Deep Learning for Recommender Systems

Balázs Hidasi

Head of Research @ Gravity R&D

balazs.hidasi@gravityrd.com

RecSys Summer School, 21-25 August, 2017, Bozen-Bolzano

GRAVITY Research & Development

What is Deep Learning?

- A class of machine learning algorithms
  - that use a cascade of multiple non-linear processing layers
  - and complex model structures
  - to learn different representations of the data in each layer
  - where higher level features are derived from lower level features
  - to form a hierarchical representation

What is Deep Learning?

- The second resurgence of neural network research
- A useful toolset for
  - pattern recognition (in various data)
  - representation learning
- A set of techniques that achieve previously unseen results on complex tasks
  - Computer vision
  - Natural language processing
  - Reinforcement learning
  - Speech recognition
  - Etc.
- A key component of recent intelligent technologies
  - Personal assistants
  - Machine translation
  - Chatbot technology
  - Self driving cars
  - Etc.
- A new trendy name for neural networks

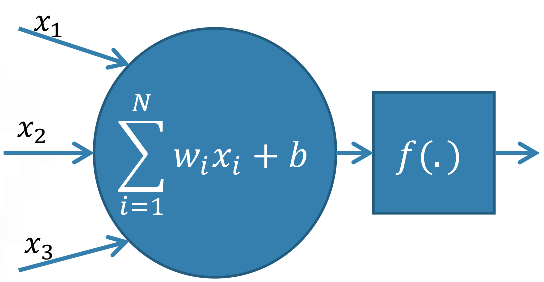
What is Deep learning NOT?

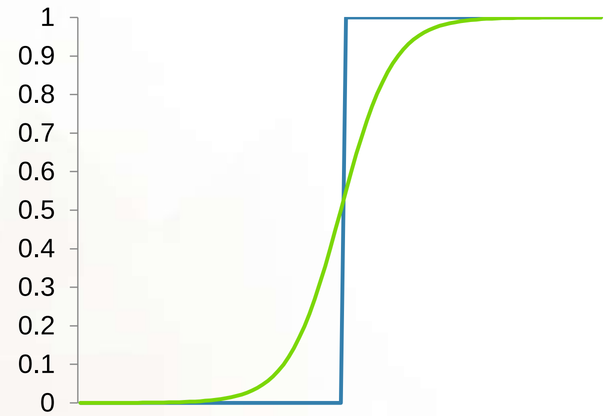
- Deep learning is NOT
  - AI (especially not general/strong AI)
    - AI has many to it than just machine learning
    - It can be part of specialized AIs
    - Might be part of a future strong AI
  - the artifical equivalent of the human brain
    - but techniques in DL are inspired by neuroscience
  - the best tool for every machine learning task
    - requires lots of data to work well
    - computationally expensive
    - „no guarantees”: theorethical results are few and far between
    - (mostly) a black box approach
    - lot of pitfalls

GRAVITY Research & Development

Neural Networks - Neuron

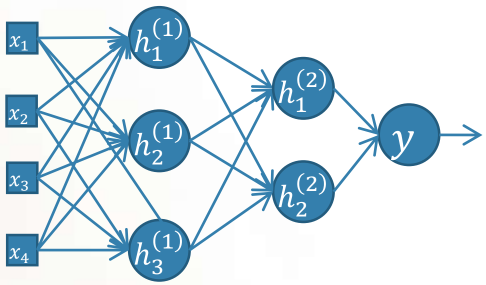
- Rough abstraction of the human neuron
  - Receives inputs (signals)
  - Sum weighted inputs is big enough → signal
    - Non-continuous step function is approximated by sigmoid
      - $\sigma(x) = \frac{1}{1+e^{-x}}$
      - $\sigma'(x) = \big(1 - \sigma(x)\big)\sigma(x)$
  - Amplifiers and inhibitors
  - Basic pattern recognition
- The combination of a linear model and an activation function
  - $y = f\left(\sum_i w_i x_i + b\right)$





Neural Networks

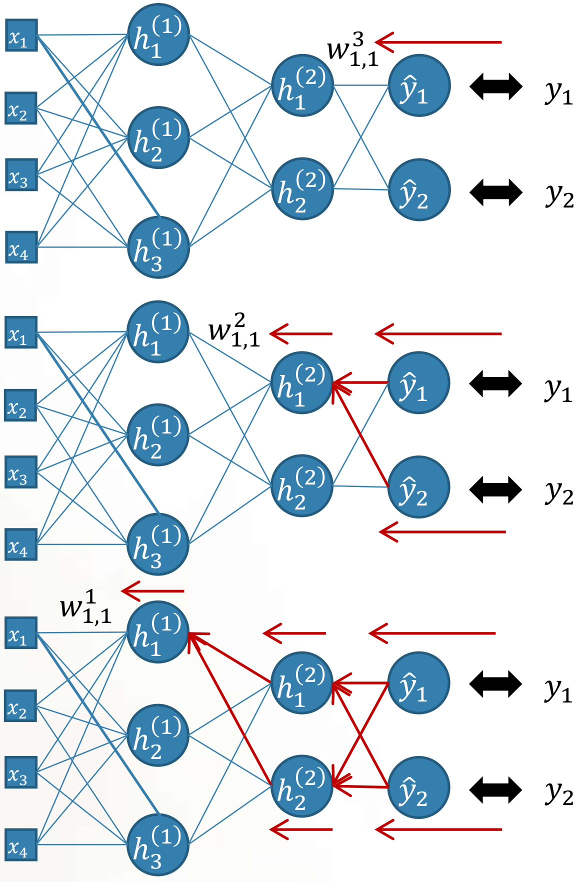
- Artificial neurons connected to each other
  - Outputs of certain neurons connected to the input of neurons
- Feedforward neural networks
  - Neurons organized in layers
    - The input of the k-th layer is the output of the (k-1)-th layer
    - Input layer: the values are set (based on data)
    - Output layer: the output is not the input of any other layer
    - Hidden layer(s): the layers inbetween
  - Forward propagation
    - $h_i^0 = x_i$
    - …
    - $s_j^k = \sum_j w_{i,j}^k h_i^{k-1} + b_j$
    - $h_j^k = f\left(s_j^k\right)$
    - ….
    - $y_i = f\left(s_i^{n+1}\right)$
    - $s^k = W^k h^{k-1} + b$
    - $h^k = f\left(s^k\right)$



GRAVITY Research & Development

Training Neural Networks - Backpropagation

- Training: modify weights to get the expected output
  - Training set: input-(expected) output pairs
  - Many ways to do this
  - Most common: gradient descent
    - Define loss between output and expected output
      - Loss (L): single scalar
      - Multiple output: individual losses (e_i) are summed
    - Compute the gradient of this loss wrt. the weights
    - Modify the weights in the (opposite) direction of the gradient
- For the hidden-to-output weights (last layer):
  - $\frac{\partial L}{\partial w_{j,i}^{n+1}} = \frac{\partial e_i}{\partial \hat{y}_i} \cdot \frac{\partial y_i}{\partial s_i^{n+1}} \cdot \frac{\partial s_i^{n+1}}{\partial w_{j,i}^{n+1}} = \frac{\partial e_i}{\partial \hat{y}_i} f'\left(s_i^{n+1}\right) h_j^n$
- For the second to last layser:
  - $\frac{\partial L}{\partial w_{k,j}^{n}} = \sum_i \frac{\partial e_i}{\partial \hat{y}_i} \cdot \frac{\partial y_i}{\partial s_i^{n+1}} \cdot \frac{\partial s_i^{n+1}}{\partial h_j^n} \cdot \frac{\partial h_j^n}{\partial s_j^n} \cdot \frac{\partial s_j^n}{\partial w_{k,j}^n} = \sum_i \frac{\partial e_i}{\partial \hat{y}_i} f'\left(s_i^{n+1}\right) w_{j,i}^{n+1} f'\left(s_j^n\right) h_k^{n-1}$
- Backpropagation of the error from layer k to (k-1)
  - $\frac{\partial L}{\partial w_{l,j}^{k-1}} = \left(\sum_i d_i^k\, w_{j,i}^k\right) f'\left(s_j^{k-1}\right) h_l^{k-2}$
  - $d_j^k = \begin{cases} \frac{\partial e_i}{\partial \hat{y}_i} & \text{if } k = n+1 \\ \sum_i d_i^{k+1} w_{j,i}^{k+1} f'\left(s_j^k\right) & \text{otherwise} \end{cases}$
  - $\frac{\partial L}{\partial W^{k-1}} = h^{k-2}\left(\left(d^k\right)^T W^k \circ f'\left(s_j^{k-1}\right)^T\right)$
  - $\left(d^k\right)^T = \left(d^{k+1}\right)^T W^{k+1} \circ f'\left(s_j^k\right)^T$



GRAVITY Research & Development

Why go deep?

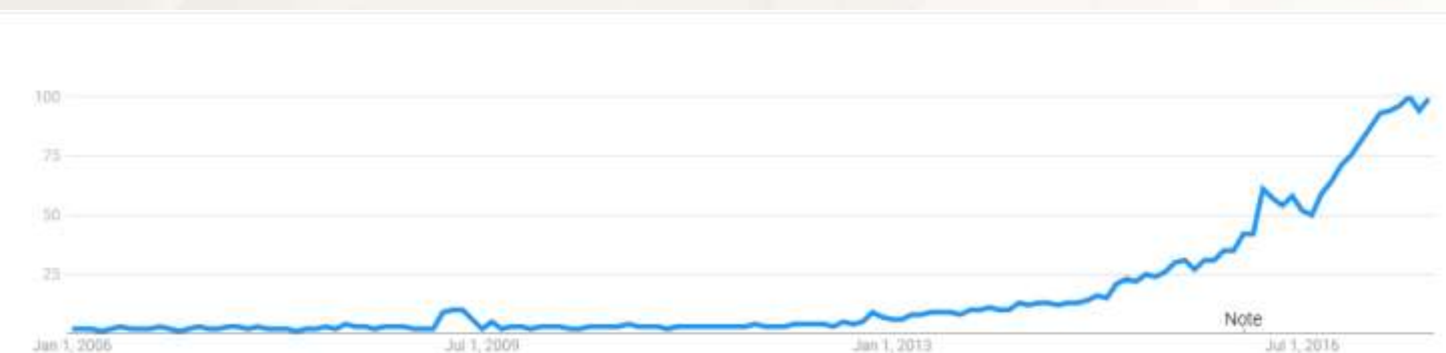
- Feedforward neural networks are universal approximators
  - Can approximate any function with arbitarily low error if they are **big enough**
- What is big enough?
  - Number of layers / neurons
  - Theoretical „big enough” conditions massively overshoot
- Go deep, not wide
  - For certain functions it is shown
  - Exists a k number
  - The number of neurons required for approximating the function is polynomial (in the input) if the network has at least k hidden layers (i.e. deep enough)
  - Otherwise the number of required units is exponential in the input

Why was it hard to train neural networks?

- Vanishing gradients
  - $\sigma'(x) = \big(1 - \sigma(x)\big)\sigma(x)$
    - x is too small or too big, the gradient becomes near zero (no update) → saturation
      - It is possible that large parts of the network stop changing
    - The maximum is 0.25 (at $x = 0$)
    - After several layers the gradient vanishes (update negligible)
- Saturation
  - Absolute value of weighted inputs is large
  - Output 1/0, gradient close to 0 (no updates)
    - Neuron doesn't learn
  - Solutions (lot of effort on each task)
    - Initialization
    - Limited activations
    - Sparse activations
- Overfitting
  - High model capacity, prone to overfitting
  - Black box, overfitting is not apparent
  - L1/L2 regularization helps, but doesn't solve the problem
  - Early stopping
- Convergence issues
  - SGD often gets stuck → momentum methods
  - Sensitivity to learning rate parameter

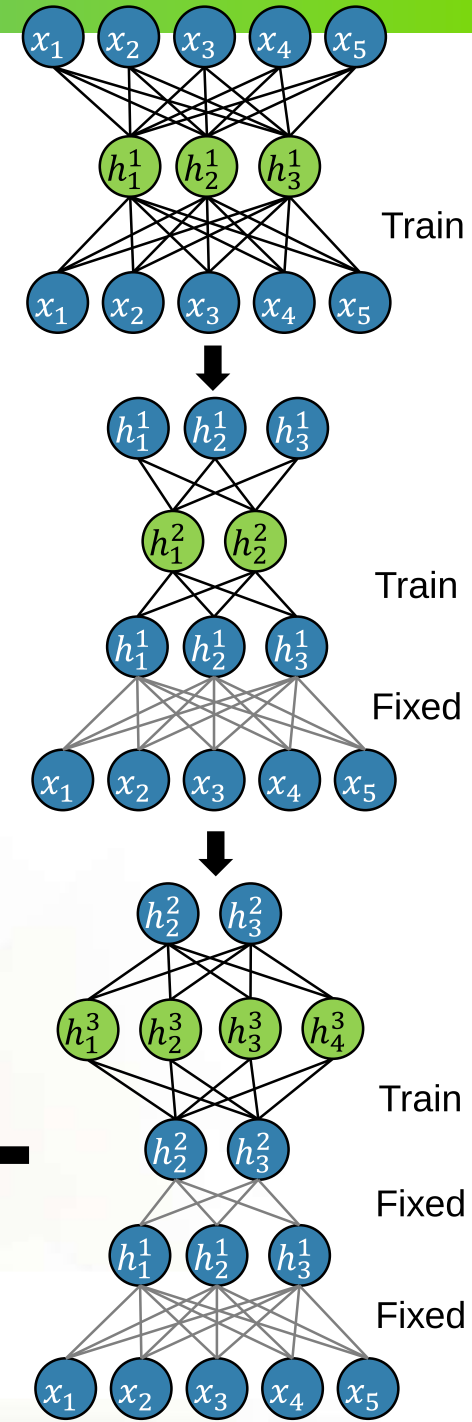
Neural Winters

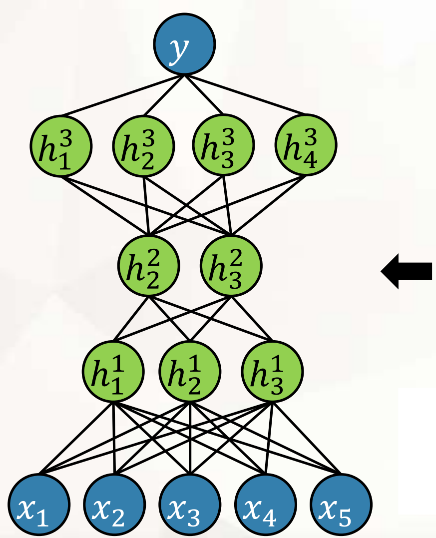
- Reasons:
  - Inflated expectations
  - Underdelivering
  - Hard to train the networks
- Results in disappointment
  - People abandoning the field
  - Lower funding
- First neural winter in the 1970s, second in the 1990s
  - Gives way to other methods
- Deep learning is not new
  - First deep models were proposed in the late 1960s
- The area was revived in the mid-2000s by layerwise training
- Deep learning boom has started around 2012-2013



Intermission – Layerwise training

- [Hinton et. al, 2006]
- To avoid saturation of the activation functions
- Layerwise training:
  - 1. Train a network with a single hidden layer, where the desired output is the same as the input
    - Unsupervised learning (autoassociative neural network)
    - The hidden layer learns a latent representation of the input
  - 2. Cut the output layer
  - 3. Train a new network with a single layer, using the hidden layer of the previous network as the input
    - Repeat from 2 fro some more layers
  - 4. For supervised learning, put a final layer on the top of this structure and optionally fine tune the weights
- What happens?
  - The weights are not initialized randomly
  - Rather they are set to produce latent representations in the hidden layer
  - Vanishing gradient is still in the lower layers
  - No problem, the weights are set to sensible values
- Deep Belief Networks (DBN), Deep Boltzmann Machines (DBM)
- Was replaced by end-to-end training & non-saturating activations





Why now? - Compute

- Natural increase in computational power
- GP GPU technology
  - NN rely on matrix and vector operations
  - Parallelization brings great speed-up
  - GPU architecture is a good fit

GRAVITY
Research & Development

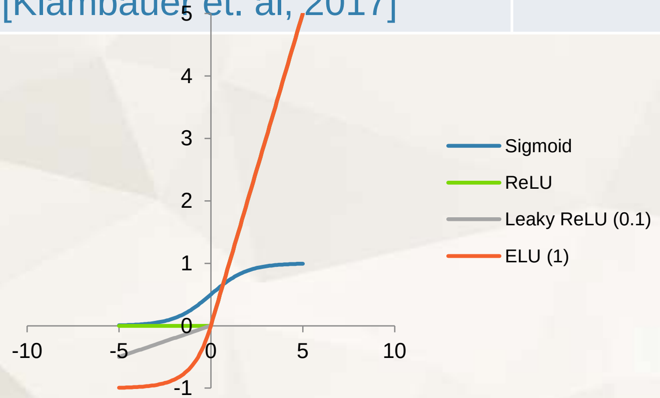
Why now? - Data

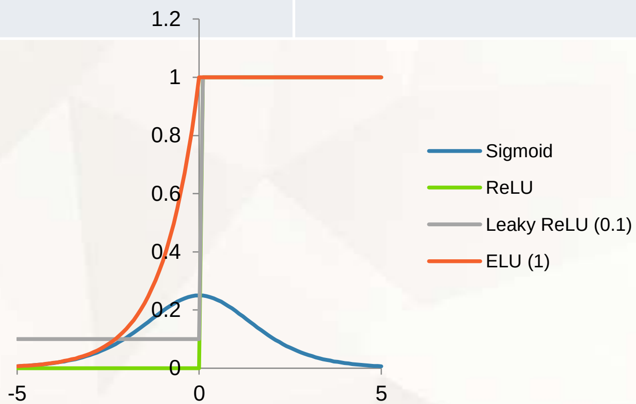
- Complex models are more efficient when trained on lots of data
- The amount of data increased quickly
  - This includes labelled data as well

GRAVITY Research & Development

Why now? – Research breakthroughs – Non-saturating activations

Name	$f(x)$	$f'(x)$	Parameters
Rectified Linear Unit (ReLU) [Nair & Hinton, 2010]	$f(x) = \max(x, 0)$	$f'(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$	None
Leaky ReLU [Maas et. al, 2013]	$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}$	$f'(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ \alpha & \text{if } x < 0 \end{cases}$	$0 < \alpha < 1$
Exponential Linear Unit (ELU) [Clevert et. al, 2016]	$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$	$f'(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ f(x) + \alpha & \text{if } x < 0 \end{cases}$	α
Scaled Exponential Linear Unit (SELU) [Klambauer et. al, 2017]	$f(x) = \lambda \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$	$f'(x) = \lambda \begin{cases} 1 & \text{if } x \geq 0 \\ f(x) + \alpha & \text{if } x < 0 \end{cases}$	α $\lambda > 1$





GRAVITY Research & Development

Why now? – Research breakthroughs – Dropout: easy but efficient regularization

- Dropout [Srivastava et. al, 2014]:
  - During training randomly disable units
  - Scale the activation of remaining units
    - So that the average expected activation remains the same
  - E.g.: dropout=0.5
    - Disable each unit in the layer with 0.5 probability
    - Multiply the activation of non-disabled units by 2
  - No dropout during inference time
- Why dropout works?
  - A form of ensemble training
    - Multiple configurations are trained with shared weights and averaged in the end
  - Reduces the reliance of neurons on each other
    - Each neuron learns something useful
    - Redundance in pattern recognition
  - Form of regularization

GRAVITY Research & Development

Why now? – Research breakthroughs – Mini-batch training

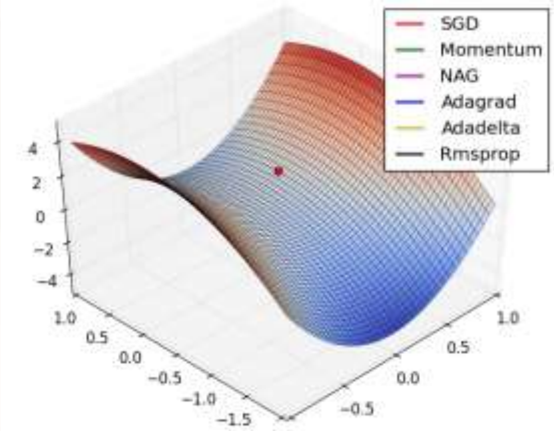
- Full-batch gradient descent
  - Compute the average gradient over the full training data
  - Pass all data points forward & backward
    - Without changing the weights
    - Save the updates
  - Compute the average update and modify the weights
  - Accurate gradients
  - Costly updates, but can be parallelized
- Stochastic gradient descent
  - Select a random data point
  - Do a forward & backwards pass
  - Update the weights
  - Repeat
  - Noisy gradient
    - Acts as regularization
  - Cheap updates, but requires more update steps
  - Overall faster conversion
- Mini-batch training
  - Select N random data points
  - Do batch training with these N data points
  - The best of both worlds

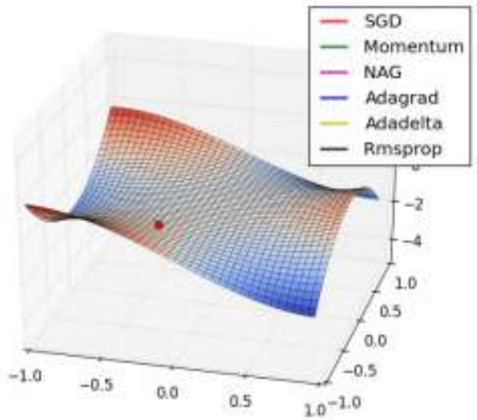
GRAVITY Research & Development

Why now? – Research breakthroughs – Adaptive learning rates

- Standard SGD gets stuck in valleys and around saddle points
  - Momentum methods
- Learning rate parameter greatly influences convergence speed
- Learning rate scheduling
  - Larger steps in the beginning
  - Smaller steps near the end
  - Various heuristics
    - E.g. multiply by $0 < \gamma < 1$ after every N updates
    - E.g. Measure error on a small validation set and decrease learning rate if there is no improvement
  - Weights are not updated with the same frequency
- Adaptive learning rates
  - Collect gradient updates on weights so far and use these to scale learning rate per weight
  - Robust training wrt initial learning rate
  - Fast convergence
  - Recent paper claims that these might be suboptimal

Method	Accumulated values	Scaling factor
Adagrad [Duchi et. al, 2011]	$G_t = G_{t-1} + (\nabla L_t)^2$	$-\frac{\eta}{\sqrt{G_t + \epsilon}}$
RMSProp [Tieleman & Hinton, 2012]	$G_t = \gamma G_{t-1} + (1-\gamma)(\nabla L_t)^2$	$-\frac{\eta}{\sqrt{G_t + \epsilon}}$
Adadelta [Zeiler, 2012]	$G_t = \gamma G_{t-1} + (1-\gamma)(\nabla L_t)^2$ $\Delta_t = \gamma \Delta_{t-1} + (1-\gamma)\left(\frac{\sqrt{\Delta_{t-1}+\epsilon}}{\sqrt{G_t+\epsilon}}\nabla L_t\right)^2$	$-\frac{\sqrt{\Delta_{t-1}+\epsilon}}{\sqrt{G_t+\epsilon}}\nabla L_t$
Adam [Kingma & Ba, 2014]	$M_t = \beta_1 M_{t-1} + (1-\beta_1)\nabla L_t$ $V_t = \beta_2 V_{t-1} + (1-\beta_2)(\nabla L_t)^2$	$-\frac{\eta \frac{M_t}{1-\beta_1^t}}{\sqrt{\frac{V_t}{1-\beta_2^t}+\epsilon}}$





Complex deep networks

- Modular view
  - Complex networks are composed from modules appropriate for certain tasks
  - E.g. Feature extraction with CNN, combined with an RNN for text representation fed to feedforward module
- Function approximation
  - The network is a trainable function in a complex system
  - E.g. DQN: the Q function is replaced with a trainable neural network
- Representation learning
  - The network learns representations of the entities
  - These representations are then used as latent features
  - E.g. Image classification with CNN + a classifier on top

Common building blocks

- Network types
  - Feedforward network (FFN, FNN)
  - Recurrent network (RNN)
    - For sequences
  - Convolutional network (CNN)
    - Exploiting locality
- Supplementary layers
  - Embedding layer (input)
  - Output layer
    - Classifier
      - Binary
      - Multiclass
    - Regressor
- Losses (common examples)
  - Binary classification: logistic loss
  - Multiclass classification: cross entropy (preceeded by a softmax layer)
  - Distribution matching: KL divergence
  - Regression: mean squared error

Common architectures

- Single network
- Multiple networks merged
- Multitask learning architectures
- Encoder-decoder
- Generative Adversarial Networks (GANs)
- And many more...

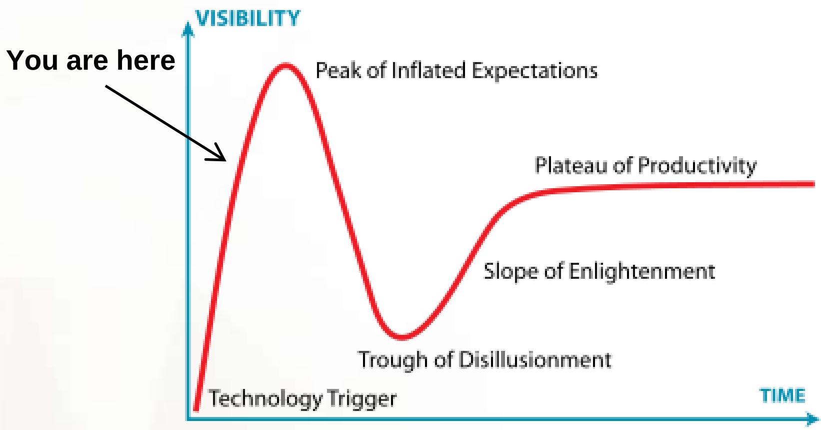
GRAVITY Research & Development

Impressive results

- Few of the many impressive results by DL fom the last year
  - Image classification accuracy exceeds human baseline
  - Superhuman performance in certain Atari games
    - Agent receives only the raw pixel input and the score
  - AlphaGo beat go world champions
  - Generative models generate realistic images
  - Large improvements in machine translation
  - Improvements in speech recognition
  - Many production services using deep learning

Don’t give in to the hype

- Deep learning is impressive but
  - deep learning is not AI
  - strong/general AI is very far away
    - instead of worrying about „sentient” AI, we should focus on the more apparent problems this technological change brings
  - deep learning is not how the human brain works
  - not all machine learning tasks require deep learning
  - deep learning requires a lot of computational power
  - the theory of deep learning is far behind of its empirical success
  - this technological change is not without potentially serious issues inflicted on society if we are not careful enough
- Deep learning is a tool
  - which is successful in certain, previously very challenging domains (speech recognition, computer vision, NLP, etc.)
  - that excels in pattern recognition



Why deep learning has potential for RecSys?

- Feature extraction directly from the content
  - Image, text, audio, etc.
  - Instead of metadata
  - For hybrid algorithms
- Heterogenous data handled easily
- Dynamic behaviour modeling with RNNs
- More accurate representation learning of users and items
  - Natural extension of CF & more
- RecSys is a complex domain
  - Deep learning worked well in other complex domains
  - Worth a try

GRAVITY Research & Development

The deep learning era of RecSys

- Brief history:
  - 2007: Deep Boltzmann Machines for rating prediction
    - Also: Asymmetric MF formulated as a neural network (NSVD1)
  - 2007-2014: calm before the storm
    - Very few, but important papers in this topic
  - 2015: first signs of a deep learning boom
    - Few seminal papers laying the groundwork for current research directions
  - 2016: steep increase
    - DLRS workshop series
    - Deep learning papers at RecSys, KDD, SIGIR, etc.
    - Distinct research directions are formed by the end of the year
  - 2017: continuation of the increase of DL in recommenders
- Current status & way forward
  - Current research directions to be continued
  - More advanced ideas from DL are yet to be tried
  - Scalability is to be kept in mind

Research directions in DL-RecSys

- As of 2017 summer, main topics:
  - Learning item embeddings
  - Deep collaborative filtering
  - Feature extraction directly from the content
  - Session-based recommendations with RNN
- And their combinations

GRAVITY
Research & Development

Best practices

- Start simple
  - Add improvements later
- Optimize code
  - GPU/CPU optimizations may differ
- Scalability is key
- Opensource code
- Experiment (also) on public datasets
- The data should be compatible with the task you want to solve
- Don't use very small datasets
- Don't work on irrelevant tasks, e.g. rating prediction

GRAVITY
Research & Development

Frameworks

- Low level
  - Torch, pyTorch - Facebook
  - Theano – University of Montreal
  - Tensorflow - Google
  - MXNet
- High level
  - Keras
  - Lasagne

GRAVITY
Research & Development

References

- [Clevert et. al, 2016] DA. Clevert, T. Unterthiner, S. Hochreiter: Fast and accurate deep network learning by exponential linear units (elus). International Conference on Learning Representations (ICLR 2016).
- [Duchi et. al, 2011] J. Duchi, E. Hazan, Y. Singer: Adaptive subgradient methods for online learning and stochastic optimization. JMLR 12, 2121–2159 (2011).
- [Hinton et. al, 2006] G. Hinton, S. Osindero, YW Teh: A fast learning algorithm for deep belief nets. *Neural computation* 18.7 (2006): 1527-1554.
- [Kingma & Ba, 2014] D. P. Kingma, J. L. Ba: Adam: A method for stochastic optimization. ArXiv preprint (2014) https://arxiv.org/abs/1412.6980.
- [Klambauer et. al, 2017] G. Klambauer, T. Unterthiner, A. Mayr, S. Hochreiter, : Self-Normalizing Neural Networks. ArXiv preprint (2017) https://arxiv.org/abs/1706.02515.
- [Maas et. al, 2013] A. L. Maas, A. Y. Hannun, A. Y. Ng: Rectifier nonlinearities improve neural network acoustic models. 30th International Conference on Machine Learning (ICML 2013).
- [Nair & Hinton, 2010] V. Nair, G. Hinton: Rectified linear units improve restricted boltzmann machines. 27th International Conference on Machine Learning (ICML 2010).
- [Srivastava et. al, 2014] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov: Dropout: a simple way to prevent neural networks from overfitting. Journal of machine learning research, 15(1), 1929-1958. 2014.
- [Tieleman & Hinton, 2012] T. Tieleman, G. Hinton: Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning. Technical report, 2012.
- [Zeiler, 2012] M. D. Zeiler: ADADELTA: an adaptive learning rate method. ArXiv preprint (2012) https://arxiv.org/abs/1212.5701.

Learning item embeddings & 2vec models

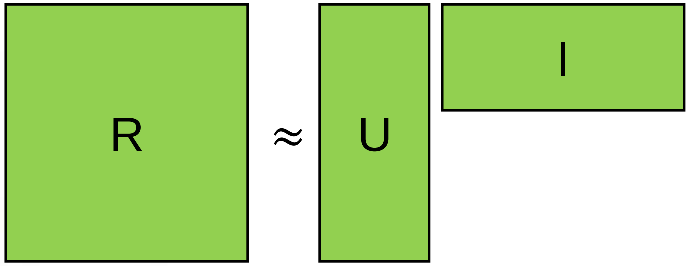
GRAVITY
Research & Development

Item embeddings

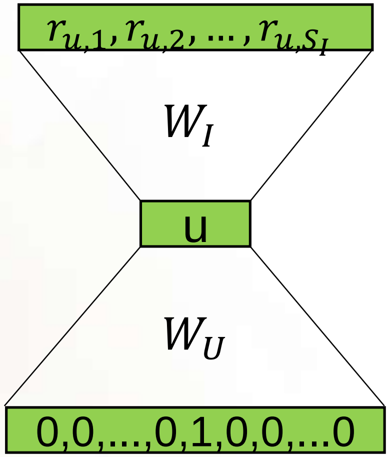
- Embedding: a (learned) real value vector representing an entity
  - Also known as:
    - Latent feature vector
    - (Latent) representation
  - Similar entities' embeddings are similar
- Use in recommenders:
  - Initialization of item representation in more advanced algorithms
  - Item-to-item recommendations

GRAVITY Research & Development

Matrix factorization as embedding learning



- MF: user & item embedding learning
  - Similar feature vectors
    - Two items are similar
    - Two users are similar
    - User prefers item
  - MF representation as a simplictic neural network
    - Input: one-hot encoded user ID
    - Input to hidden weights: user feature matrix
    - Hidden layer: user feature vector
    - Hidden to output weights: item feature matrix
    - Output: preference (of the user) over the items
- Asymmetric MF
  - Instead of user ID, the input is a vector of interactions over the items



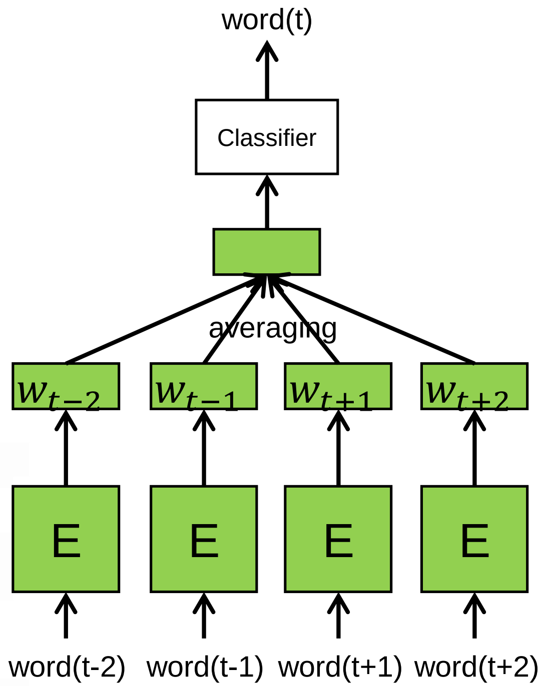
GRAVITY Research & Development

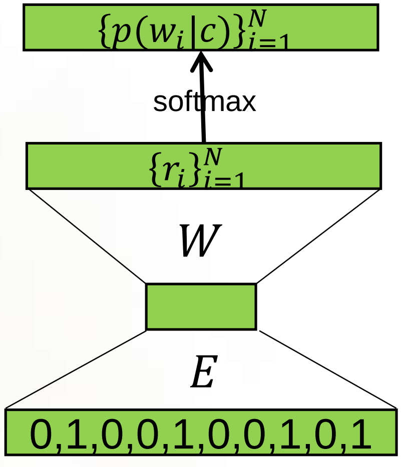
Word2Vec

- [Mikolov et. al, 2013a]
- Representation learning of words
- Shallow model
- Linear operations in the vector space can be associated with semantics
  - king – man + woman ~ queen
  - Paris – France + Italy ~ Rome
- Data: (target) word + context pairs
  - Sliding window on the document
  - Context = words near the target
    - In sliding window
    - 1-5 words in both directions
- Two models
  - Continous Bag of Words (CBOW)
  - Skip-gram

Word2Vec - CBOW

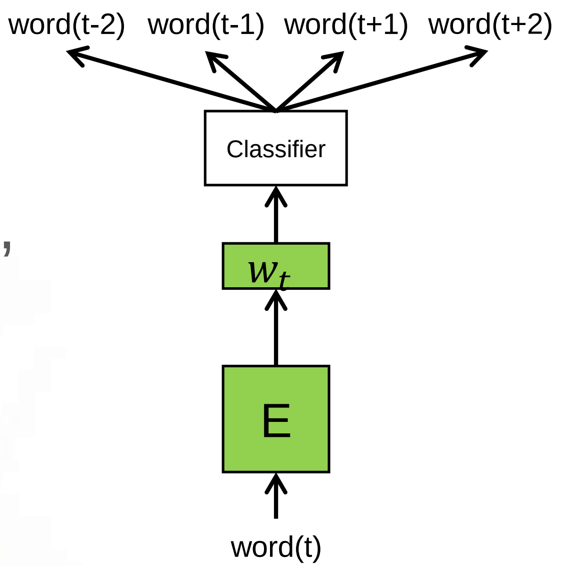
- Continuous Bag of Words
- Maximalizes the probability of the target word given the context
- Model
  - Input: one-hot encoded words
  - Input to hidden weights
    - Embedding matrix of words
  - Hidden layer
    - Sum of the embeddings of the words in the context
  - Hidden to output weights
  - Softmax transformation
    - Smooth approximation of the max operator
    - Highlights the highest value
    - $s_i = \frac{e^{r_i}}{\sum_{j=1}^{N} e^{r_j}}$, ($r_j$: scores)
  - Output: likelihood of words of the corpus given the context
- Embeddings are taken from the input to hidden matrix
  - Hidden to output matrix also has item representations (but not used)

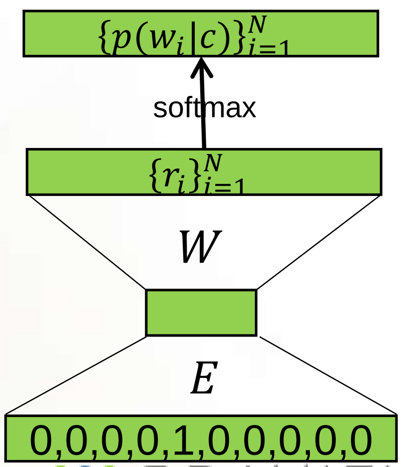




Word2Vec – Skip-gram

- Maximalizes the probability of the context, given the target word
- Model
  - Input: one-hot encoded word
  - Input to hidden matrix: embeddings
  - Hidden state
    - Item embedding of target
  - Softmax transformation
  - Output: likelihood of context words (given the input word)
- Reported to be more accurate





GRAVITY Research & Development

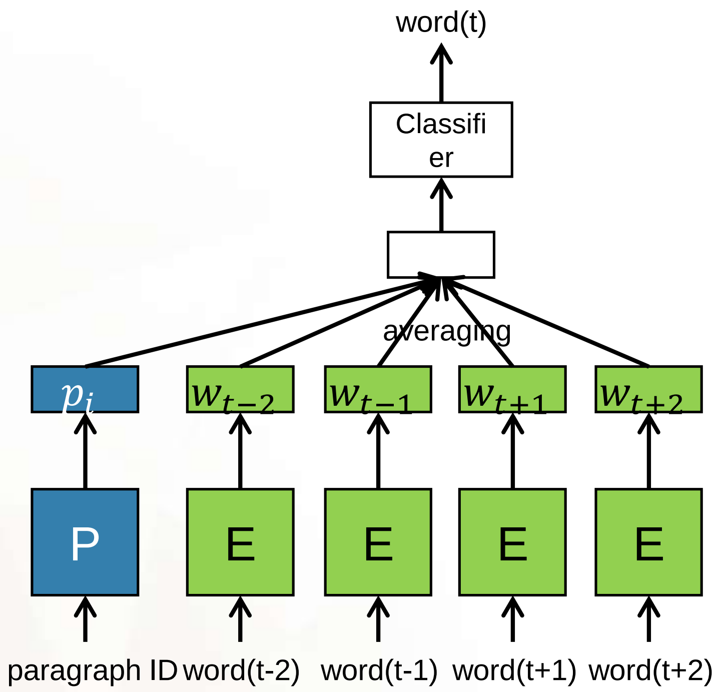
Speed-up

- Hierarchical softmax [Morin & Bengio, et. al, 2005]
  - Softmax computation requires every score
  - Reduce computations to $\mathrm{O}(\log_2 N)$ by using a binary tree
    - Leaves words
    - Each inner node has a trainable vector (v)
    - $\sigma(v^T v_c)$ is the probability that the left child of the current node is the next step we have to take in the tree
      - Probability of a word: $p(w|w_c) = \prod_{j=1}^{L(w_t)-1} \sigma\left(I_{n(w,j+1)=ch(n(w,j))} v_{n(w,j)}^T v_c\right)$
        - $n(w,j)$: j-th node on the path to w
        - $ch(n)$: left child of node n
    - During learning the vectors in the nodes are modified so that the target word becomes more likely
- Skip-gram with negative sampling (SGNS) [Mikolov, et. al, 2013b]
  - Input: target word
  - Desired output: sampled word from context
  - Score is computed for the desired output and a few negative samples

GRAVITY
Research & Development

Paragraph2vec, doc2vec

- [Le & Mikolov, 2014]
- Learns representation of paragraph/document
- Based on CBOW model
- Paragraph/document embedding added to the model as global context



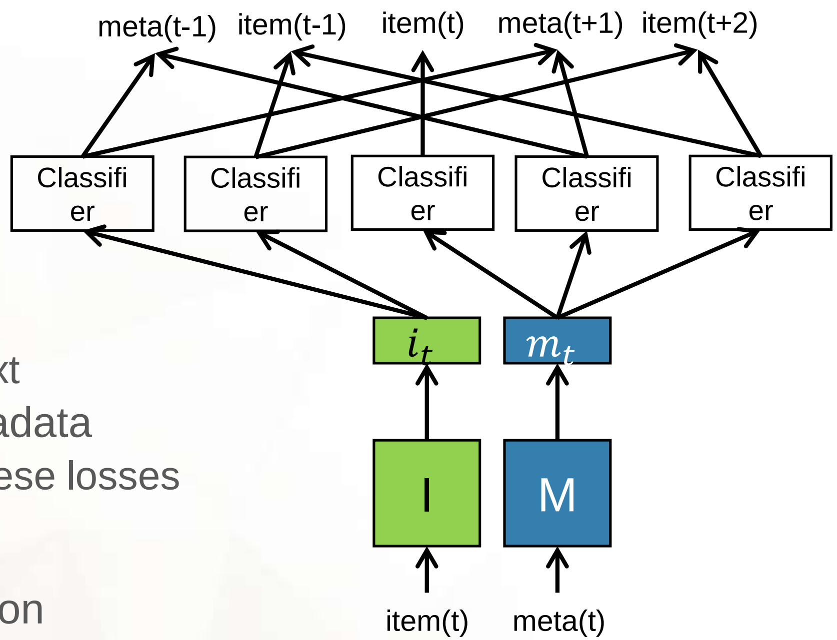
Prod2Vec

- [Grbovic et. al, 2015]
- Skip-gram model on products
  - Input: i-th product purchased by the user
  - Context: the other purchases of the user
- Bagged prod2vec model
  - Input: products purchased in one basket by the user
    - Basket: sum of product embeddings
  - Context: other baskets of the user
- Learning user representation
  - Follows paragraph2vec
  - User embedding added as global context
  - Input: user + products purchased except for the i-th
  - Target: i-th product purchased by the user
- [Barkan & Koenigstein, 2016] proposed the same model later as item2vec
  - Skip-gram with Negative Sampling (SGNS) is applied to event data

GRAVITY
Research & Development

Utilizing more information

- Meta-Prod2vec [Vasile et. al, 2016]
  - Based on the prod2vec model
  - Uses item metadata
    - Embedded metadata
    - Added to both the input and the context
  - Losses between: target/context item/metadata
    - Final loss is the combination of 5 of these losses
- Content2vec [Nedelec et. al, 2017]
  - Separate moduls for multimodel information
    - CF: Prod2vec
    - Image: AlexNet (a type of CNN)
    - Text: Word2Vec and TextCNN
  - Learns pairwise similarities
    - Likelihood of two items being bought together



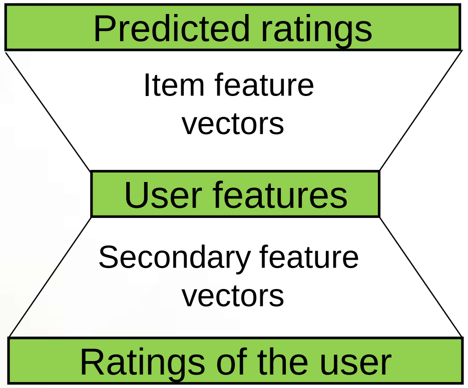
References

- [Barkan & Koenigstein, 2016] O. Barkan, N. Koenigstein: ITEM2VEC: Neural item embedding for collaborative filtering. IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP 2016).
- [Grbovic et. al, 2015] M. Grbovic, V. Radosavljevic, N. Djuric, N. Bhamidipati, J. Savla, V. Bhagwan, D. Sharp: E-commerce in Your Inbox: Product Recommendations at Scale. 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'15).
- [Le & Mikolov, 2014] Q. Le, T. Mikolov: Distributed Representations of Sentences and Documents. 31st International Conference on Machine Learning (ICML 2014).
- [Mikolov et. al, 2013a] T. Mikolov, K. Chen, G. Corrado, J. Dean: Efficient Estimation of Word Representations in Vector Space. ICLR 2013 Workshop.
- [Mikolov et. al, 2013b] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean: Distributed Representations of Words and Phrases and Their Compositionality. 26th Advances in Neural Information Processing Systems (NIPS 2013).
- [Morin & Bengio, 2005] F. Morin, Y. Bengio: Hierarchical probabilistic neural network language model. International workshop on artificial intelligence and statistics, 2005.
- [Nedelec et. al, 2017] T. Nedelec, E. Smirnova, F. Vasile: Specializing Joint Representations for the task of Product Recommendation. 2nd Workshop on Deep Learning for Recommendations (DLRS 2017).
- [Vasile et. al, 2016] F. Vasile, E. Smirnova, A. Conneau: Meta-Prod2Vec – Product Embeddings Using Side-Information for Recommendations. 10th ACM Conference on Recommender Systems (RecSys'16).

Deep collaborative filtering

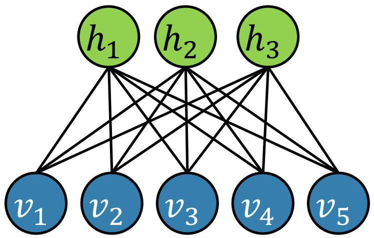
CF with Neural Networks

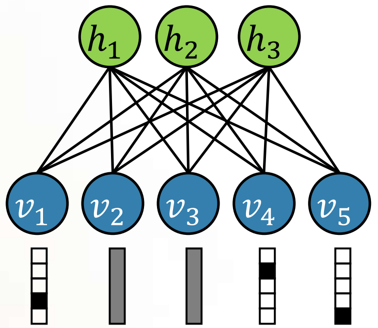
- Natural application area
- Some exploration during the Netflix prize
- E.g.: NSVD1 [Paterek, 2007]
  - Asymmetric MF
  - The model:
    - Input: sparse vector of interactions
      - Item-NSVD1: ratings given for the item by users
        - Alternatively: metadata of the item
      - User-NSVD1: ratings given by the user
    - Input to hidden weights: „secondary" feature vectors
    - Hidden layer: item/user feature vector
    - Hidden to output weights: user/item feature vectors
    - Output:
      - Item-NSVD1: predicted ratings on the item by all users
      - User-NSVD1: predicted ratings of the user on all items
  - Training with SGD
  - Implicit counterpart by [Pilászy et. al, 2009]
  - No non-linarities in the model



Restricted Boltzmann Machines (RBM) for recommendation

- RBM
  - Generative stochastic neural network
  - Visible & hidden units connected by (symmetric) weights
    - Stochastic binary units
    - Activation probabilities:
      - $p(h_j = 1|v) = \sigma\left(b_j^h + \sum_{i=1}^{m} w_{i,j} v_i\right)$
      - $p(v_i = 1|h) = \sigma\left(b_i^v + \sum_{j=1}^{n} w_{i,j} h_j\right)$
  - Training
    - Set visible units based on data
    - Sample hidden units
    - Sample visible units
    - Modify weights to approach the configuration of visible units to the data
- In recommenders [Salakhutdinov et. al, 2007]
  - Visible units: ratings on the movie
    - Softmax unit
      - Vector of length 5 (for each rating value) in each unit
      - Ratings are one-hot encoded
    - Units correnponding to users who not rated the movie are ignored
  - Hidden binary units



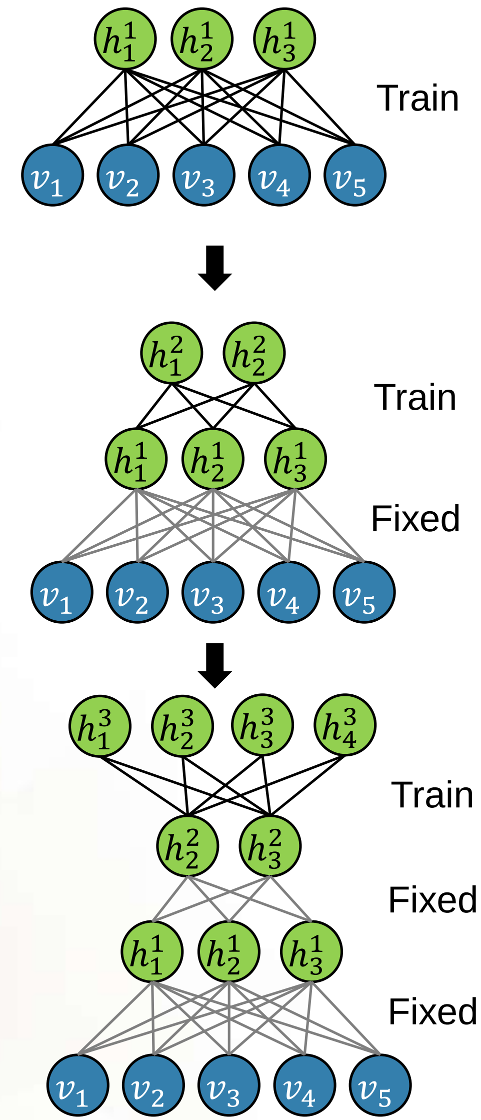


r_i: 2 ? ? 4 1

GRAVITY
Research & Development

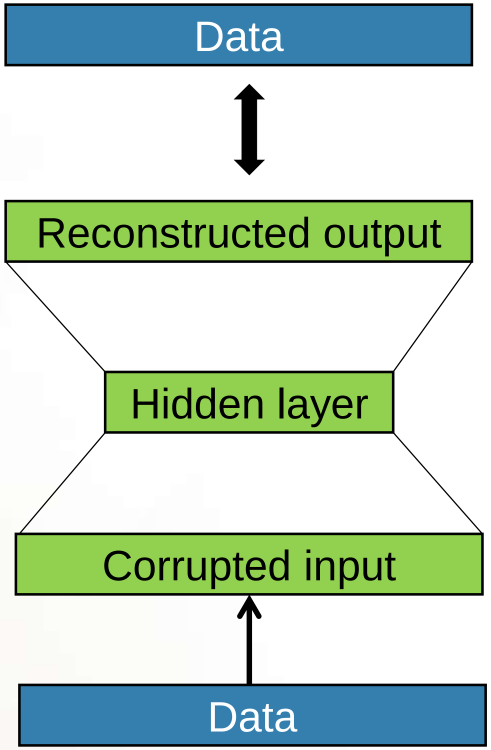
Deep Boltzmann Machines (DBM)

- Layer-wise training
  - Train weights between visible and hidden units in an RBM
  - Add a new layer of hidden units
  - Train weights connecting the new layer to the network
    - All other weights (e.g. visible-hidden weights) are fixed



Autoencoders

- Autoencoder
  - One hidden layer
  - Same number of input and output units
  - Try to reconstruct the input on the output
  - Hidden layer: compressed representation of the data
- Constraining the model: improve generalization
  - Sparse autoencoders
    - Activations of units is limited
    - Activation penalty
    - Requires the whole train set to compute
  - Denoising autoencoders [Vincent et. al, 2008]
    - Corrupt the input (e.g. set random values to zero)
    - Restore the original on the output
- Deep version
  - Stacked autoencoders
  - Layerwise training (historically)
  - End-to-end training (more recently)



Autoencoders for recommendation

- Reconstruct corrupted user interaction vectors
- Variants
  - CDL [Wang et. al, 2015]
    - Collaborative Deep Learning
    - Uses Bayesian stacked denoising autoencoders
    - Uses tags/metadata instead of the item ID
  - CDAE [Wu et. al, 2016]
    - Collaborative Denoising Auto-Encoder
    - Additional user node on the input and bias node beside the hidden layer

Recurrent autoencoder

- CRAE [Wang et. al, 2016]
  - Collaborative Recurrent Autoencoder
  - Encodes text (e.g. movie plot, review)
  - Autoencoding with RNNs
    - Encoder-decoder architecture
    - The input is corrupted by replacing words with a deisgnated BLANK token
  - CDL model + text encoding simultaneously
    - Joint learning

Other DeepCF methods (1/2)

- MV-DNN [Elkahky et. al, 2015]
  - Multi-domain recommender
  - Separate feedforward networks for user and items per domain (D+1 networks in total)
    - Features first are embedded
    - Then runthrough sevaral layers
  - Similarity of the final layers (user and item representation) is maximized over items the user visited (against negative examples)
- TDSSM [Song et. al, 2016]
  - Temporal Deep Semantic Structured Model
  - Similar to MV-DNN
  - User features are the combination of a static and a time dependent part
  - The time dependent part is modeled by an RNN
- Coevolving features [Dai et. al, 2016]
  - Users' taste and items' audiences change over time (e.g. forum discussions)
  - User/item features depend on time
  - User/item features are composed of
    - Time drift vector
    - Self evolution
    - Co-evolution with items/users
    - Interaction vector
  - Feature vectors are learned by RNNs

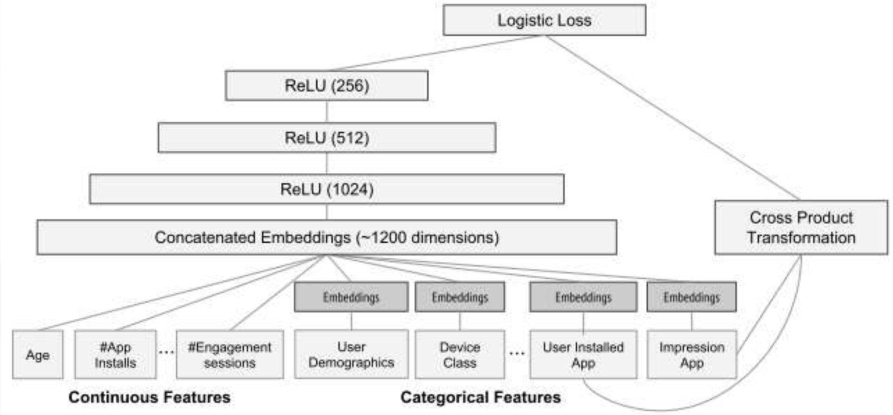
GRAVITY Research & Development

Other DeepCF methods (2/2)

- Product Neural Network (PNN) [Qu et. al, 2016]
  - For CTR estimation
  - Embedded features
  - Pairwise layer: all pairwise combination of embedded features
    - Like Factorization Machines
    - Outer/inner product of feature vectors or both
  - Several fully connected layers
- CF-NADE [Zheng et. al, 2016]
  - Neural Autoregressive Collaborative Filtering
  - User events → preference (0/1) + confidence (based on occurence)
  - Reconstructs some of the user events based on others (not the full set)
    - Random ordering of user events
    - Reconstruct the preference i, based on preferences and confidences up to i-1
  - Loss is weighted by confidences

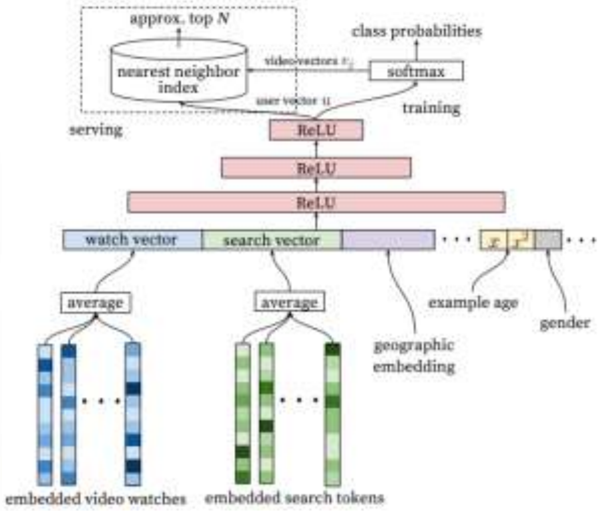
Applications: app recommendations

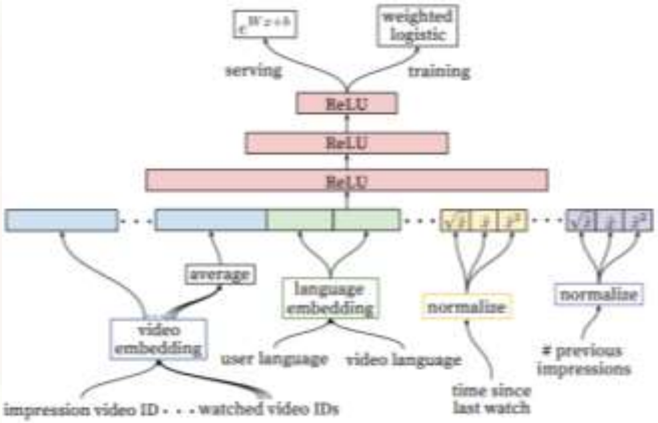
- Wide & Deep Learning [Cheng et. al, 2016]
- Ranking of results matching a query
- Combination of two models
  - Deep neural network
    - On embedded item features
    - „Generalization”
  - Linear model
    - On embedded item features
    - And cross product of item features
    - „Memorization”
  - Joint training
  - Logistic loss
- Improved online performance
  - +2.9% deep over wide
  - +3.9% deep+wide over wide



Applications: video recommendations

- YouTube Recommender [Covington et. al, 2016]
  - Two networks
  - Candidate generation
    - Recommendations as classification
      - Items clicked / not clicked when were recommended
    - Feedforward network on many features
      - Average watch embedding vector of user (last few items)
      - Average search embedding vector of user (last few searches)
      - User attributes
      - Geographic embedding
    - Negative item sampling + softmax
  - Reranking
    - More features
      - Actual video embedding
      - Average video embedding of watched videos
      - Language information
      - Time since last watch
      - Etc.
    - Weighted logistic regression on the top of the network





References

- [Cheng et. al, 2016] HT. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, H. Shah: Wide & Deep Learning for Recommender Systems. 1st Workshop on Deep Learning for Recommender Systems (DLRS 2016).
- [Covington et. al, 2016] P. Covington, J. Adams, E. Sargin: Deep Neural Networks for YouTube Recommendations. 10th ACM Conference on Recommender Systems (RecSys'16).
- [Dai et. al, 2016] H. Dai, Y. Wang, R. Trivedi, L. Song: Recurrent Co-Evolutionary Latent Feature Processes for Continuous-time Recommendation. 1st Workshop on Deep Learning for Recommender Systems (DLRS 2016).
- [Elkahky et. al, 2015] A. M. Elkahky, Y. Song, X. He: A Multi-View Deep Learning Approach for Cross Domain User Modeling in Recommendation Systems. 24th International Conference on World Wide Web (WWW'15). [Paterek, 2007] A. Paterek: Improving regularized singular value decomposition for collaborative filtering. KDD Cup and Workshop 2007.
- [Paterek, 2007] A. Paterek: Improving regularized singular value decomposition for collaborative filtering. KDD Cup 2007 Workshop.
- [Pilászy & Tikk, 2009] I. Pilászy, D. Tikk: Recommending new movies: even a few ratings are more valuable than metadata. 3rd ACM Conference on Recommender Systems (RecSys'09).
- [Qu et. al, 2016] Y. Qu, H. Cai, K. Ren, W. Zhang, Y. Yu: Product-based Neural Networks for User Response Prediction. 16th International Conference on Data Mining (ICDM 2016).
- [Salakhutdinov et. al, 2007] R. Salakhutdinov, A. Mnih, G. Hinton: Restricted Boltzmann Machines for Collaborative Filtering. 24th International Conference on Machine Learning (ICML 2007).
- [Song et. al, 2016] Y. Song, A. M. Elkahky, X. He: Multi-Rate Deep Learning for Temporal Recommendation. 39th International ACM SIGIR conference on Research and Development in Information Retrieval (SIGIR'16).
- [Vincent et. al, 2008] P. Vincent, H. Larochelle, Y. Bengio, P. A. Manzagol: Extracting and Composing Robust Features with Denoising Autoencoders. 25th international Conference on Machine Learning (ICML 2008).
- [Wang et. al, 2015] H. Wang, N. Wang, DY. Yeung: Collaborative Deep Learning for Recommender Systems. 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'15).
- [Wang et. al, 2016] H. Wang, X. Shi, DY. Yeung: Collaborative Recurrent Autoencoder: Recommend while Learning to Fill in the Blanks. Advances in Neural Information Processing Systems (NIPS 2016).
- [Wu et. al, 2016] Y. Wu, C. DuBois, A. X. Zheng, M. Ester: Collaborative Denoising Auto-encoders for Top-n Recommender Systems. 9th ACM International Conference on Web Search and Data Mining (WSDM'16)
- [Zheng et. al, 2016] Y. Zheng, C. Liu, B. Tang, H. Zhou: Neural Autoregressive Collaborative Filtering for Implicit Feedback. 1st Workshop on Deep Learning for Recommender Systems (DLRS 2016).

Feature extraction from content for hydrid recommenders

GRAVITY
Research & Development

Content features in recommenders

- Hybrid CF+CBF systems
  - Interaction data + metadata
- Model based hybrid solutions
  - Initiliazing
    - Obtain item representation based on metadata
    - Use this representation as initial item features
  - Regularizing
    - Obtain metadata based representations
    - The interaction based representation should be close to the metadata based
    - Add regularizing term to loss of this difference
  - Joining
    - Obtain metadata based representations
    - Have the item feature vector be a concatenation
      - Fixed metadata based part
      - Learned interaction based part

Feature extraction from content

- Deep learning is capable of direct feature extraction
  - Work with content directly
  - Instead (or beside) metadata
- Images
  - E.g.: product pictures, video thumbnails/frames
  - Extraction: convolutional networks
  - Applications (e.g.):
    - Fashion
    - Video
- Text
  - E.g.: product description, content of the product, reviews
  - Extraction
    - RNNs
    - 1D convolution networks
    - Weighted word embeddings
    - Paragraph vectors
  - Applications (e.g.):
    - News
    - Books
    - Publications
- Music/audio
  - Extraction: convolutional networks (or RNNs)

Convolutional Neural Networks (CNN)

- Speciality of images
  - Huge amount of information
    - 3 channels (RGB)
    - Lots of pixels
    - Number of weights required to fully connect a 320x240 image to 2048 hidden units:
      - 3*320*240*2048 = 471,859,200
  - Locality
    - Objects' presence are independent of their location or orientation
    - Objects are spatially restricted

Convolutional Neural Networks (CNN)

- Image input
  - 3D tensor
    - Width
    - Height
    - Channels (R,G,B)
- Text/sequence inputs
  - Matrix
  - of one-hot encoded entities
- Inputs must be of same size
  - Padding
- (Classic) Convolutional Nets
  - Convolution layers
  - Pooling layers
  - Fully connected layers

Convolutional Neural Networks (CNN)

- Convolutional layer (2D)
  - Filter
    - Learnable weights, arranged in a small tensor (e.g. 3x3xD)
      - The tensor's depth equals to the depth of the input
    - Recognizes certain patterns on the image
  - Convolution with a filter
    - Apply the filter on regions of the image
      - $y_{a,b} = f\left(\sum_{i,j,k} w_{i,j,k} I_{i+a-1,j+b-1,k}\right)$
        - Filters are applied over all channels (depth of the input tensor)
        - Activation function is usually some kind of ReLU
      - Start from the upper left corner
      - Move left by one and apply again
      - Once reaching the end, go back and shift down by one
    - Result: a 2D map of activations, high at places corresponding to the pattern recognized by the filter
  - Convolution layer: multiple filters of the same size
    - Input size $(W_1 \times W_2 \times D)$
    - Filter size $(F \times F \times D)$
    - Stride (shift value) (S)
    - Number of filters (N)
    - Output size: $\left(\frac{W_1 - F}{S} + 1\right) \times \left(\frac{W_2 - F}{S} + 1\right) \times N$
    - Number of weights: $F \times F \times D \times N$
  - Another way to look at it:
    - Hidden neurons organized in a $\left(\frac{W_1 - F}{S} + 1\right) \times \left(\frac{W_2 - F}{S} + 1\right) \times N$ tensor
    - Weights a shared between neurons with the same depth
    - A neuron processe an $F \times F \times D$ region of the input
    - Neighboring neurons process regions shifted by the stride value

1	3	8	0
0	7	2	1
2	5	5	1
4	2	3	0

→

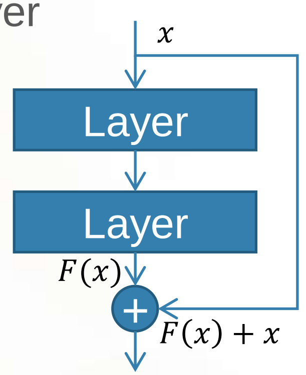
-1	-2	-1
-2	12	-2
-1	-2	-1

→

48	-27
19	28

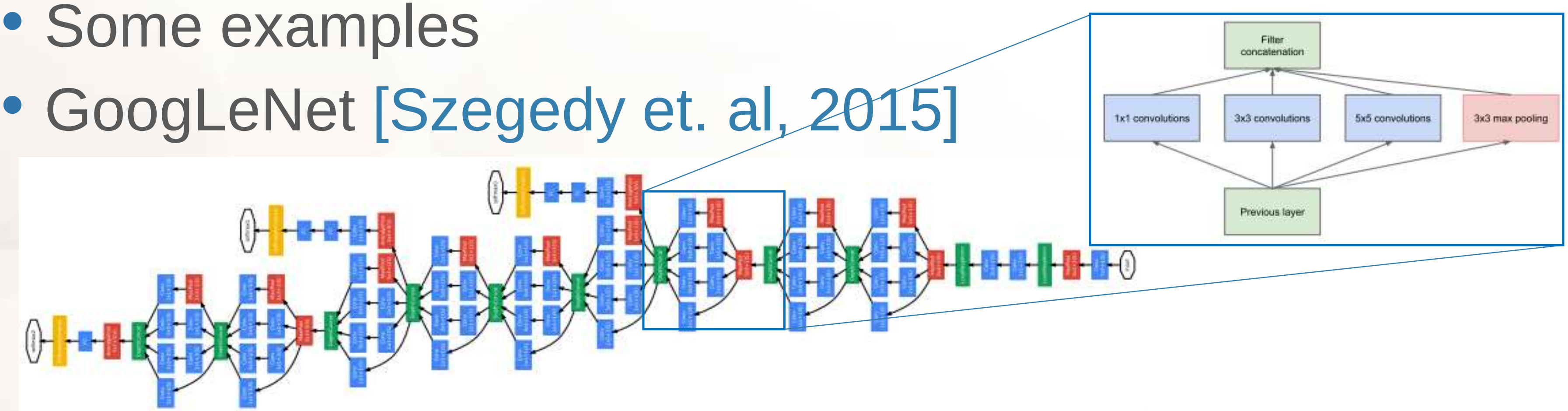
Convolutional Neural Networks (CNN)

- Pooling layer
  - Mean pooling: replace an $R \times R$ region with the mean of the values
  - Max pooling: replace an $R \times R$ region with the maximum of the values
  - Used to quickly reduce the size
  - Cheap, but very aggressive operator
    - Avoid when possible
    - Often needed, because convolutions don't decrease the number of inputs fast enough
  - Input size: $W_1 \times W_2 \times N$
  - Output size: $\frac{W_1}{R} \times \frac{W_2}{R} \times N$
- Fully connected layers
  - Final few layers
  - Each hidden neuron is connected with every neuron in the next layer
- Residual connections (improvement) [He et. al, 2016]
  - Very deep networks degrade performance
  - Hard to find the proper mappings
  - Reformulation of the problem: F(x) → F(x)+x

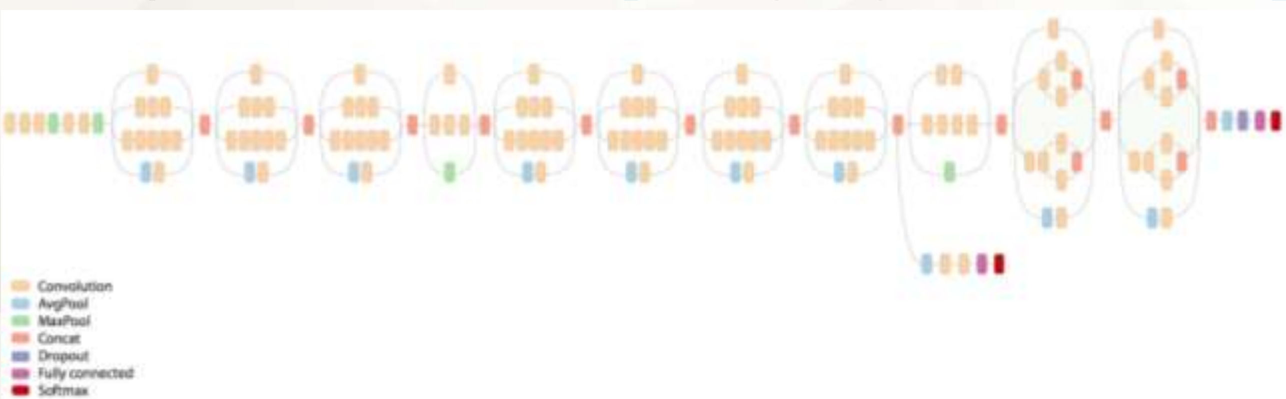


Convolutional Neural Networks (CNN)

- Some examples
- GoogLeNet [Szegedy et. al, 2015]



- Inception-v3 model [Szegedy et. al, 2016]



- ResNet (up to 200+ layers) [He et. al, 2016]

Images in recommenders

- [McAuley et. Al, 2015]
  - Learns a parameterized distance metric over visual features
    - Visual features are extracted from a pretrained CNN
    - Distance function: Eucledian distance of „embedded” visual features
      - Embedding here: multiplication with a weight matrix to reduce the number of dimensions
  - Personalized distance
    - Reweights the distance with a user specific weight vector
  - Training: maximizing likelihood of an existing relationship with the target item
    - Over uniformly sampled negative items
- Visual BPR [He & McAuley, 2016]
  - Model composed of
    - Bias terms
    - MF model
    - Visual part
      - Pretrained CNN features
      - Dimension reduction through „embedding”
      - The product of this visual item feature and a learned user feature vector is used in the model
    - Visual bias
      - Product of the pretrained CNN features and a global bias vector over its features
  - BPR loss
  - Tested on clothing datasets (9-25% improvement)

Music representations

- [Oord et. al, 2013]
  - Extends iALS/WMF with audio features
    - To overcome cold-start
  - Music feature extraction
    - Time-frequency representation
    - Applied CNN on 3 second samples
    - Latent factor of the clip: average predictions on consecutive windows of the clip
  - Integration with MF
    - (a) Minimize distance between music features and the MF’s feature vectors
    - (b) Replace the item features with the music features (minimize original loss)

Textual information improving recommendations

- [Bansal et. al, 2016]
  - Paper recommendation
  - Item representation
    - Text representation
      - Two layer GRU (RNN): bidirectional layer followed by a unidirectional layer
      - Representation is created by pooling over the hidden states of the sequence
    - ID based representation (item feature vector)
    - Final representation: ID + text added
  - Multi-task learning
    - Predict both user scores
    - And likelihood of tags
  - End-to-end training
    - All parameters are trained simultaneously (no pretraining)
    - Loss
      - User scores: weighted MSE (like in iALS)
      - Tags: weighted log likelihood (unobserved tags are downweighted)

References

- [Bansal et. al, 2016] T. Bansal, D. Belanger, A. McCallum: Ask the GRU: Multi-Task Learning for Deep Text Recommendations. 10th ACM Conference on Recommender Systems (RecSys'16).
- [He et. al, 2016] K. He, X. Zhang, S. Ren, J. Sun: Deep Residual Learning for Image Recognition. CVPR 2016.
- [He & McAuley, 2016] R. He, J. McAuley: VBPR: Visual Bayesian Personalized Ranking from Implicit Feedback. 30th AAAI Conference on Artificial Intelligence (AAAI' 16).
- [McAuley et. Al, 2015] J. McAuley, C. Targett, Q. Shi, A. Hengel: Image-based Recommendations on Styles and Substitutes. 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'15).
- [Oord et. al, 2013] A. Oord, S. Dieleman, B. Schrauwen: Deep Content-based Music Recommendation. Advances in Neural Information Processing Systems (NIPS 2013).
- [Szegedy et. al, 2015] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich: Going Deeper with Convolutions. CVPR 2015.
- [Szegedy et. al, 2016] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna: Rethinking the Inception Architecture for Computer Vision. CVPR 2016.

Recurrent Neural Networks & Session-based recommendations

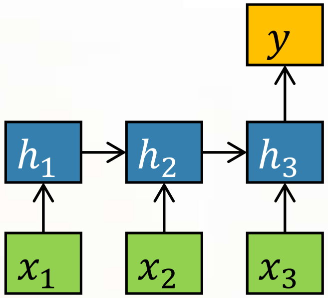
GRAVITY
Research & Development

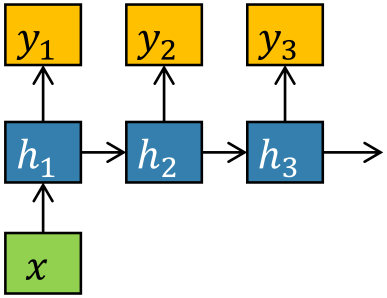
Recurrent Neural Networks

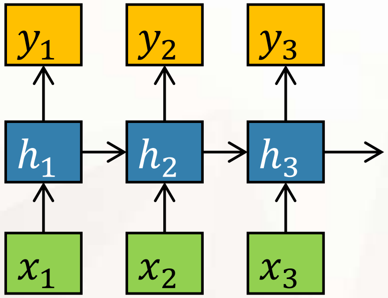
- Input: sequential information ($\{x_t\}_{t=1}^{T}$)
- Hidden state (h_t):
  - representation of the sequence so far
  - influenced by every element of the sequence up to t
- $h_t = f(Wx_t + Uh_{t-1} + b)$

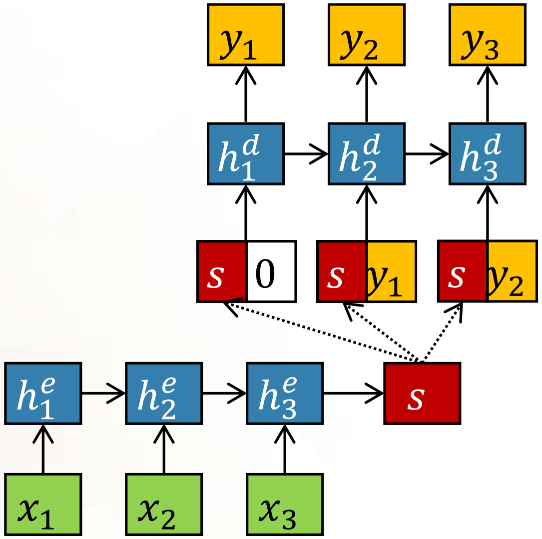
RNN-based machine learning

- Sequence to value
  - Encoding, labeling
  - E.g.: time series classification
- Value to sequence
  - Decoding, generation
  - E.g.: sequence generation
- Sequence to sequence
  - Simultaneous
    - E.g.: next-click prediction
  - Encoder-decoder architecture
    - E.g.: machine translation
    - Two RNNs (encoder & decoder)
      - Encoder produces a vector describing the sequence
        - Last hidden state
        - Combination of hidden states (e.g. mean pooling)
        - Learned combination of hidden states
      - Decoder receives the summary and generates a new sequence
        - The generated symbol is usually fed back to the decoder
        - The summary vector can be used to initialize the decoder
        - Or can be given as a global context
    - Attention mechanism (optionally)









Exploding/Vanishing gradients

- $h_t = f(Wx_t + Uh_{t-1} + b)$
- Gradient of h_t wrt. x_1
  - Simplification: linear activations
    - In reality: bounded
  - $\frac{\partial h_t}{\partial x_1} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial x_1} = U^{t-1}W$
    - $\|U\|_2 < 1$ → vanishing gradients
      - The effect of values further in the past is neglected
      - The network forgets
    - $\|U\|_2 > 1$ → exploding gradients
      - Gradients become very large on longer sequences
      - The network becomes unstable

GRAVITY
Research & Development

Handling exploding gradients

- Gradient clipping
  - If the gradient is larger than a threshold, scale it back to the threshold
  - Updates are not accurate
  - Vanishing gradients are not solved
- Enforce $||U||_2 = 1$
  - Unitary RNN
  - Unable to forget
- Gated networks
  - Long-Short Term Memory (LSTM)
  - Gated Recurrent Unit (GRU)
  - (and a some other variants)

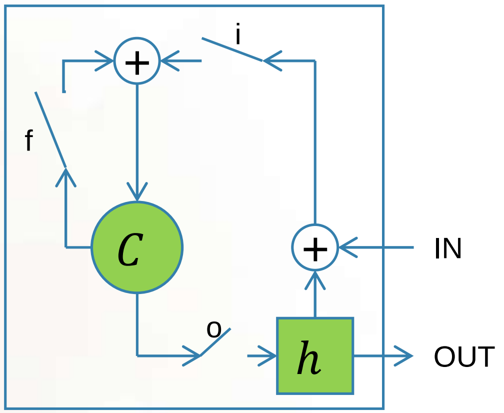
GRAVITY
Research & Development

Long-Short Term Memory (LSTM)

- [Hochreiter & Schmidhuber, 1999]
- Instead of rewriting the hidden state during update, add a delta
  - $s_t = s_{t-1} + \Delta s_t$
  - Keeps the contribution of earlier inputs relevant
- Information flow is controlled by gates
  - Gates depend on input and the hidden state
  - Between 0 and 1
  - Forget gate (f): 0/1 → reset/keep hidden state
  - Input gate (i): 0/1 → don't/do consider the contribution of the input
  - Output gate (o): how much of the memory is written to the hidden state
- Hidden state is separated into two (read before you write)
  - Memory cell (c): internal state of the LSTM cell
  - Hidden state (h): influences gates, updated from the memory cell

$$f_t = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right)$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

$$\tilde{c}_t = \tanh(W x_t + U h_{t-1} + b)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ \tanh(c_t)$$

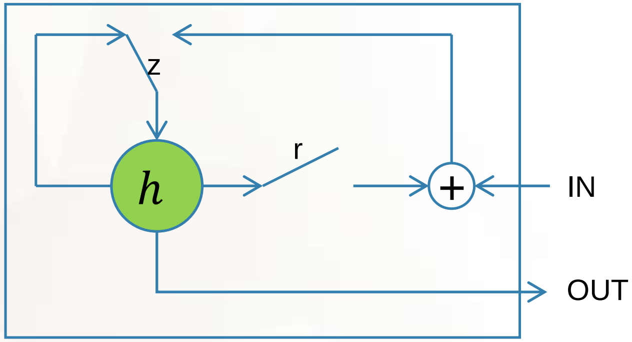


Gated Recurrent Unit (GRU)

- [Cho et. al, 2014]
- Simplified information flow
  - Single hidden state
  - Input and forget gate merged → update gate (z)
  - No output gate
  - Reset gate (r) to break information flow from previous hidden state
- Similar performance to LSTM

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$

$$\tilde{h}_t = \tanh(W x_t + r_t \circ U h_{t-1} + b)$$
$$h_t = z_t \circ h_t + (1 - z_t) \circ \tilde{h}_t$$



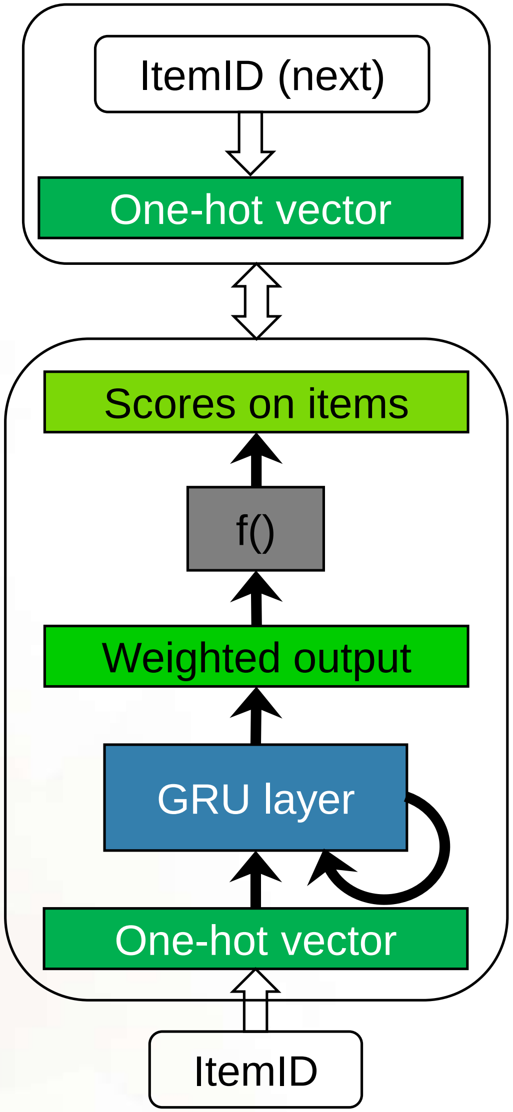
Session-based recommendations

- Sequence of events
  - User identification problem
  - Disjoint sessions (instead of consistent user history)
- Tasks
  - Next click prediction
  - Predicting intent
- Classic algorithms can’t cope with it well
  - Item-to-item recommendations as approximation in live systems
- Area revitalized by RNNs

GRAVITY
Research & Development

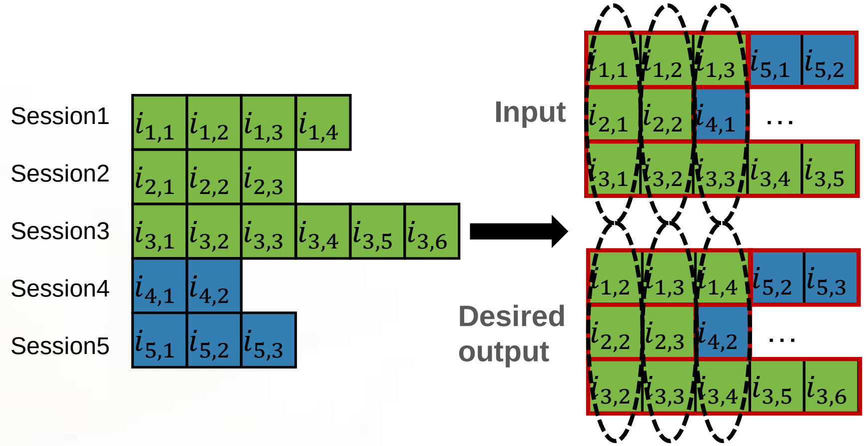
GRU4Rec (1/3)

- [Hidasi et. al, 2015]
- Network structure
  - Input: one hot encoded item ID
  - Optional embedding layer
  - GRU layer(s)
  - Output: scores over all items
  - Target: the next item in the session
- Adapting GRU to session-based recommendations
  - Sessions of (very) different length & lots of short sessions: session-parallel mini-batching
  - Lots of items (inputs, outputs): sampling on the output
  - The goal is ranking: listwise loss functions on pointwise/pairwise scores

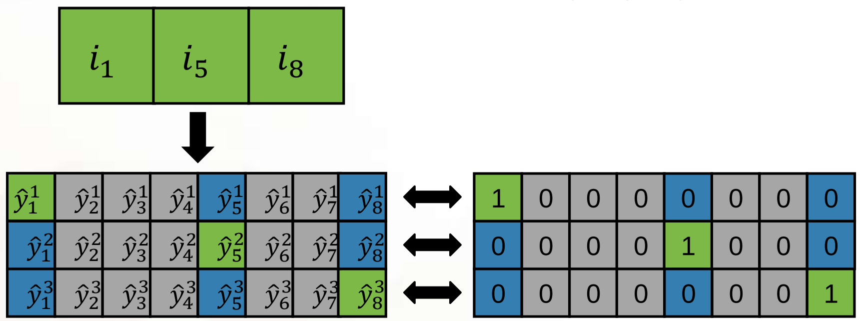


GRU4Rec (2/3)

- Session-parallel mini-batches
  - Mini-batch is defined over sessions
  - Update with one step BPTT
    - Lots of sessions are very short
    - 2D mini-batching, updating on longer sequences (with or without padding) didn't improve accuracy



- Output sampling
  - Computing scores for all items (100K – 1M) in every step is slow
  - One positive item (target) + several samples
  - Fast solution: scores on mini-batch targets
    - Items of the other mini-batch are negative samples for the current mini-batch



- Loss functions
  - Cross-entropy + softmax
  - Average of BPR scores
  - TOP1 score (average of ranking error + regularization over score values)

$$XE = -\log(s_i),\ s_i = \frac{e^{\hat{y}_i}}{\sum_{j=1}^{N_S} e^{\hat{y}_j}}$$

$$BPR = \frac{-\sum_{j=1}^{N_S} \log\left(\sigma\left(\hat{y}_i - \hat{y}_j\right)\right)}{N_S}$$

$$TOP1 = \frac{\sum_{j=1}^{N_S} \sigma\left(\hat{y}_j - \hat{y}_i\right) + \sum_{j=1}^{N_S} \sigma\left(\hat{y}_j^2\right)}{N_S}$$

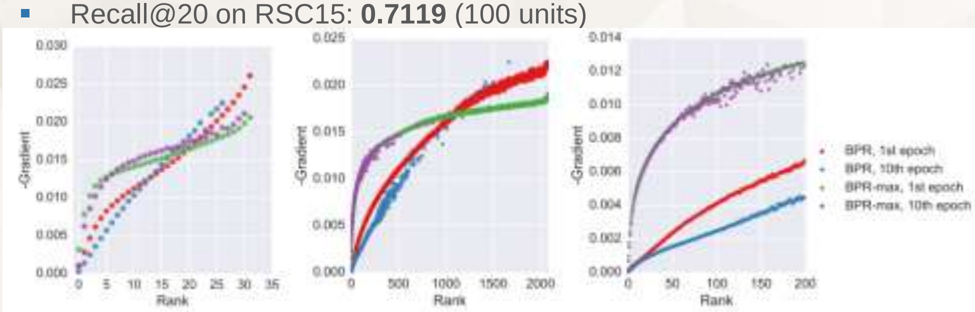
GRAVITY Research & Development

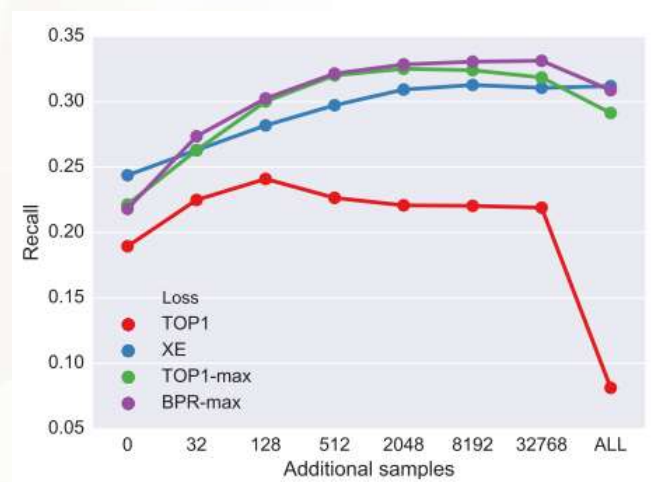
GRU4Rec (3/3)

- Observations
  - Similar accuracy with/without embedding
  - Multiple layers rarely help
    - Sometimes slight improvement with 2 layers
    - Sessions span over short time, no need for multiple time scales
  - Quick conversion: only small changes after 5-10 epochs
  - Upper bound for model capacity
    - No improvement when adding additional units after a certain threshold
    - This threshold can be lowered with some techniques
- Results
  - 20-30% improvement over item-to-item recommendations

Improving GRU4Rec

- Recall@20 on RSC15 by GRU4Rec: **0.6069** (100 units), **0.6322** (1000 units)
- Data augmentation [Tan et. al, 2016]
  - Generate additional sessions by taking every possible sequence starting from the beginning of a session
  - Randomly remove items from these sequences
  - Long training times
  - Recall@20 on RSC15 (using the full training set for training): ~**0.685** (100 units)
- Bayesian version (ReLeVar) [Chatzis et. al, 2017]
  - Bayesian formulation of the model
  - Basically additional regularization by adding random noise during sampling
  - Recall@20 on RSC15: **0.6507** (1500 units)
- New losses and additional sampling [Hidasi & Karatzoglou, 2017]
  - Use additional samples beside minibatch samples
  - Design better loss functions: $\mathrm{BPR}_{\max} = -\log\left(\sum_{j=1}^{N_S} s_j \sigma(r_i - r_j)\right) + \lambda \sum_{j=1}^{N_S} r_j^2$
  - Recall@20 on RSC15: **0.7119** (100 units)





Extensions

- Multi-modal information (p-RNN model) [Hidasi et. al, 2016]
  - Use image and description besides the item ID
  - One RNN per information source
  - Hidden states concatenated
  - Alternating training
- Item metadata [Twardowski, 2016]
  - Embed item metadata
  - Merge with the hidden layer of the RNN (session representation)
  - Predict compatibility using feedforward layers
- Contextualization [Smirnova & Vasile, 2017]
  - Merging both current and next context
  - Current context on the input module
  - Next context on the output module
  - The RNN cell is redefined to learn context-aware transitions
- Personalizing by inter-session modeling
  - Hierarchical RNNs [Quadrana et. al, 2017], [Ruocco et. al, 2017]
    - One RNN works within the session (next click prediction)
    - The other RNN predicts the transition between the sessions of the user

GRAVITY Research & Development

References

- [Chatzis et. al, 2017] S. P. Chatzis, P. Christodoulou, A. Andreou: Recurrent Latent Variable Networks for Session-Based Recommendation. 2nd Workshop on Deep Learning for Recommender Systems (DLRS 2017). https://arxiv.org/abs/1706.04026
- [Cho et. al, 2014] K. Cho, B. van Merrienboer, D. Bahdanau, Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. https://arxiv.org/abs/1409.1259
- [Hidasi et. al, 2015] B. Hidasi, A. Karatzoglou, L. Baltrunas, D. Tikk: Session-based Recommendations with Recurrent Neural Networks. International Conference on Learning Representations (ICLR 2016). https://arxiv.org/abs/1511.06939
- [Hidasi et. al, 2016] B. Hidasi, M. Quadrana, A. Karatzoglou, D. Tikk: Parallel Recurrent Neural Network Architectures for Feature-rich Session-based Recommendations. 10th ACM Conference on Recommender Systems (RecSys'16).
- [Hidasi & Karatzoglou, 2017] B. Hidasi, Alexandros Karatzoglou: Recurrent Neural Networks with Top-k Gains for Session-based Recommendations. https://arxiv.org/abs/1706.03847
- [Hochreiter & Schmidhuber, 1997] S. Hochreiter, J. Schmidhuber: Long Short-term Memory. Neural Computation, 9(8):1735-1780.
- [Quadrana et. al, 2017] M. Quadrana, A. Karatzoglou, B. Hidasi, P. Cremonesi: Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks. 11th ACM Conference on Recommender Systems (RecSys'17). https://arxiv.org/abs/1706.04148
- [Ruocco et. al, 2017] M. Ruocco, O. S. Lillestøl Skrede, H. Langseth: Inter-Session Modeling for Session-Based Recommendation. 2nd Workshop on Deep Learning for Recommendations (DLRS 2017). https://arxiv.org/abs/1706.07506
- [Smirnova & Vasile, 2017] E. Smirnova, F. Vasile: Contextual Sequence Modeling for Recommendation with Recurrent Neural Networks. 2nd Workshop on Deep Learning for Recommender Systems (DLRS 2017). https://arxiv.org/abs/1706.07684
- [Tan et. al, 2016] Y. K. Tan, X. Xu, Y. Liu: Improved Recurrent Neural Networks for Session-based Recommendations. 1st Workshop on Deep Learning for Recommendations (DLRS 2016). https://arxiv.org/abs/1606.08117
- [Twardowski, 2016] B. Twardowski: Modelling Contextual Information in Session-Aware Recommender Systems with Neural Networks. 10th ACM Conference on Recommender Systems (RecSys'16).

GRAVITY
Research & Development