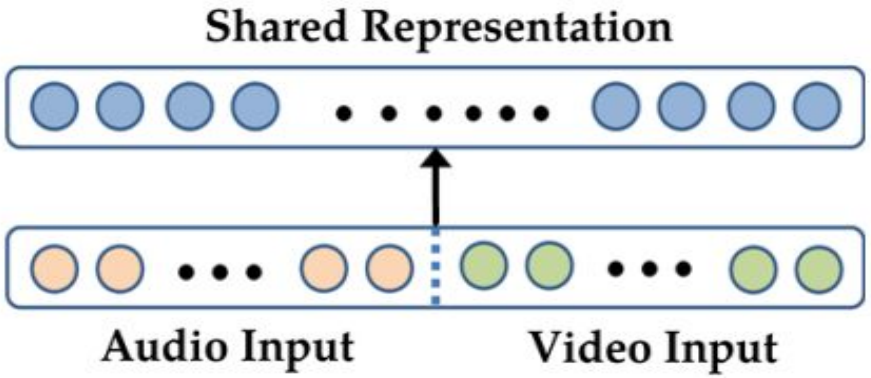
I can easily obtain the unimodal representations for each input.

But what then ?

(Credits for images and a lot of content go to the *11777 Multimodal Machine Learning* course.)

Shallow representations

Simply concatenate representations and plug that in your end-to-end network.



Advantage : There's basically nothing to do

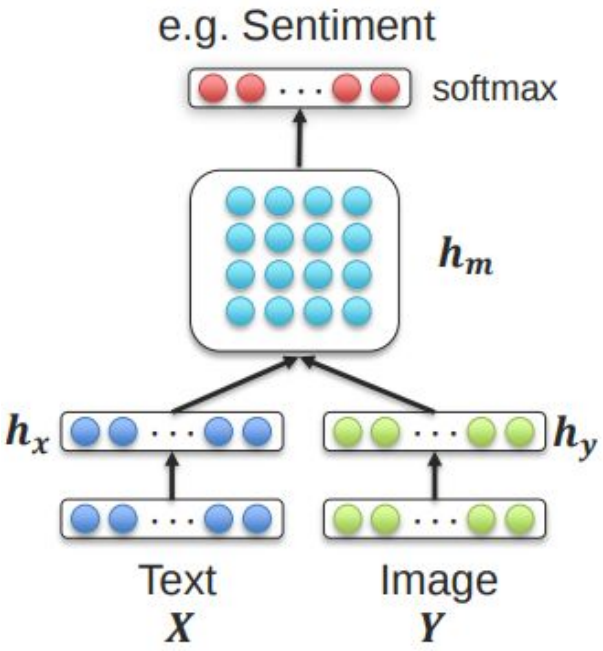
Drawback : It doesn't capture much interactions between modalities → the weakest representation

Bilinear pooling

Use an outer product on unimodal representation vectors to get a (large) multimodal vector.

Advantage :

- Captures every pattern
- Great for **complementary** modalities, i.e. when they contain different information



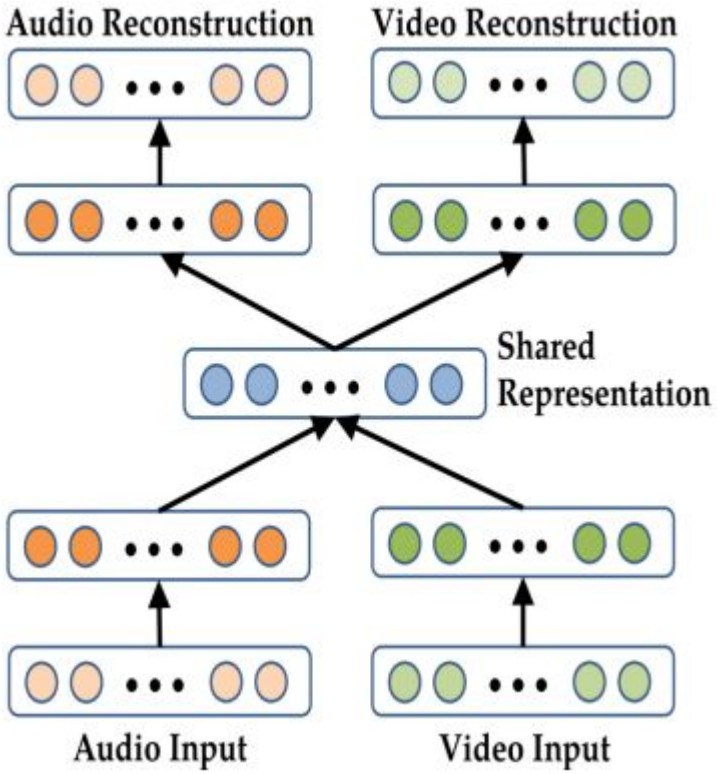
Problems:

- Can be more rich than needed, especially when modalities contain redundant (supplementary) information
- Untrackable. But there are improvements like Compact Bilinear pooling or Tensor Fusion. Look it up !

Autoencoders !

Encode modalities in a shared space.

Train unsupervised. Then when training the downstream task keep only the encoder part.



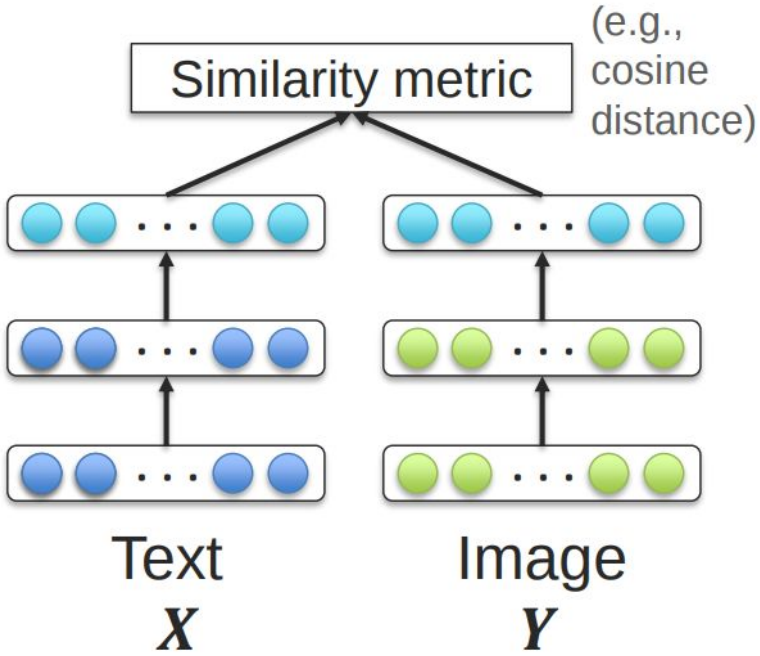
Advantages : robust in many contexts. If properly trained it can reconstruct missing modalities !

Problems : Needs separate training, and often not state-of-the-art compared to pooled or coordinated representations.

Coordinated representation

Instead of using a shared representation, keep them separated but enforce that they are related to each other.

Specifically, enforce a *correlation* objective based on multivariate stats
→ **Canonical correlation analysis**

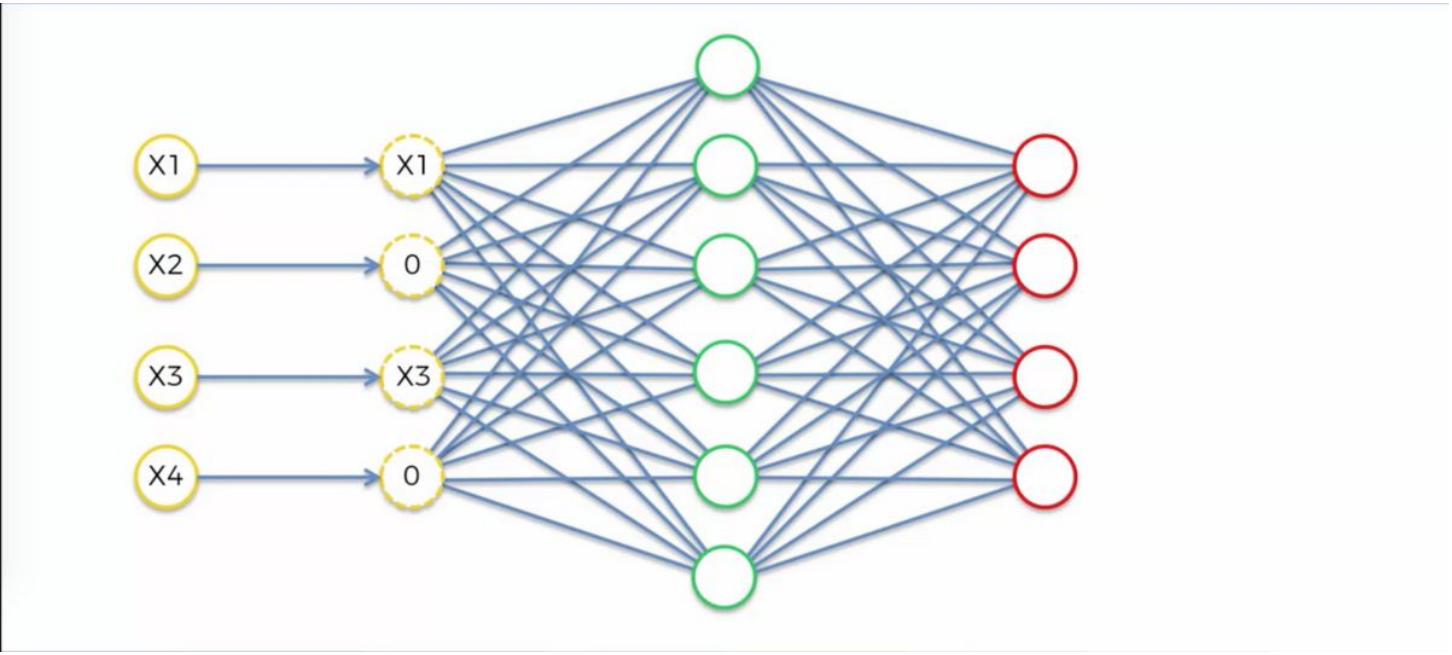


Advantages : The best in some specific contexts (when the correlated part of modalities corresponds to your task's objective).

Problems : complicated (which is why details aren't here) and mixes +/- well with neural networks. There are extensions like Deep CCA but they currently don't have a good Pytorch binding.

Bonus : denoising autoencoders

Denoising autoencoder : Reconstruct corrupted data



Deep Learning A-Z™: Hands-On Artificial Neural Networks)

Note that the hidden representation has larger dimension than the input.

This idea can be used to defend against **adversarial attacks !**