

# Part 2. Deep Learning Methods for Recommendation

Olivier Koch - Amine Benhaloum - Zofia Trstanova - Ugo Taniellian

CAIL

*June 26, 2019*

# Contents

# Structure of the talk

- Introduction to Deep Recommendations
- Complex input Recommendation with Convolutions
  - 2D CNNs for Image-based Reco
  - 1D CNNs for Text, Music and Sequence-based Reco
- User modeling with RNNs for Recommendation

# Introduction to Deep Learning for Recommendation

# What is Deep Learning?

Deep learning is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms.

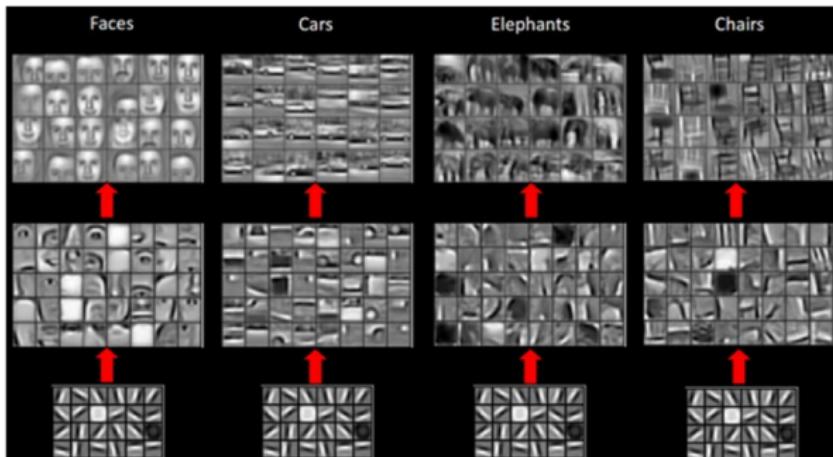
# Whats so special about Deep learning?

DLs performance and speed of adoption relies on 3 Key Concepts:

- End-to-end ML using Representation Learning
- Advances in practical Non-linear Optimization: SGD, Backpropagation and AutoDiff
- Flexibility: Differentiable Programming

# What is Representation Learning?

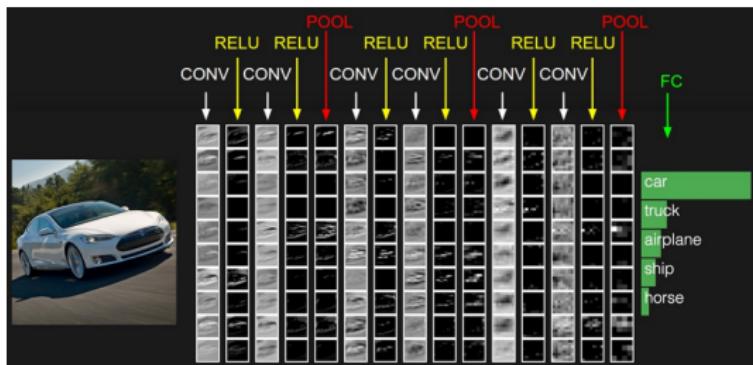
Representation Learning = automatic feature engineering



Extracted image features Image Source: <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

# What is Representation Learning?

ML Systems that do not require any human intervention on the collected input in order to learn and predict: end-to-end learning systems



CNN Architecture Image Source: <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

# What is Representation Learning?

Advantages:

- This makes manual feature engineering obsolete and saves the large amounts of time previously needed in order to find the right features (80% data preprocessing, 20% model tuning)

Furthermore, it allows a machine to learn in the same time:

- the specific task (using the features) and
- the best features for the task,
- The end-to-end automation can dramatically increase the final performance of the model

# Optimization: SGD, BackProp and AutoDiff

## SGD: Stochastic Gradient Descent

Most of the reported results of Deep Learning are due to the  
*Unreasonable Effectiveness of SGD*

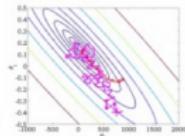
### Stochastic gradient descent

$$\text{Minimize } F(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x})$$

Initialize  $\mathbf{x}_0$

For  $j = 1, 2, \dots$

draw index  $i = i_j$  at random  
 $\mathbf{x}^{(j+1)} = \mathbf{x}^{(j)} - \gamma \nabla f_i(\mathbf{x}^{(j)})$

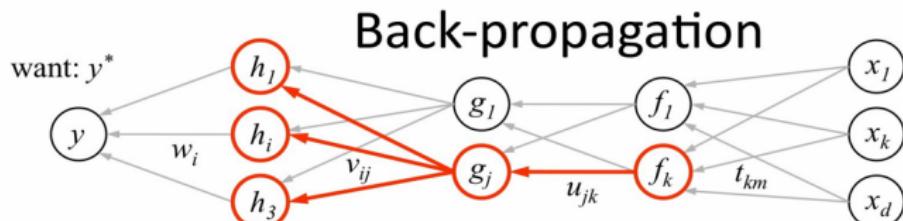


Goal: nonasymptotic bounds on  $\mathbb{E}\|\mathbf{x}^{(j)} - \mathbf{x}^*\|^2$

### The SGD algorithm

# Back-propagation

It's really just the chain rule a simple calculus trick applied in a very elegant way.



1. receive new observation  $\mathbf{x} = [x_1 \dots x_d]$  and target  $y^*$
2. **feed forward:** for each unit  $g_j$  in each layer  $1 \dots L$   
compute  $g_j$  based on units  $f_k$  from previous layer:  $g_j = \sigma(u_{j0} + \sum_k u_{jk} f_k)$
3. get prediction  $y$  and error ( $y - y^*$ )
4. **back-propagate error:** for each unit  $g_j$  in each layer  $L \dots 1$

(a) compute error on  $g_j$

$$\frac{\partial E}{\partial g_j} = \sum_i \sigma'(h_i) v_{ij} \frac{\partial E}{\partial h_i}$$

should  $g_j$  be higher or lower?  
how  $h_i$  will change as  $g_j$  changes  
was  $h_i$  too high or too low?

(b) for each  $u_{jk}$  that affects  $g_j$

(i) compute error on  $u_{jk}$       (ii) update the weight

$$\frac{\partial E}{\partial u_{jk}} = \frac{\partial E}{\partial g_j} \sigma'(g_j) f_k$$

do we want  $g_j$  to be higher/lower  
how  $g_j$  will change if  $u_{jk}$  is higher/lower

$$u_{jk} \leftarrow u_{jk} - \eta \frac{\partial E}{\partial u_{jk}}$$

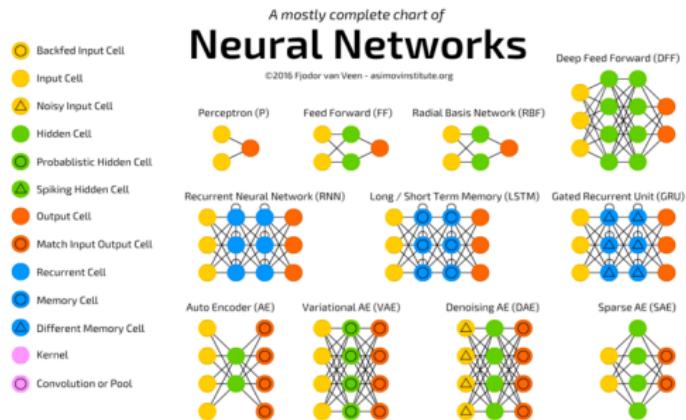
# AutoDiff: Automatic Differentiation

AutoDiff: Set of techniques to numerically evaluate the derivative of a function specified by a computer program.

3 Steps:

- Decompose original Deep Neural Net into intrinsic functions (addition, subtraction, exp, sin, cos, tanh)
- Differentiate the intrinsic functions
- Multiply together according to the chain rule

# Flexibility: Differentiable Programming



## DNN architecture types

Image Source: <http://www.asimovinstitute.org/neural-network-zoo/>

## Recap: 3 Key Concepts of DL

- Representation Learning capabilities
- The relative generality of the optimization solution: SGD, Backprop & AutoDiff
- Differentiable Programming framework

Because of the flexibility and the composability of neural modules, coupled with the possibility of training end-to-end ML systems, Deep Learning sees an accelerated adoption curve in many industries.

Q: What are we doing about it?

# What can Deep Learning do for Recommendation?

In Recommendation, we can leverage Deep Learning in order to:

- Model the complex content information on the items (CNNs)
- Model the complex user interaction with the items (RNNs, TCNs)

# Convolutional Neural Networks for Recommendation

## Recommendation with Complex Content

In general, the items to be recommended have some meta-data associated to them.

These can be information such as: category, price, product image, textual description or in the case of media items, they are themselves pieces of content (songs, videos).

Some of this content has a complex structure, such as in the case of:

- Images
- Sounds
- Text
- Sequences (time)
- Combinations of the above

## Recommendation with Complex Content

We are interested in ways to deal with complex content in a way that makes our recommendation more performant.

Of course, as in most content-based recommendation work, the main attraction of using content on top of collaborative filtering signal is to improve the cold start performance (the recommendation of new items).

# Recommendation with Complex Content

CNNs are useful for a variety of content understanding tasks

## **CNNs: Convolutional Neural Networks**

But first, we need to:

- Learn what are Convolutional Neural Networks and how do they work
- Study the existing deep architectures that are using CNNs for content-based recommendation

Quick intro to CNNs with an image  
processing example

# What are Convolutional Neural Networks?

CNNs are using stacked convolutional modules paired with various output layers (e.g. softmax).

A convolutional module generally comprises of three main steps:

- Convolution
- Non-linearity
- Pooling

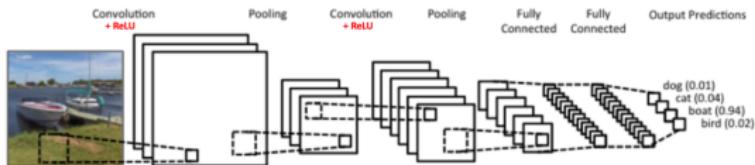


Image ConvNet Basic Architecture Image Source:  
<https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

# 1. The Convolution Step

**Intuition:** Computing activations for a fully connected 1-layer neural network = Product between the input vector and the weight matrix describing the layer.

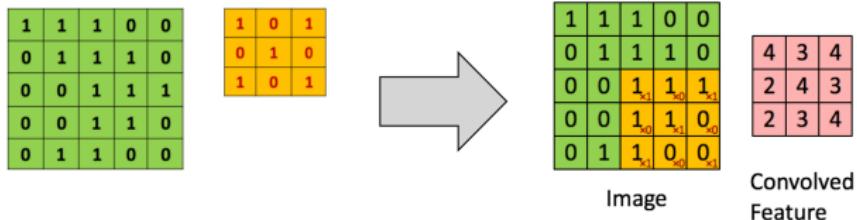
# 1. The Convolution Step

- What if we know that not all inputs are connected to the next layer?
- What if we know (like in the case of images) that there is a local structure in place (the relationship between neighboring pixels in 2D for images).
- Could we use this information when designing our network?
- Answer: Yes! And that it is exactly what a convolution is!
- An image convolution is a network that only takes the neighboring pixels into account when computing its activation.
- Because it takes a 2D neighborhood region, we will denote it as a 2D convolution.

# 1. The Convolution Step

**Image Convolution:** Local dot product of an input matrix with a smaller matrix that scans/moves along the input matrix

**Example:** See below the convolution of a 5x5 image (matrix of its binary pixels) with a smaller 3x3 matrix:



Convolution Example - Image Source: <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

# 1. The Convolution Step

## Vocabulary:

- The 33 matrix: filter, kernel, feature detector (since the matrices act as feature detectors from the original input image)
- The output of the operation: convolved feature, activation map, feature map

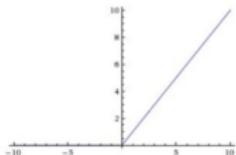
## 2. Non-linearity

Apply a non-linear function, usually **Rectified Linear Unit (ReLU)**:

$$y = \max(0, x)$$

It is a pixel-wise transformation that replaces all negative pixel values in the feature map by zero.

Output = Max(zero, Input)



ReLU function - Image source: <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

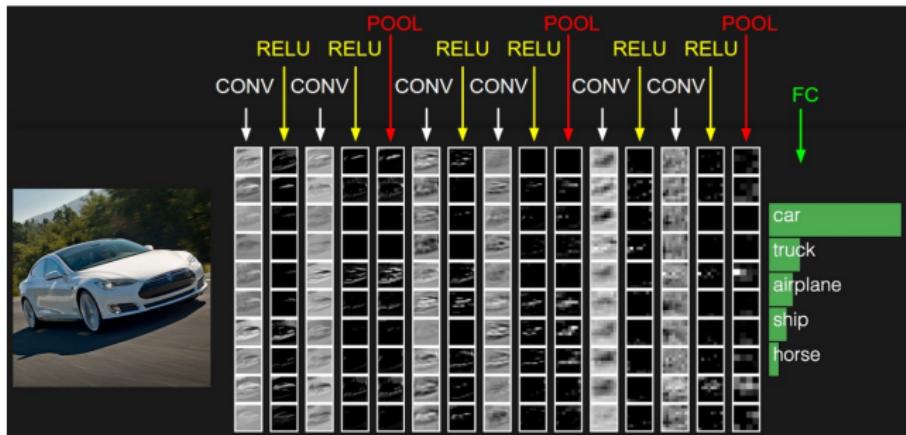
### 3. Pooling

- In the Pooling step, we define a region (e.g., a  $3 \times 3$  window) and apply a function that summarizes the region (computes a statistic).
- In Max Pooling, we take the largest element from the rectified feature map within that window.
- Other pooling functions: Min, Average, Sum etc.

The function of pooling is to shrink the size of the input representation which reduces the number of parameters and associated computations and makes the network more invariant/robust to small transformations.

# Stacking

ConvNet: Stacking the convolutional modules + output layer



Full ConvNet Architecture – Image source: <http://cs231n.github.io/convolutional-networks/>

# 2D Convolutions for Image Recommendation: Visual Bayesian Personalized Ranking (VBPR)

# Image: Visual Bayesian Personalized Ranking (VBPR)

**Main idea:** Use visual features extracted from product images using pre-trained deep networks

Learn an additional layer on top of them that summarizes the visual impact of the item description in order to better explain the observed personalized purchase outcomes.

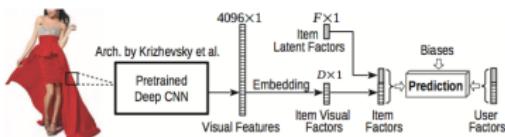


Figure 1: Diagram of our preference predictor. Rating dimensions consist of visual factors and latent (non-visual) factors. Inner products between users and item factors model the compatibility between users and items.

## VBPR Architecture

He, Ruining, and Julian McAuley. "VBPR: Visual Bayesian Personalized Ranking from Implicit Feedback." AAAI. 2016.

## Image: Visual Bayesian Personalized Ranking (VBPR)

**Task:** Learn a ranking for each user over items from logged implicit feedback (e.g. purchase histories).

**Model:** The preference model is parametrized in terms of:

- Global bias term
- Bias terms for user  $u$  and item  $i$
- Latent vectors for user  $u$  and item  $i$
- Visual vectors for  $u$  and  $i$
- The visual vector for  $i$  is a compressed representation of the ConvNet output (ConvNet used is AlexNet)
- Visual bias vector

$$\hat{x}_{u,i} = \alpha + \beta_u + \beta_i + \gamma_u^T \gamma_i + \theta_u^T (Ef_i) + \beta'^T f_i$$

## Image: Visual Bayesian Personalized Ranking (VBPR)

**Loss function:** *Bayesian Personalized Ranking (BPR)* - pairwise ranking loss

Each training example is a triple of the form  $(u, i, j)$ , Where:

- $(u, i)$  - positive pair
- $(u, j)$  - unobserved pair

$$\sum_{(u,i,j) \in D_S} \ln \sigma(\hat{x}_{uij}) - \lambda_\theta \|\Theta\|^2$$

where:

$$\hat{x}_{uij} = \hat{x}_{u,i} - \hat{x}_{u,j}$$

# Image: Visual Bayesian Personalized Ranking (VBPR)

**Results:** VBPR improves on BPR-MF by over 12% for all items recommendations, and by more than 28% for cold start items.

*Anecdote:* visual features show greater benefits on clothing than cellphone datasets.



Visualizing the geometry of the product image embeddings

# 1D Convolutions for Text

# 1D Text Convolutions

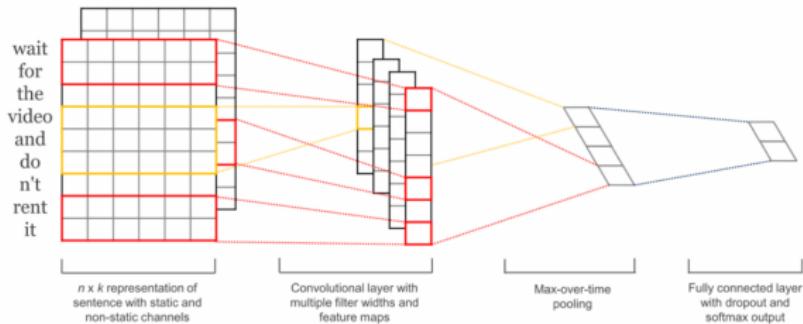
## 1D Convolutions

The input of the convolution is still 2D dimensional:

$$nb\_words \times word\_embedding\_size$$

In vision, the filters slide over local areas of an image, but in NLP the filters slide over words (represented as rows of the matrix)

Input is a 2D matrix, but the geometry of the neighborhood is 1D).

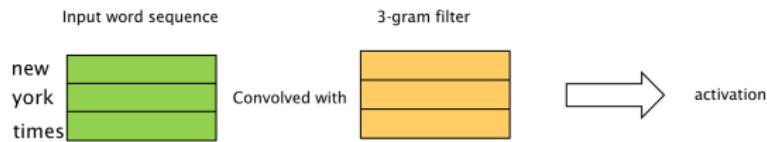


The Anatomy of Text Convolutions - Image source:

<http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/>

# 1D Text Convolutions

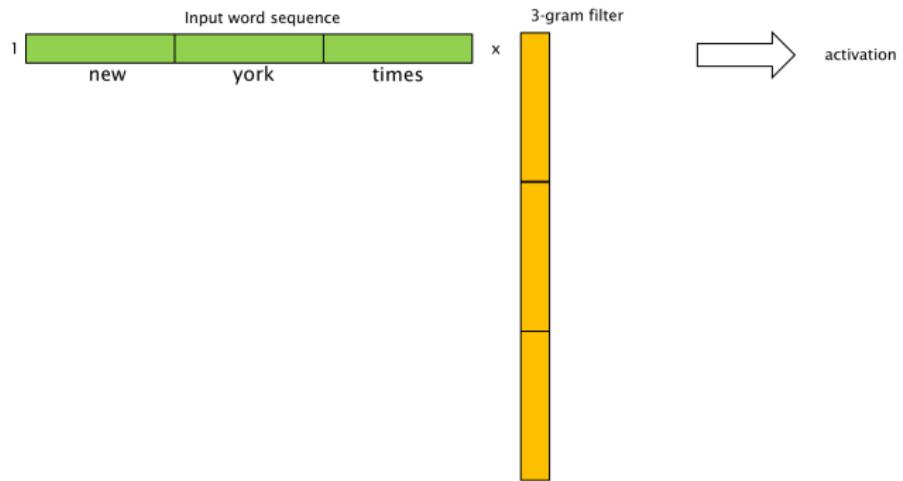
## 1D Convolutions Classical way of representing convolutions



Classical representation of 1D Convolution

# 1D Text Convolutions

1D Convolutions Alternative way that makes it more visible the 1Dimensional aspect of the operation



Alternative representation of 1D Convolution

## 1D Convolutions

Can be used for other types of sequence of set like objects (shopping baskets for instance as we will see in the practical session)

# 1D Convolutions for Music: Deep content-based music recommendation

# 1D Music Convolutions

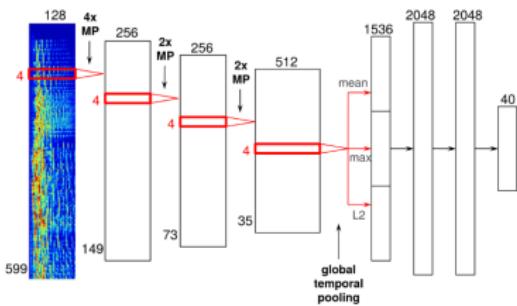
## Main idea:

- Predict song preferences from audio signals
- The model is trained to map audio signal to pre-existing latent representations of songs that were obtained from a collaborative filtering model.

**Task:** The CNN is trained to minimize the mean squared error (MSE) between the audio-based predictions and the CF-based latent factor vectors.

# 1D Music Convolutions

**Model:** Paper: 2 convolutional + 2 fully connected layers. Shown below  
(blogpost on the paper): 4 convolutional layers and 3 fully connected  
layers

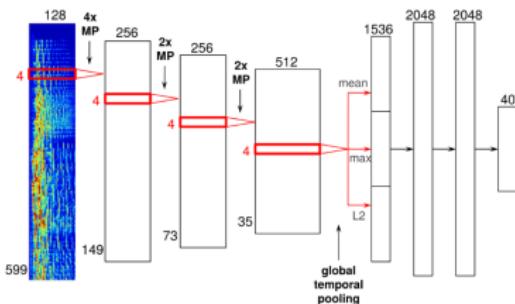


Music ConvNet Architecture – Image source: <http://benanne.github.io/2014/08/05/spotify-cnns.html>

# 1D Music Convolutions

## Model:

- Input: mel spectrograms of 3 second audio fragments
- 1D convolutional layers: shown as red rectangles
- 1D: the convolution happens only in the time dimension, not in the frequency dimension.
- The activation function: ReLU  $\max(0, x)$



Music ConvNet Architecture

# 1D Music Convolutions

## Model - pooling operators:

**Max-pooling (MP):** downsample the current input in time.

**Global temporal pooling layer:** pool across the entire time axis, effectively computing statistics of the learned features across time.

3 different global pooling functions:

- the mean
- the maximum
- the L2-norm

# 1D Music Convolutions

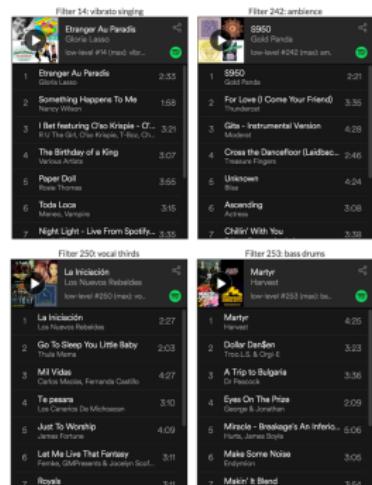
## Results:

What are the filters learning?

- Playlist for low-level features (filters from the first convolutional layer)
- Each playlist is generated by finding songs that maximally activate a certain filter.

## Example:

- Filter 14 - vibrato singing.
- Filter 242 - ringing ambience.
- Filter 250 - vocal thirds
- Filter 253 - bass drum sounds.



Low-level filters-based Reco - Image source:  
<http://benanne.github.io/2014/08/05/spotify-cnns.html>

# 1D Music Convolutions

**Results:** What are the filters learning?

- Playlist for high-level features
- Filters from the layer previous to the output
- The filters seem to pick certain subgenres.

The image shows four separate Spotify playlists, each with a unique filter name and genre. The first two filters are labeled 'high-level' while the last two are labeled '1024i'. Each playlist includes a play button, album art, song title, artist, and duration.

Filter	Genre	Song 1	Song 2	Song 3	Song 4	Song 5	Song 6	Song 7
Filter 26: gospel	gospel	God Great God - Kurt Carr	Glory and Honor - J.J. Hairston & Youthful Praise	Right Time Right Place - Kurt Carr & The Kurt Carr Singers	You - J.J. Hairston & Youthful Praise	Second Chance - Hezekiah Walker & The Love Fellow...	We Made It - (Radio Edit) - Hezekiah Walker & The Love Fellow...	Now (feat. James Fortune) - Show Luo
Filter 37: Chinese pop	Chinese pop	喜相逢 - Nicholas Tse	美丽的县城 - Justin Lo	不说再见 - G.E.M.	愛情信標 - Angelababy	兒歌 - Fish Leong	幸福不減 - Show Luo	
Filter 49: chiptune	chiptune	Last Hope (Bonus Track) - Big Giant Circles	Bed Intruder Chiptune Cover - Röyksopp	Super Boy of Little Powers - Chiptop	Captain Planet - Super Power Club	Catch Twenty Two - Chiptop	Razor Comeback Intro feat. Zedd - Dubmood	Hokkaido - Uru
Filter 1024i: deep house	deep house	Sonnenblut am Platz der P... - Dosenstutzen, Constantin...	Chewy Mobil - Super Fly, Andree	30 Northeast - Julian Jewell R... - John Digweed, Nick Muir	Something Soul - Beatnervine, David Aich	Monday 16th - Langenberg	It's Not Enough - Dusky, Joris	Never Know Me - Original Mix - 6:44

High-level filters-based Reco - Image source:  
<http://benanne.github.io/2014/08/05/spotify-cnns.html>

# 1D Music Convolutions

## Results: Seed Song similarity playlists

The Notorious B.I.G. - Juicy (hip hop)

Rank	Song	Artist	Length
1	Juicy	The Notorious B.I.G.	5:02
2	You Got Me	The Roots, Eve, Jill Scott	4:56
3	A Long Walk - The Jazzy Jeff ...	Jill Scott	3:57
4	The Only One You Need	Donald Jones	3:46
5	Sock It 2 Me (feat. Da Brat)	Missy Elliott, Da Brat	4:17
6	Doo Wop (That Thing)	Ms. Lauryn Hill	5:20
7	Sprinkle Me		4:10

John Coltrane - My Favorite Things (jazz)

Rank	Song	Artist	Length
1	... (partially visible)		17:51
2	Da-Me Um Beijo	Eis Regina	3:25
3	Crawfish	Elvis Presley	1:51
4	Old Man Blues	Sidney Bechet and his New Orleans ...	2:49
5	Tu che m'hai preso il cuor	Fausto Papetti	2:01
6	Falling in Love with Love	Toots Thielemans	2:33
7	Rhumbboogie		2:37

Song-based Reco - Image source: <http://benanne.github.io/2014/08/05/spotify-cnns.html>

# 1D Convolutions for Sequences: TemporalCNs (TCNs)

## Sequences/Time: TemporalCNNs (TCNs)<sup>1</sup>

**Main ideas:** Sequence modeling does not have to be based on RNNs.

In fact, the authors show that for a variety of task temporal CNNs outperform RNNs and present other advantages.

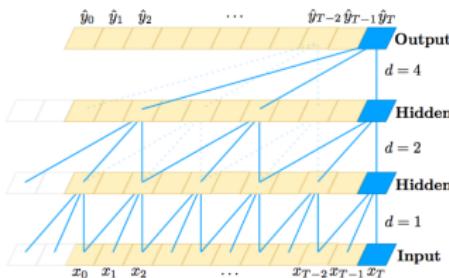
---

<sup>1</sup>Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling." arXiv preprint arXiv:1803.01271 (2018).

# Sequences/Time: TemporalCNNs (TCNs)

## Model:

- **Causal Convolutions:** Convolutions where an output at time  $t$  is convolved only with elements from time  $t$  or older.
- **Dilated Convolutions:** Causal convolutions that skip past states with dilation step  $d$ . By stacking dilated convolutions with exponentially increasing  $d$ , we can compress a very long history:



TCN Architecture

# Sequences/Time: TemporalCNNs (TCNs)

TCNs vs RNNs:

## **Advantages of TCNs:**

- Parallelism.
- Flexible (longer) receptive field size.
- Stable gradients. TCNs avoid the problem of exploding/vanishing gradients, which is a major issue for RNNs.
- Low memory requirement for training. TCNs use in practice an order of magnitude less memory than RNNs.

## **Disadvantages of TCNs:**

- Data storage during evaluation. TCNs have to keep in memory the raw sequence, RNNs only the last hidden state
- Potential parameter change for a transfer of domain.

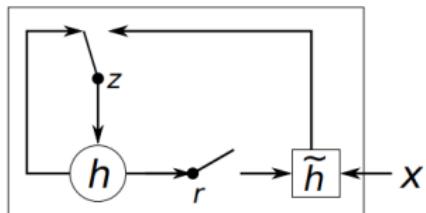
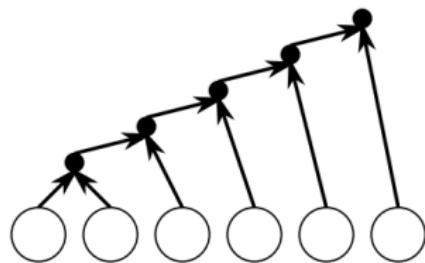
## Sequences/Time: TemporalCNNs (TCNs)

**Results:** TCNs outperform RNNs in 10/11 of the benchmark tasks  
(seq2seq tasks such as translation are outside of the scope)

# User Modeling with RNNs for Recommendation

# Recurrent Neural Networks

RNNs have a feedback loop in the hidden layer that lets them maintain information over time.



# Recurrent Neural Networks

Given a sequence  $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$ , the hidden state update is given by:

$$\mathbf{h}_t = g(W\mathbf{x}_t + U\mathbf{h}_{t-1}) \quad (1)$$

# Recurrent Neural Networks

The sequence probability is given by:

$$p(x_1, \dots, x_T) = p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \cdots p(x_T | x_1, \dots, x_{T-1}) \quad (2)$$

where

$$p(x_t | x_1, x_2, \dots, x_{t-1}) = g(h_t) \quad (3)$$

## The vanishing gradient problem

Even in very simple situations, gradient descent fails for long-term input/output dependencies

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Transactions on Neural Networks*, 1994.

# The vanishing gradient problem

Two ways to solve it:

- use a better optimizer than SGD (e.g. clipped gradient)
- use a better activation function (recurrent unit = LSTM, gate recurrent unit = GRU)

## Long Short Term Memory

- Standard RNNs apply a non-linear function to the weighted sum of input signal.
- LSTMs maintain a memory cell over time and can capture long-distance dependencies
- A set of gates controls when memory enters the memory cell (and when it is forgotten)

# LSTMs in the real world

LSTMs are widely used in industry

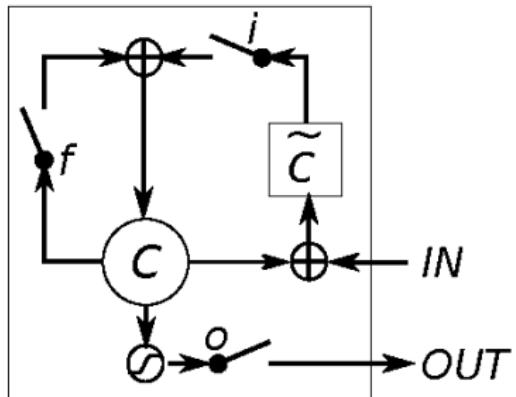
- Apple uses LSTM for the "Quicktype" function on the iPhone and for Siri
- Google uses LSTM for voice search
- “Microsoft’s speech recognition system is now as good as a human“

## Gated Recurrent Units

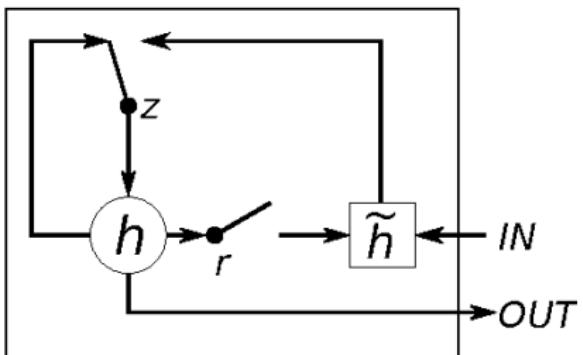
GRUs are similar to LSTMs but use a simpler architecture. They use fewer cells than LSTM and do not require a memory cell.

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, J. Chung et al, NIPS 2014 Workshop on Deep Learning

# GRU and LSTM



(a) Long Short-Term Memory



(b) Gated Recurrent Unit

# GRU4REC

- GRU based RNN trained on shopping session data, adapted to the recommendation task
- Predicting next product in a session
- Sounds natural but it was only done in 2015 !

Session-based Recommendations with Recurrent Neural Networks by Massimo Quadrana, Balzs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, Domonkos Tikk (2015)

# GRU4REC

- But there is an issue: predicting a probability of each product in the catalog, at each time step ! (Softmax layer)
- Solution: don't score everything (Negative Sampling), but only try to differentiate the true product from the false ones
- How to sample ? Uniformly, by popularity ?
- Sampling is non trivial, solution proposed was to take the products from the other users of the batch as negatives (pretty neat)

# GRU4REC

What about the loss ? Several choice were tried ( $r_i$  denotes predicted score)

- Cross entropy, can be seen as a listwise ranking loss

$$XE = -\log(s_i) \quad s_i = \frac{\exp r_i}{\sum_{j=1..K} \exp r_j}$$

- BPR loss, pairwise ranking loss

$$BPR = -\frac{1}{K} \sum_{j=1}^K \log(\sigma(r_i - r_j))$$

- TOP-1 loss, kind of heuristic

$$TOP1 = \frac{1}{K} \sum_{j=1}^K \sigma(r_i - r_j) + \sigma(r_i^2)$$

- But all of them perform similarly, TOP1 slightly better

# Recurrent Neural Networks with Top-k Gains for Session-based Recommendations

Follow up paper to GRU4REC, with an interesting observation

- When number of negatives is moderate: good results
- When number of negatives is big: bad results

Recurrent Neural Networks with Top-k Gains for Session-based Recommendations  
Balzs Hidasi, Alexandros Karatzoglou (2017)

# Recurrent Neural Networks with Top-k Gains for Session-based Recommendations

Why is that, let's take the derivative of BPR loss wrt to the positive item's score

- $\frac{\partial L}{\partial r_i} = \frac{1}{K} \sum_{j=1}^K (1 - \sigma(r_i - r_j))$
- For irrelevant negative sample  $(1 - \sigma(r_i - r_j)) \approx 0$ , so gradient vanishes the better the model becomes

# Recurrent Neural Networks with Top-k Gains for Session-based Recommendations

One solution, only consider the negative sample with max score, and try to make positive product score bigger than that

- $P(r_i > r_{MAX}) = \sum_{j=1}^K P(r_i > r_j | r_j = r_{MAX})P(r_j = r_{MAX})$
- $P(r_i > r_j | r_j = r_{MAX}) = \sigma(r_i - r_j)$
- $P(r_j = r_{MAX}) = \text{softmax}(r_k)_i$

The loss then becomes

$$L = -\log\left(\sum_{j=1}^K s_i * \sigma(r_i - r_j)\right)$$

# Recurrent Neural Networks with Top-k Gains for Session-based Recommendations

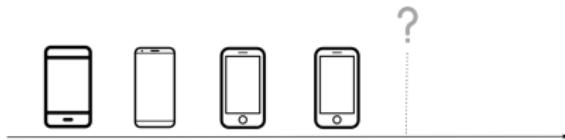
Let's check the derivative now of the loss wrt to the positive product score  $i$

$$\frac{\partial L}{\partial r_i} = \frac{\sum_{j=1}^K s_j \sigma(r_i - r_j)(1 - \sigma(r_i - r_j))}{\sum_{j=1}^K s_j \sigma(r_i - r_j)}$$

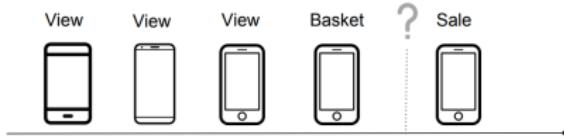
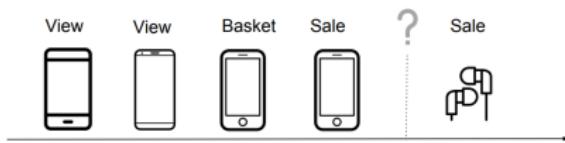
The contribution of each negative sample is weighted by its "relevance"

# Contextual data is important

Next event prediction with no event type information

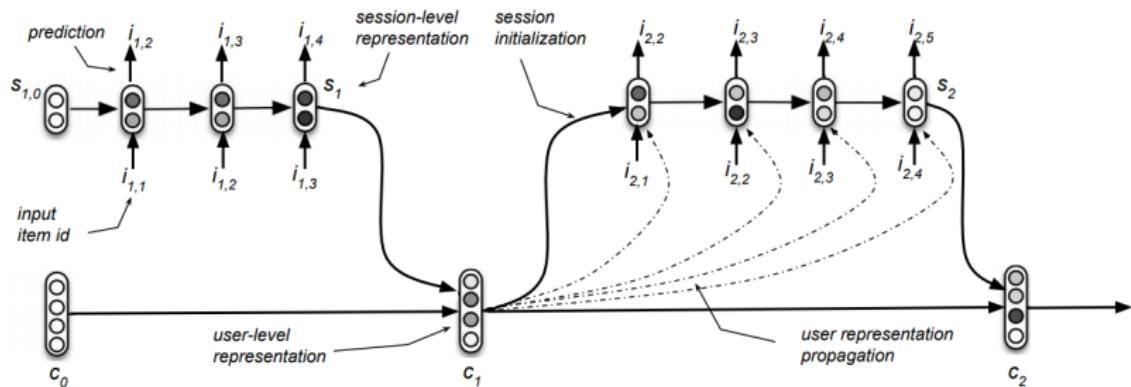


Next event prediction with event type information



# Hierarchical RNNs

Combining a user-level GRU and a session-level GRU



Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks by Massimo Quadrana, Alexandros Karatzoglou, Balzs Hidasi, Paolo Cremonesi, RecSys 2017

# Conclusions

- RNNs model temporal dependencies
- They are making their way into industrial applications
- They are harder to train than static neural nets

Thank You!