Deep Learning architectures for NLP

G. Shang, A. Tixier, M. Vazirgiannis

LIX, Ecole Polytechnique

December 2018

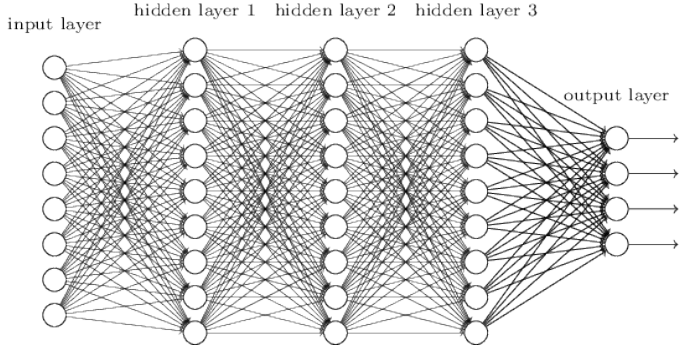
https://tinyurl.com/y9669llp

- Convolutional Neural Networks
- Recurrent NNs + LSTMs
- Other architectures (Attention, Siamese, EBL…)

Convolutional Neural Networks (CNNs)

NNs fully-connected layers to classify images.

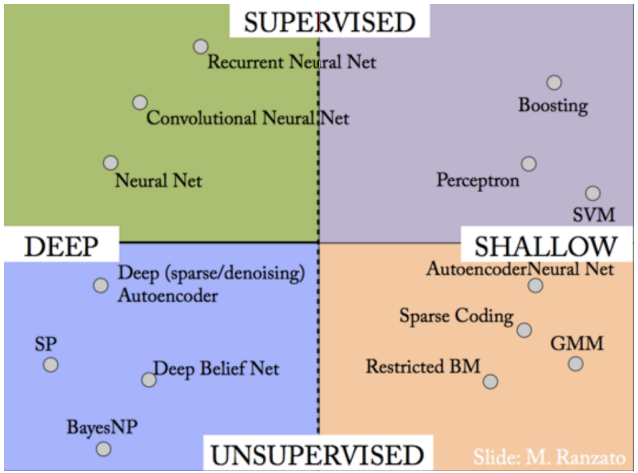
- do not take into account the spatial structure of the images.
  - treat input pixels far apart and close together in the same manner.
  - spatial structure must instead be inferred from the training data.



- CNNs takes advantage of the spatial structure
- special architecture well-adapted to classify images.
- fast to train thus feasibility for training deep, many-layer networks,
- deep CNNs used for image recognition, text classification etc.
- Basic components:
  - *local receptive fields*,
  - *shared weights*
  - *pooling*.

Deep Models

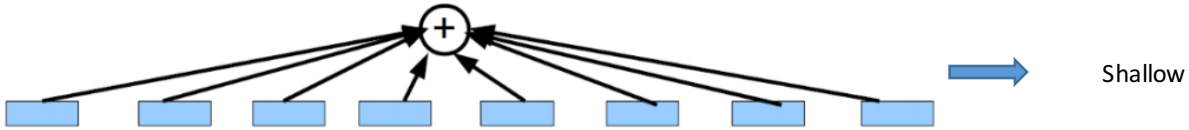
- Motivation: Automatic Feature Representation Learning
- Instead of designing features, design feature learners



Learning non-linear functions

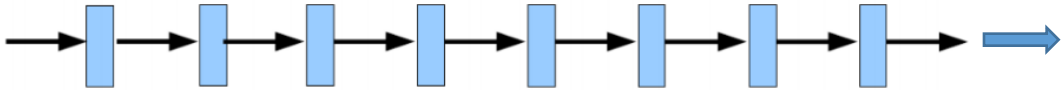
Given a dictionary of simple non-linear functions: $g_1, \ldots, g_n$

- Proposal 1: linear combination $f(x) \approx \sum_j g_j$



Shallow

- Proposal 2: composition $f(x) \approx g_1(g_2(\ldots g_n(x)\ldots))$



Deep

Convolutional Neural Networks

- In 1995, Yann LeCun and Yoshua Bengio introduced the concept of convolutional neural networks.
- Neuro-biologically motivated by the findings of locally sensitive and orientation-selective nerve cells in the visual cortex.
- They designed a network structure that implicitly extracts relevant features.
- Convolutional Neural Networks are a special kind of multi-layer neural networks.

[2] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. 3, 4

Convolutional Neural Networks

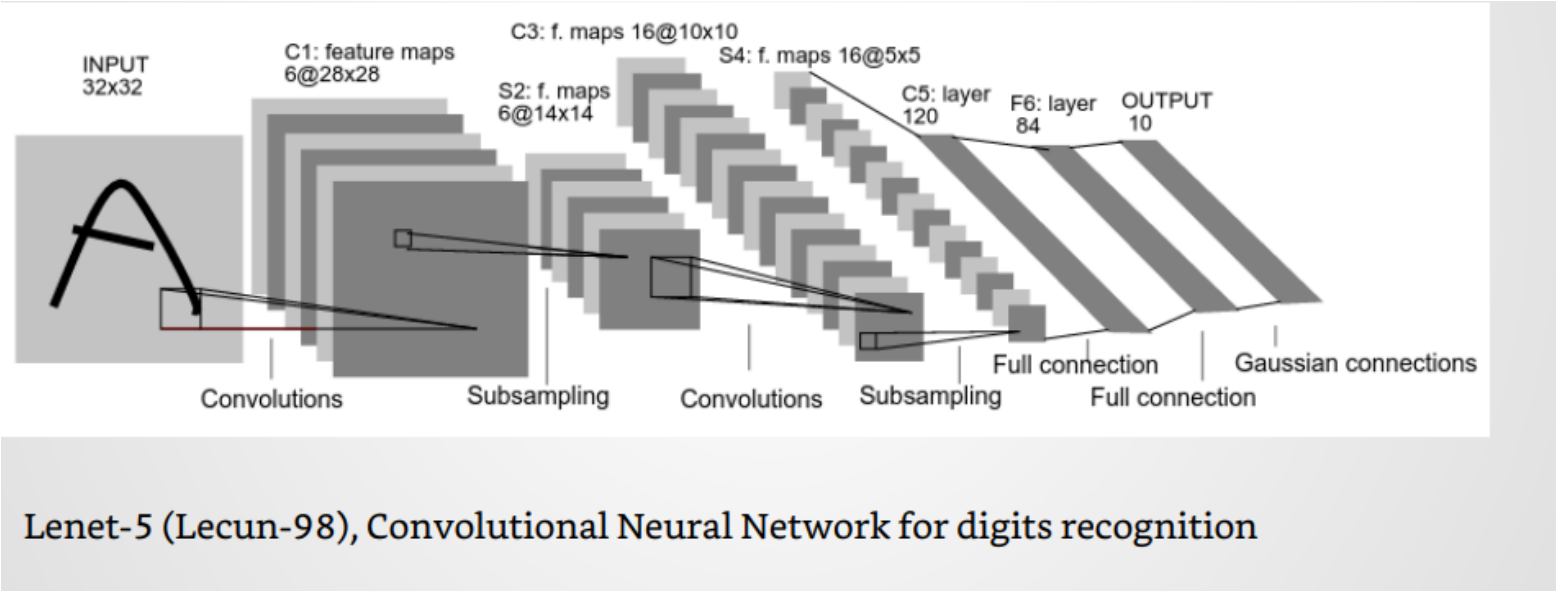
- inspired by studies of the cat's visual cortex [1], developed in computer vision to work on regular grids such as images [2].
- Feed-forward NNs , each neuron receives input from a neighborhood of the neurons (receptive fields) in the previous layer.
- Receptive fields, allow CNNs to recognize complex patterns in a hierarchical way, by combining lower-level, elementary features into higher-level features *compositionality*.
  - raw pixels=> edges =>shapes =>objects.
- absolute positions of features in the image are not important – only useful respective positions is useful composing higher-level patterns.
- Model detect a feature regardless of its position in the image - local invariance.
- Compositionality, local invariance two key concepts of CNNs.

[1] Hubel, David H., and Torsten N. Wiesel (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of physiology 160.1:106-154. 4
[2] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. 3, 4

CNN vs NN

- From a memory and capacity standpoint the CNN is not much bigger than a regular two layer network.
- At runtime the convolution operations are computationally expensive and take up about 67% of the time.
- CNN's are about 3X slower than their fully connected equivalents (size-wise).

CNN architecture



Lenet-5 (Lecun-98), Convolutional Neural Network for digits recognition

Convolution

- The convolution of f and g, written as f*g, is defined as the integral of the product of the two functions after one is reversed and shifted

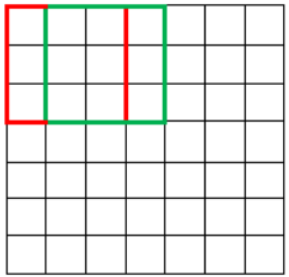
$$s(t) = \int x(a)w(t-a)da \qquad s(t) = (x * w)(t)$$

- Convolution is commutative.
- Can be viewed as a weighted average operation at every moment (for this w need to be a valid probability density function)
- Discrete Convolution:

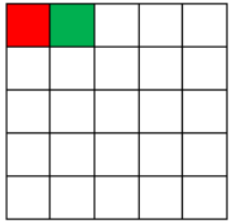
$$s[t] = (x * w)(t) = \sum_{a=-\infty}^{\infty} x[a]w[t-a]$$

Example

7 x 7 Input Volume

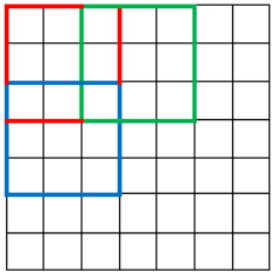


5 x 5 Output Volume

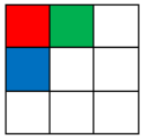


Stride = 1

7 x 7 Input Volume



3 x 3 Output Volume

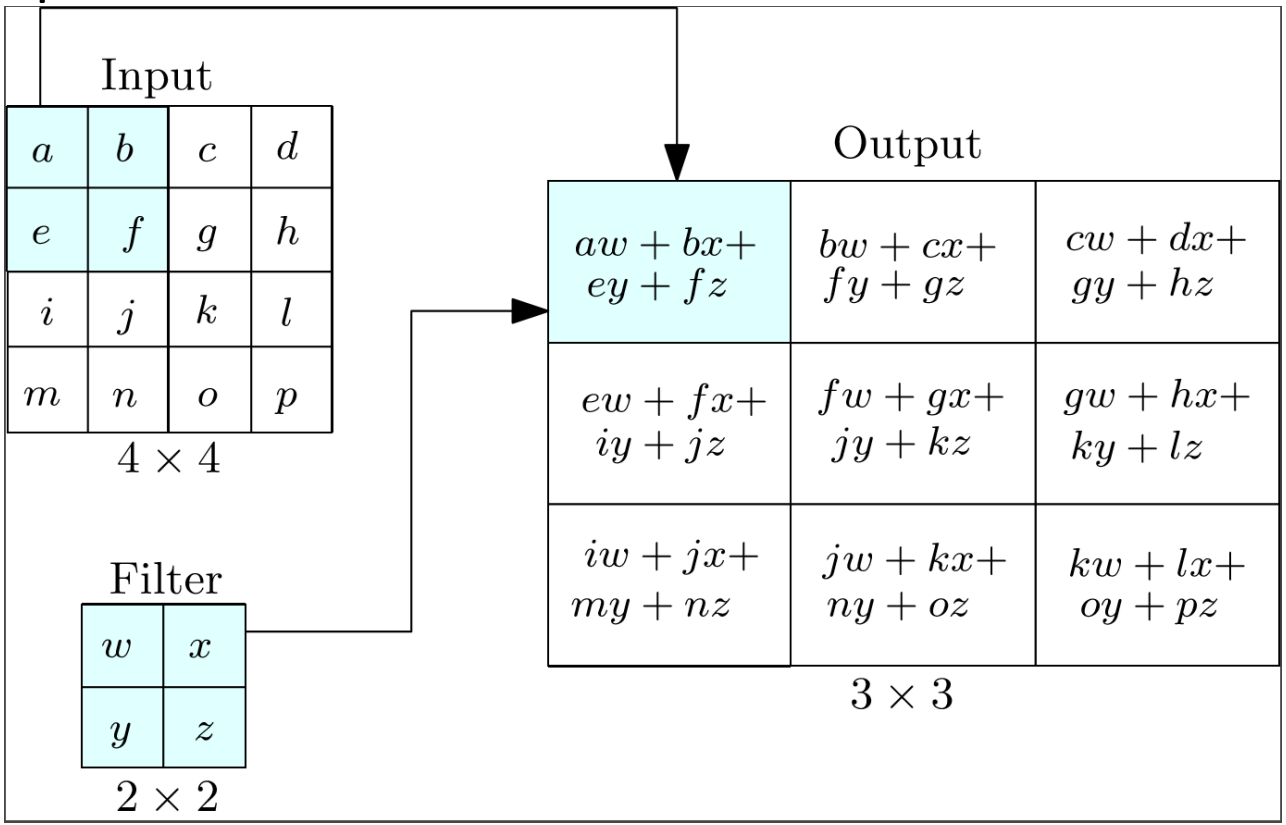


Stride = 2

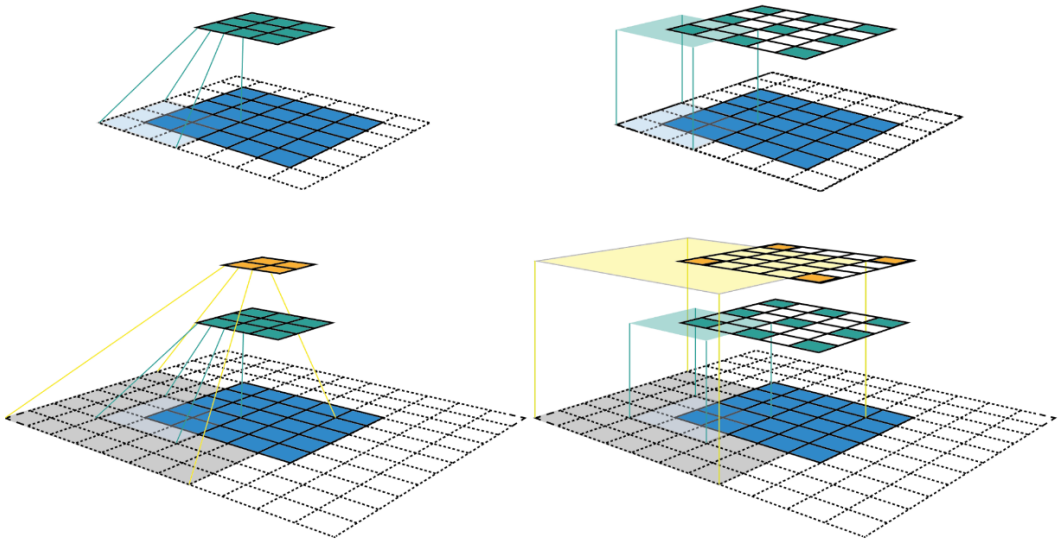
Convolution Variants

- **Full**: Add zero-padding to the feature matrix enough for every feature to be visited *k* times in each direction
  - (the maximum padding which does not result in a convolution over just padded elements)
- **Valid**: With no zero-padding, kernel is restricted to traverse only within the feature matrix
- **Same**: Add zero-padding to the feature matrix to have the output of the same size as the feature matrix
- **Stride**: Down-sampling the output of convolution by sampling only every *s* features in each direction.

Example


Input
a b c d
e f g h
i j k l
m n o p
4 × 4
Filter
w x
y z
2 × 2
Output
aw + bx + ey + fz
bw + cx + fy + gz
cw + dx + gy + hz
ew + fx + iy + jz
fw + gx + jy + kz
gw + hx + ky + lz
iw + jx + my + nz
jw + kx + ny + oz
kw + lx + oy + pz
3 × 3

CNN feature maps



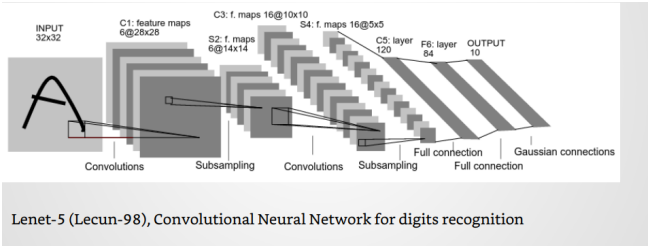
Why Convolution?

- Convolution exploits the property of spatial local correlations in the feature space by enforcing local connectivity pattern between neurons of adjacent layers

- Drastic reduce in the number of free parameters compared to fully connected network reducing overfitting and more importantly, computational complexity of the network

Feature maps

- Feature Map - Obtained by convolution of the feature matrix with a linear filter, adding a bias term and applying a non-linear function
- Non-linear functions:
  - Sigmoid $\frac{1}{1+e^{-x}}$
  - Tanh $\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$



Lenet-5 (Lecun-98), Convolutional Neural Network for digits recognition

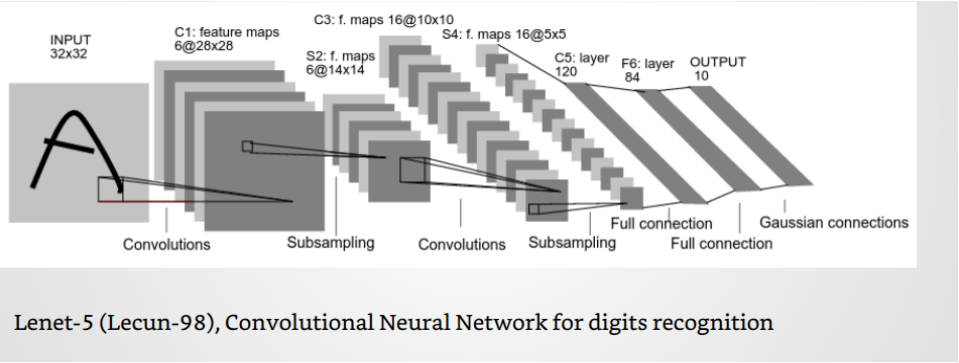
  - Rectified Linear Unit (ReLU) -> Most popular choice avoids saturation issues, makes learning faster

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

- Require a number of such feature maps at each layer to capture sufficient features

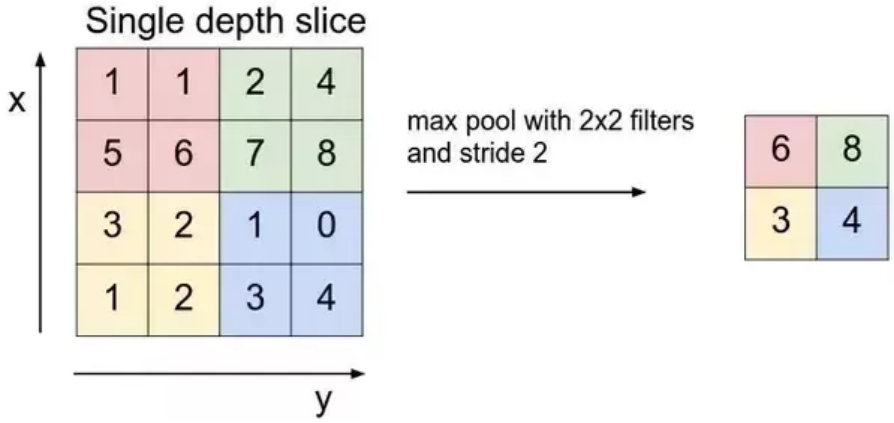
Pooling

- Sub-sampling layer
- Variants:
  - Max pooling
  - Weighted average
  - L2 norm of neighborhood
- Provides translation invariance
- Reduces computation



Lenet-5 (Lecun-98), Convolutional Neural Network for digits recognition

Max pooling - Example



See demo: http://cs231n.github.io/understanding-cnn/

How to reduce overfitting in CNNs

- Dropout: Randomly set the output value of network neurons to 0
  - Works as a regularization alternative
- Weight decay: keeps the magnitude of weights close to zero
- Data Augmentation: Slightly modified instances of the data

CNN for Text Classification

- Use the word embeddings of the document terms as input for Convolutional Neural Network
- Input must be fixed size
- Applies multiple filters to concatenated word vectors
- Produces new features for every filter
- picks the max as a feature for the CNN

CNN architecture for document classification

- Use the high quality embeddings as input for Convolutional Neural Network
- Applies multiple filters to concatenated word vector

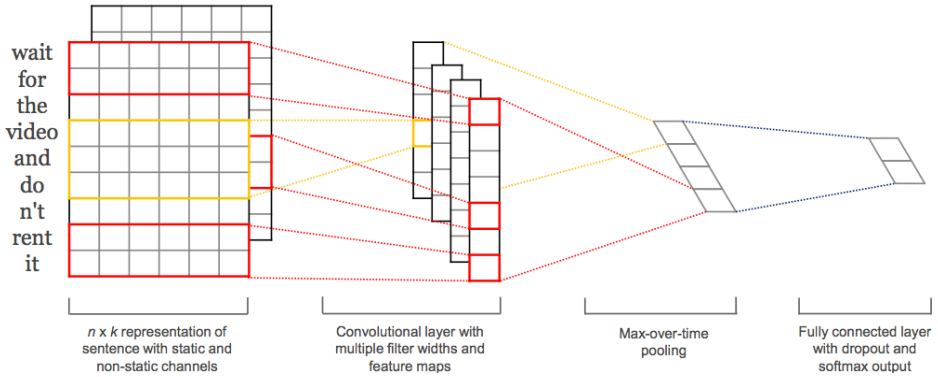
$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$$

- Produces new features for every filter

$$c_i = f(\mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b)$$

- And picks the max as a feature for the CNN

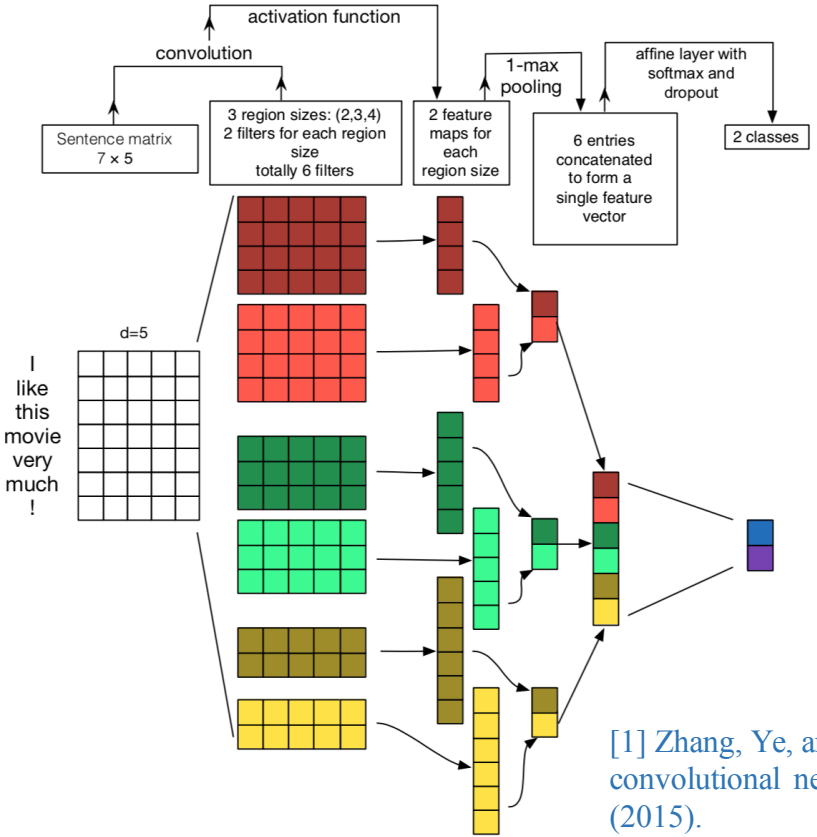
$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \qquad \hat{c} = \max\{\mathbf{c}\}$$



Yoon Kim - Convolutional Neural Networks for Sentence Classification

CNN architecture for document classification

[1]



- Data (text) only 1st column of input
- Rest of each row: embedding (in images 2D+RGB dimension)
- Filters of different sizes (4x5, 3x5 etc.)
  - Each size captures different features (need ~10^2 filters/size)
- Feature maps:
  - As many as the times filter fits on data matrix
- Max pooling maintains the “best features”
- Global feature map => classification via softmax

[1] Zhang, Ye, and Byron Wallace. "A sensitivity analysis of (and practitioners’ guide to) convolutional neural networks for sentence classification." arXiv preprint arXiv:1510.03820 (2015).

CNN for text classification

Many variations of the model [1]

- use existing vectors as input (CNN-static)
- learn vectors for the specific classification task through backpropagation (CNN-rand)
- Modify existing vectors for the specific task through backpropagation(CNN-non-static)

[1] Y. Kim, Convolutional Neural Networks for Sentence Classification, EMNLP 2014

CNN for text classification

- Combine multiple word embeddings
- Each set of vectors is treated as a ‘channel’
- Filters applied to all channels
- Gradients are back-propagated only through one of the channels
- Fine-tunes one set of vectors while keeping the other static

CNN for text classification

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

Accuracy scores (Kim et al vs others)

CNN architecture for (short) document classification – T-SNE visualization (see Lab notes)

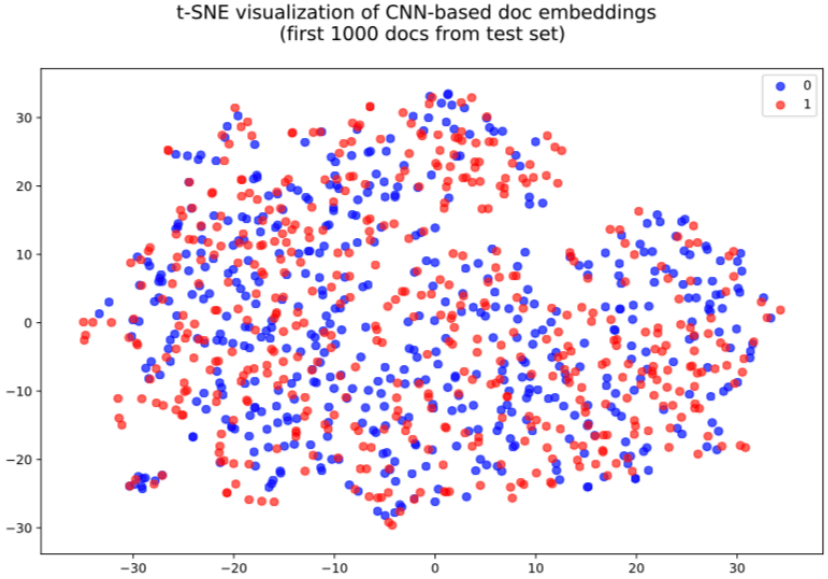


Figure 2: Doc embeddings before training.

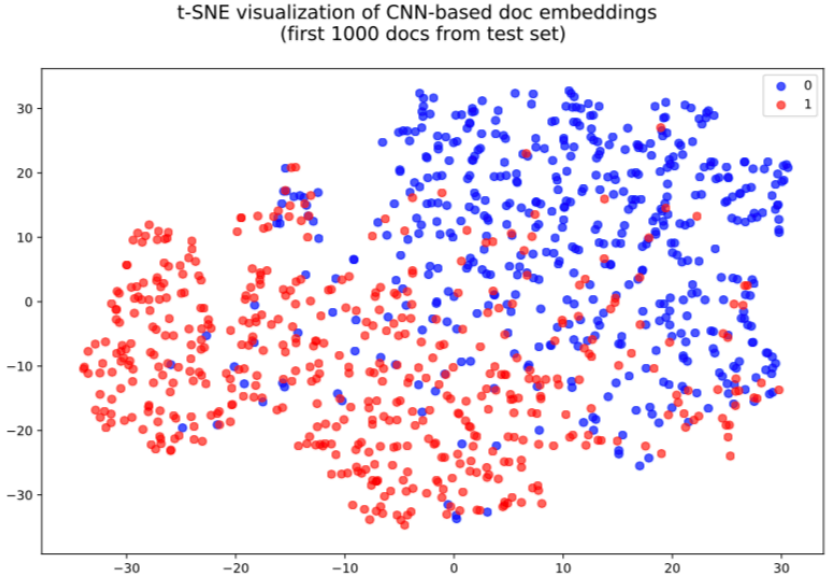


Figure 3: Doc embeddings after 2 epochs.

CNN architecture for document classification - Saliency maps (see Lab notes)

- words are most related to changing the doc classification
- A in R^{sxd}, s:# sentence words, d:size of embeddings

$$\text{saliency}(a) = \left| \frac{\partial(\text{CNN})}{\partial a} \Big|_a \right|$$

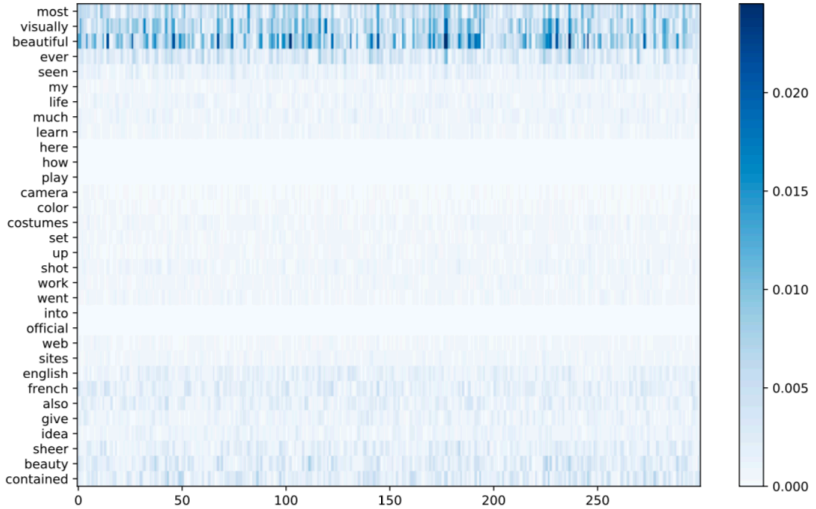


Figure 4: Saliency map for document 1 of the IMDB test set (true label: positive)

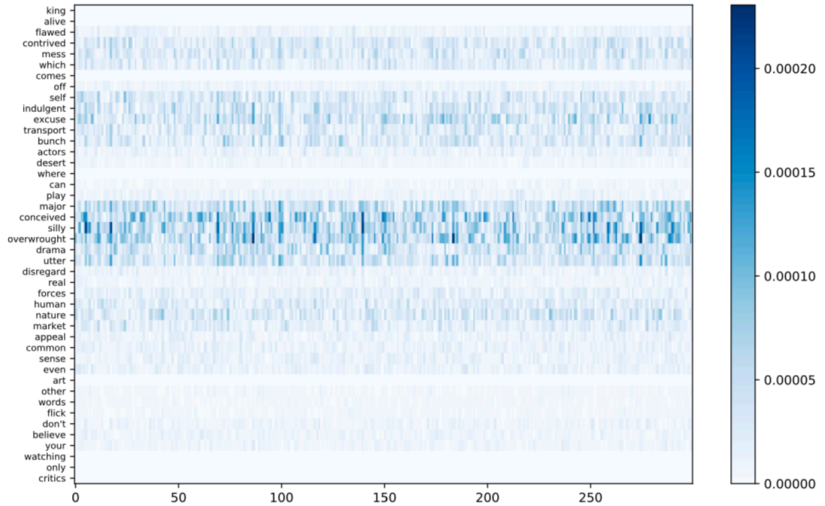
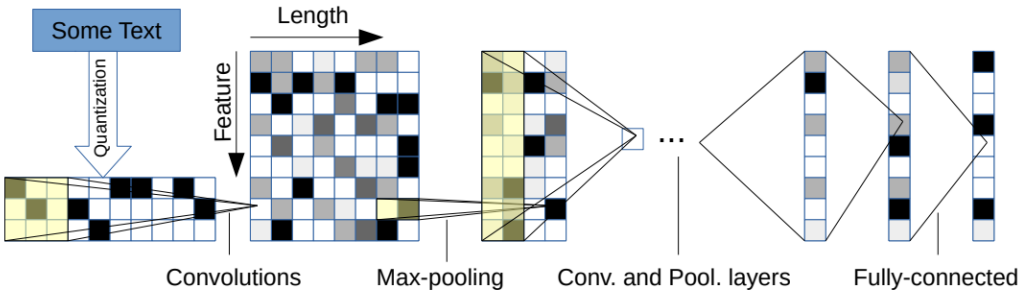


Figure 5: Saliency map for document 15 of the IMDB test set (true label: negative)

Character-level CNN for Text Classification

- Input: sequence of encoded characters
- quantize each character using “one-hot” encoding
- input feature length is 1014 characters
- 1014 characters able capture most of the texts of interest
- Also perform Data Augmentation using Thesaurus as preprocessing step

Model Architecture



- 9 layers deep
- 6 convolutional layers
- 3 fully-connected layers
- 2 dropout modules in between the fully-connected layers for regularization

Model Comparison

Model	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
BoW	11.19	7.15	3.39	7.76	42.01	31.11	45.36	9.60
BoW TFIDF	10.36	6.55	2.63	6.34	40.14	28.96	44.74	9.00
ngrams	7.96	2.92	1.37	**4.36**	43.74	31.53	45.73	7.98
ngrams TFIDF	**7.64**	**2.81**	**1.31**	4.56	45.20	31.49	47.56	8.46
Bag-of-means	**16.91**	**10.79**	**9.55**	**12.67**	**47.46**	**39.45**	**55.87**	**18.39**
LSTM	13.94	4.82	1.45	5.26	41.83	29.16	40.57	6.10
Lg. w2v Conv.	9.92	4.39	1.42	4.60	40.16	31.97	44.40	5.88
Sm. w2v Conv.	11.35	4.54	1.71	5.56	42.13	31.50	42.59	6.00
Lg. w2v Conv. Th.	9.91	-	1.37	4.63	39.58	31.23	43.75	5.80
Sm. w2v Conv. Th.	10.88	-	1.53	5.36	41.09	29.86	42.50	5.63
Lg. Lk. Conv.	8.55	4.95	1.72	4.89	40.52	29.06	45.95	5.84
Sm. Lk. Conv.	10.87	4.93	1.85	5.54	41.41	30.02	43.66	5.85
Lg. Lk. Conv. Th.	8.93	-	1.58	5.03	40.52	28.84	42.39	5.52
Sm. Lk. Conv. Th.	9.12	-	1.77	5.37	41.17	28.92	43.19	5.51
Lg. Full Conv.	9.85	8.80	1.66	5.25	38.40	29.90	40.89	5.78
Sm. Full Conv.	11.59	8.95	1.89	5.67	38.82	30.01	40.88	5.78
Lg. Full Conv. Th.	9.51	-	1.55	4.88	38.04	29.58	40.54	5.51
Sm. Full Conv. Th.	10.89	-	1.69	5.42	**37.95**	29.90	40.53	5.66
Lg. Conv.	12.82	4.88	1.73	5.89	39.62	29.55	41.31	5.51
Sm. Conv.	15.65	8.65	1.98	6.53	40.84	29.84	40.53	5.50
Lg. Conv. Th.	13.39	-	1.60	5.82	39.30	**28.80**	40.45	**4.93**
Sm. Conv. Th.	14.80	-	1.85	6.49	40.16	29.84	**40.43**	5.67

Testing errors for all models

Blue->best, Red->worst

links

- http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
- https://arxiv.org/pdf/1509.01626.pdf
- http://www.aclweb.org/anthology/D14-1181
- http://cs231n.github.io/convolutional-networks/
- http://ufldl.stanford.edu/tutorial/supervised/Pooling/

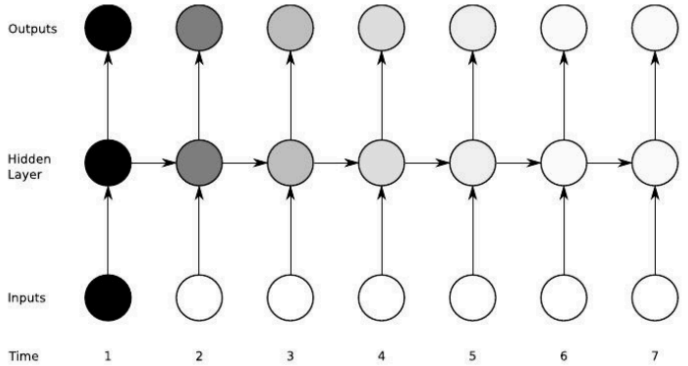
Recurrent Neural Networks

- CNNs are good at image classification
  - Input: an image
  - Output: probability of the class
    - $p(Red\ Panda|\ image) = 0,9$
    - $p(Cat\ |\ image) = 0,1$

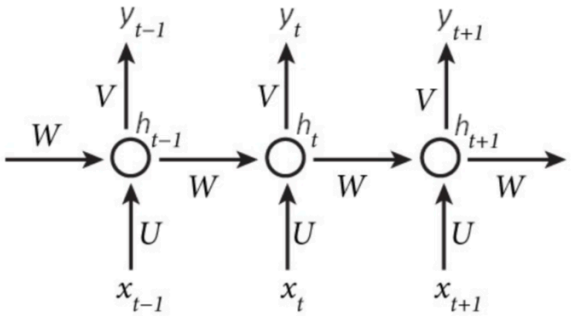
- Sequence learning: study of machine learning algorithms designed for sequential data (time series, language…)
  - Translate: "*machine learning is a challenging topic*" to French: "*L'apprentissage automatique est un sujet difficile*"
  - Predict the next word: "*John visited Paris, the capital of ….* "
  - Input and output strongly correlated within the sequence.

Recurrent Neural Networks

- Update the hidden state in a deterministic nonlinear way.
- RNNs are powerful,
- Distributed hidden state that allows them to store a lot of information about the past efficiently.
- Non-linear dynamics that allows them to update their hidden state in complicated ways.
- No need to infer hidden state, pure deterministic.
- Weight sharing

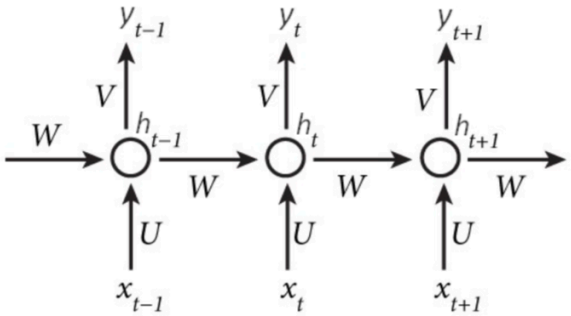


Recurrent Neural Networks



- input: ordered list of input vectors $x_1, \ldots, x_T$ initial hidden state h_0 initialized to all zeros,
- output
  - ordered list of hidden states $h_1, \ldots, h_T$,
  - ordered list of output vectors $y_1, \ldots, y_T$.
  - The output vectors may serve as input for other RNN units, when considering deep architectures .
  - The hidden states correspond to the “short-term” memory of the network.
- The last hidden state represents the encoding (embedding) of the time series

Recurrent Neural Networks



$$h_t = f(Ux_t + Wh_{t-1} + b)$$

- f a nonlinear function
- $x_t \in R^{d_{in}}, U \in R^{Hxd_{in}}, W \in R^{HxH}$,

Parameter matrices

- d_{in} size of the vocabulary
- H: dimension of the hidden layer (H~100)
- $y_t \in R^{d_{out}}$ transforms the current hidden state h_t to depend on the final task:
  - i.e. for classification $y_t = softmax(Vh_t)$
- $V \in R^{d_{out}\chi H}$ parameter matrix shared across all steps (i.e. for word level language model $dout=|V|$

Deep RNNs

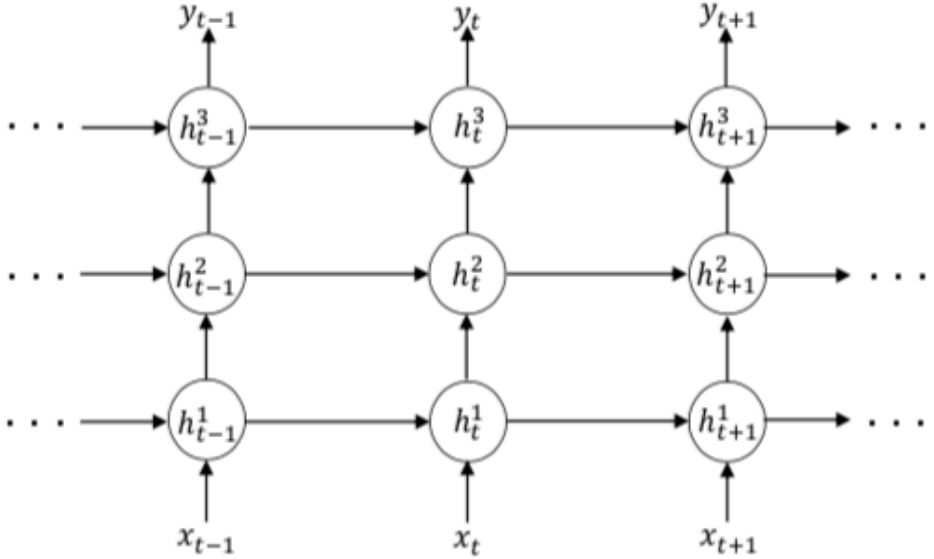
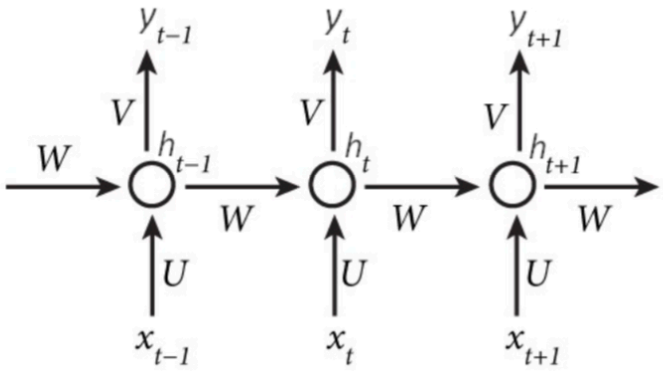


Figure 7: 3 steps of an unrolled deep RNN. Each circle represents a RNN unit. The hidden state of each unit in the inner layers (1 & 2) serves as input to the corresponding unit in the layer above.

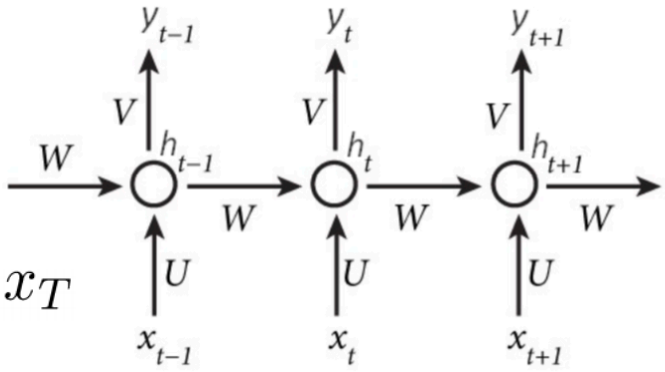
Recurrent Neural Networks



3 steps of an unrolled RNN

- CNNs are naturally efficient with grids[8],
- RNNs were specifically developed to be used with sequences
  - time series, or, in NLP, words (sequences of letters) or sentences (sequences of words).
  - language modeling P [$w_n | w_1, ..., w_{n-1}$].
  - RNNs trained with such objectives can be used to generate new and quite convincing sentences from scratch
  - RNN can be considered as a chain of simple neural layers that share the same parameters.

Learning in RNNs



- Assuming input $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$
- As a single time: $h_t = \sigma(W h_{t-1} + U x_t)$

$$y_t = softmax(V h_t)$$

Learning in RNNs

$$h_t = \sigma(W h_{t-1} + U x_t)$$
$$y_t = softmax(V h_t)$$

- Main idea: we use the same set of W,U,V weights at all time steps!
- $h_0 \in R^{HxH}$ initialization vector for the hidden layer at time step 0
- $\hat{y} \in R^{|V|}$ is a probability distribution over the vocabulary
- Loss function (@ time t): cross entropy predicting words instead of classes

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

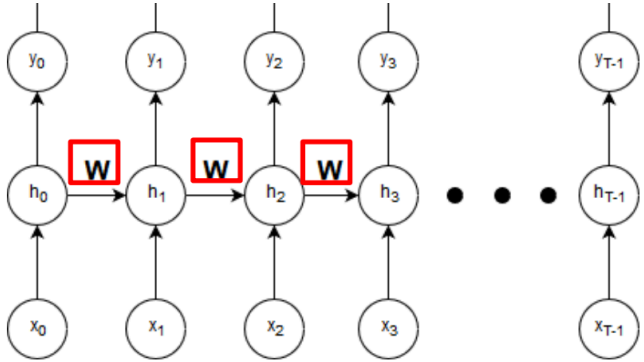
Learning in RNNs – backpropagation in time

- Learning the weights of W:

$$\mathbf{W} -> \mathbf{W} - \alpha \frac{\partial \mathbf{y}}{\partial \mathbf{W}}$$

- Computation involves summation over all paths regarding
- i. time

$$\frac{\partial \mathbf{y}}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \frac{\partial \mathbf{y}_j}{\partial \mathbf{W}}$$



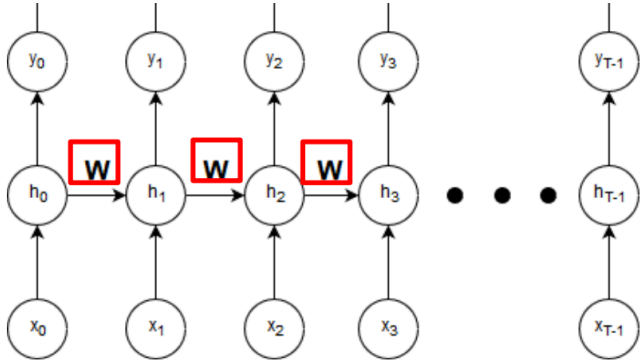
- ii. Levels of hidden layers

$$\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

Learning in RNNs – backpropagation in time

- Therefore:

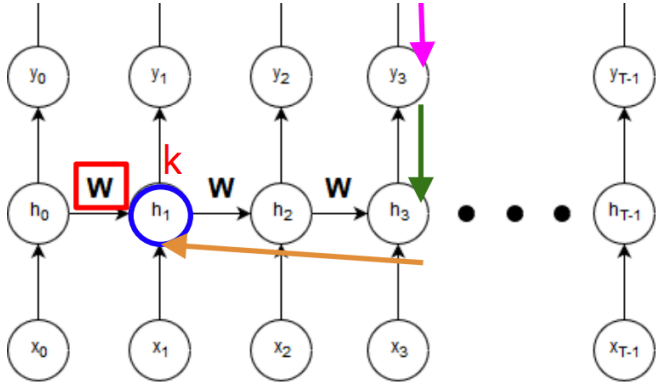
$$\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$



- Indirect dependency. One final use of the chain rule:

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}}$$

Learning in RNNs – backpropagation in time



$$\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W}}$$

Learning in RNNs – vanishing/exploding gradients

$$\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W}}$$

$$h_m = f(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)$$

$$\frac{\partial h_m}{\partial h_{m-1}} = \mathbf{W}_h^T \operatorname{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \mathbf{W}_h^T \operatorname{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

Weight Matrix

Derivative of activation function

- Repeated matrix multiplications leads to vanishing and exploding gradients.

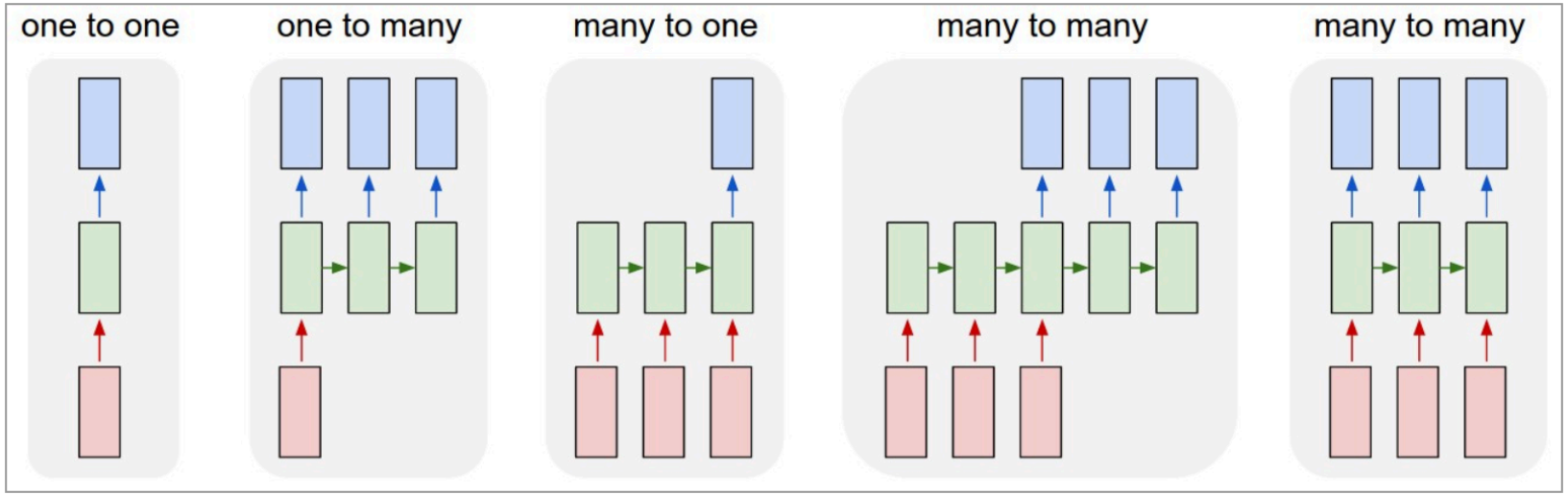
Learning in RNNs – vanishing/exploding gradients

- Initialization of W with 1s
- Using relu as activation function $f(z) = rect(z) = max(z, 0)$
- Using - clip gradients to a maximum value [Mikolov]

Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

RNN Applications



fixed-sized input to fixed-sized output (e.g. image classification).

Sequence output (e.g. image captioning takes an image and outputs a sentence of words).

Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).

Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French).

Synced sequence input and output (e.g. video classification where we wish to label each frame of the video)..

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

LSTM Unit

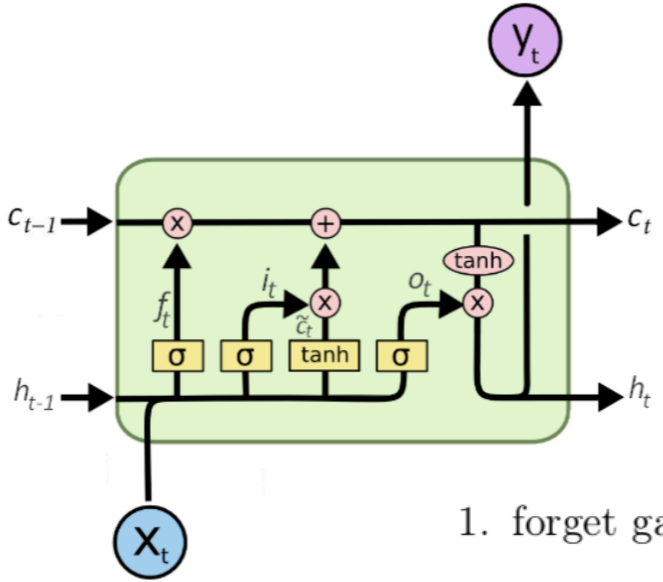
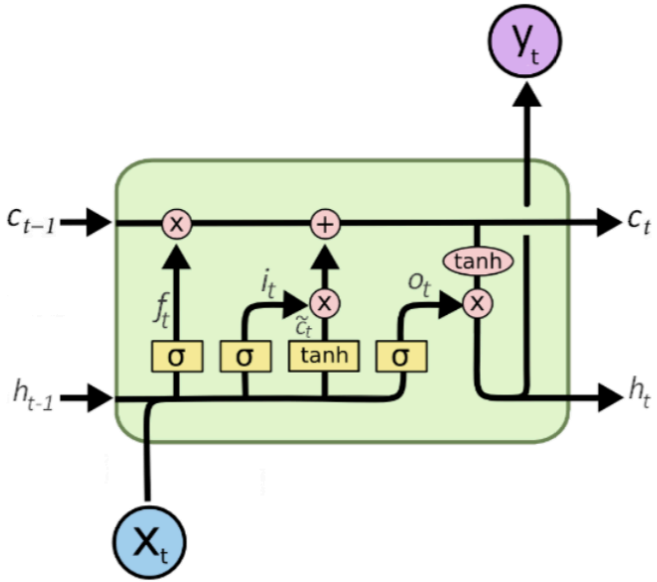


Figure 8: The LSTM unit. Adapted fro

- To tackle RNN problems of i. vanishing gradients and ii. Keep track of information for longer term.
- There is a "memory bus" Ct on which we chose how much to write and read

1. forget gate layer: $f_t = \sigma\left(U_f x_t + W_f h_{t-1} + b_f\right)$
2. input gate layer: $i_t = \sigma\left(U_i x_t + W_i h_{t-1} + b_i\right)$
3. candidate values computation layer: $\tilde{c}_t = \tanh\left(U_c x_t + W_c h_{t-1} + b_c\right)$
4. output gate layer: $o_t = \sigma\left(U_o x_t + W_o h_{t-1} + b_o\right)$

LSTM Unit



1. forget gate layer: $f_t = \sigma\left(U_f x_t + W_f h_{t-1} + b_f\right)$
2. input gate layer: $i_t = \sigma\left(U_i x_t + W_i h_{t-1} + b_i\right)$
3. candidate values computation layer: $\tilde{c}_t = \tanh\left(U_c x_t + W_c h_{t-1} + b_c\right)$
4. output gate layer: $o_t = \sigma\left(U_o x_t + W_o h_{t-1} + b_o\right)$

Assume a new training example x_t and the current hidden state h_{t-1},

- forget gate layer f_t determines how much of the previous cell state c_{t-1} should be forgotten (what fraction of the memory should be freed up),
- input gate layer decides how much of the candidate values $\tilde{c_t}$ should be written to the memory: how much of the new information should be learned. Combining the output of the two filters updates the cell state:

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

$$h_t = \tanh(c_t) \circ o_t$$

$$y_t = \text{softmax}(V h_t)$$

References (1/2)

1. Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., & Wierstra, D. (2015). DRAW: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*.
2. Hadsell, Raia, Sumit Chopra, and Yann LeCun. "Dimensionality reduction by learning an invariant mapping." *Computer vision and pattern recognition, 2006 IEEE computer society conference on*. Vol. 2. IEEE, 2006.
3. Luong, Minh-Tang, Hieu Pham, and Christopher D. Manning. "Effective approaches to attention-based neural machine translation." arXiv preprint arXiv:1508.04025 (2015).
4. Mueller, J., & Thyagarajan, A. (2016, February). Siamese Recurrent Architectures for Learning Sentence Similarity. In *AAAI* (pp. 2786-2792).
5. Course slides "Recurrent Neural Networks" Richard Socher, Stanford, 2016
6. Recurrent Neural Network Architectures Abhishek Narwekar, Anusri Pampari, University of Illinois, 2016
7. Neculoiu, Paul, Maarten Versteegh, and Mihai Rotaru. "Learning text similarity with siamese recurrent networks." *Proceedings of the 1st Workshop on Representation Learning for NLP*. 2016.
8. Rush, Alexander M., Sumit Chopra, and Jason Weston. "A neural attention model for abstractive sentence summarization." arXiv preprint arXiv:1509.00685 (2015).
9. Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.
10. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... & Bengio, Y. (2015, June). Show, attend and tell: Neural image caption generation with visual attention. In International Conference on Machine Learning (pp. 2048-2057).
11. Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016). Hierarchical attention networks for document classification. NAACL 2016 (pp. 1480-1489).

- Convolutional Neural Networks
- Recurrent NNs + LSTMs
- Other architectures (Attention, Siamese, EBL…)

Attention is ubiquitous in DL today

Image captioning

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

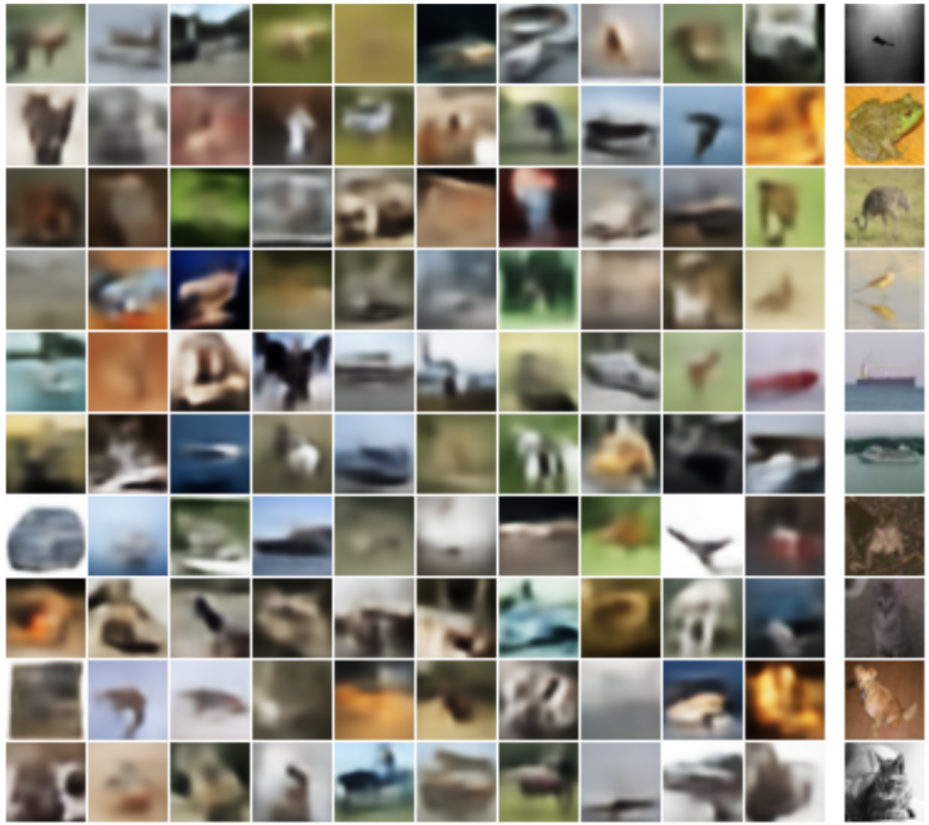
A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Show, attend and tell: Neural image caption generation with visual attention (Xu et al. 2015)

Attention is ubiquitous in DL today

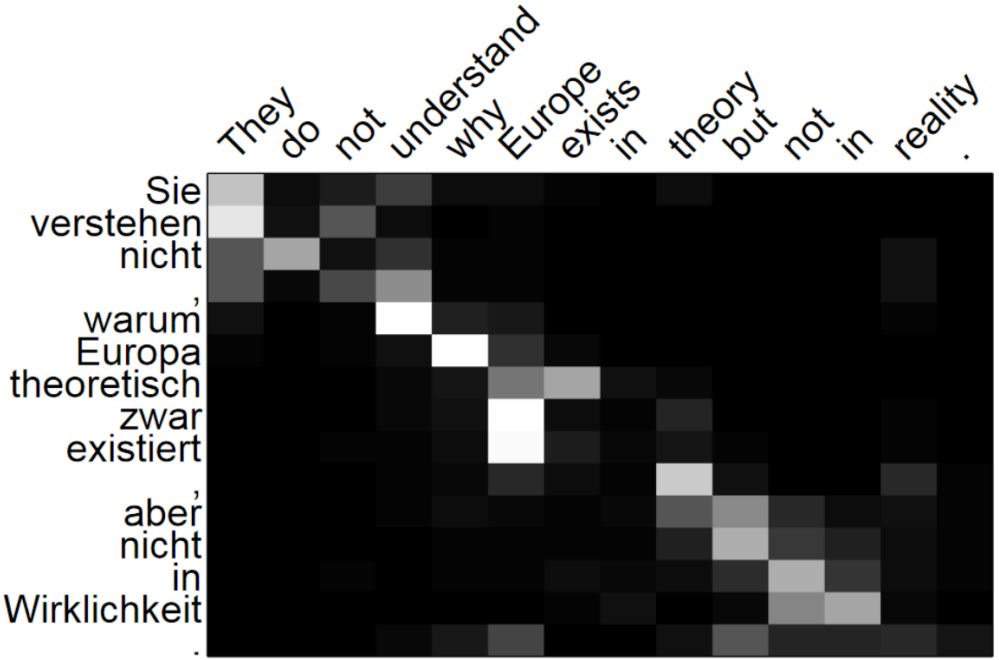
Image generation



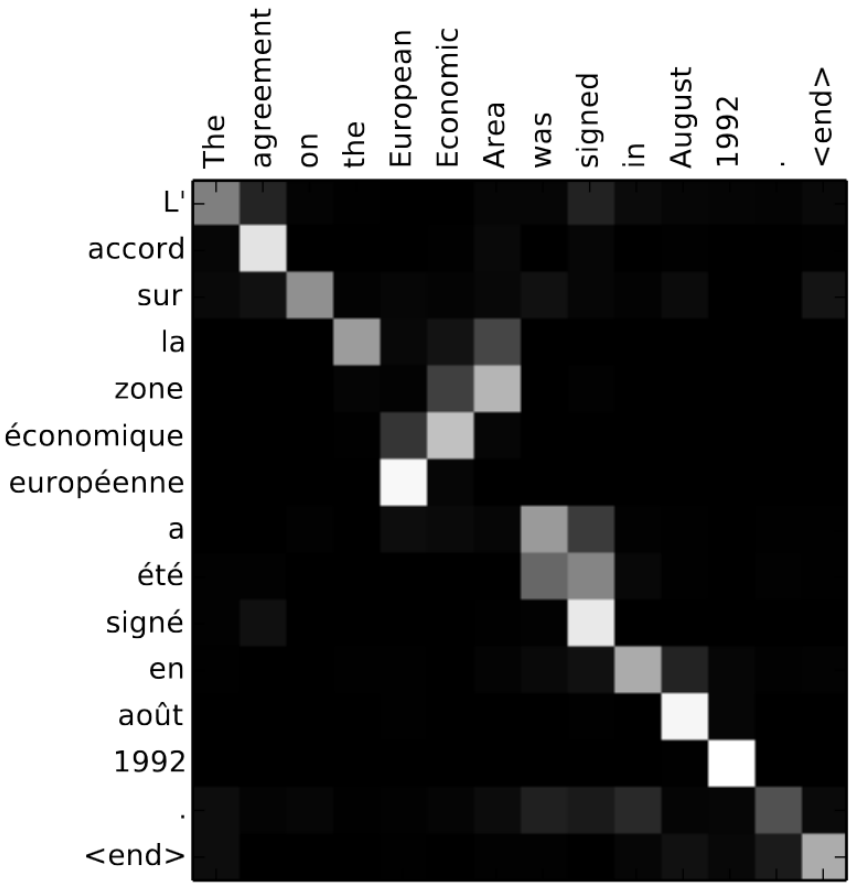
DRAW: A recurrent neural network for image generation (Gregor et al. 2015)

Attention is ubiquitous in DL today

Neural Machine Translation



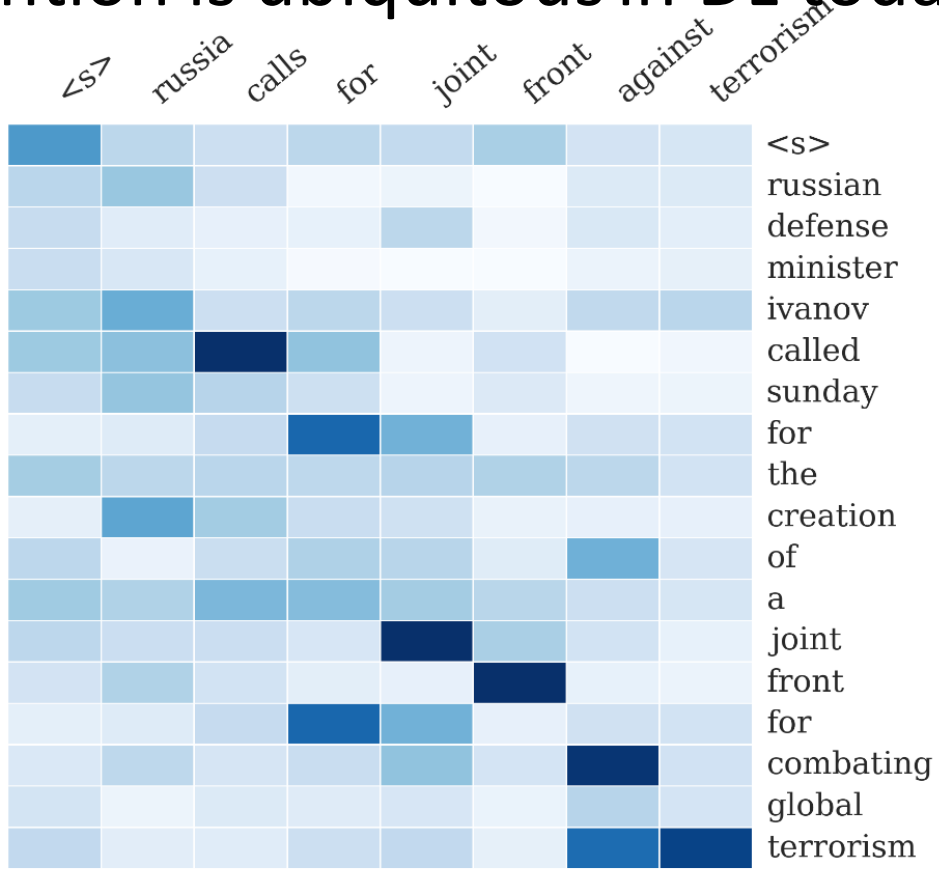
Effective approaches to attention-based neural machine translation (Luong et al. 2015)



Neural machine translation by jointly learning to align and translate (Bahdanau et al. 2014)

Attention is ubiquitous in DL today

Abstractive summarization



A neural attention model for abstractive sentence summarization (Rush et al. 2015)

Attention is ubiquitous in DL today

Sentiment analysis

GT: 4 Prediction: 4

pork belly = delicious .
scallops ?
i do n't .
even .
like .
scallops , and these were a-m-a-z-i-n-g .
fun and tasty cocktails .
next time i 'm in phoenix , i will go
back here .
highly recommend .

GT: 0 Prediction: 0

terrible value .
ordered pasta entree .
.
$ 16.95 good taste but size was
appetizer size .
.
no salad , no bread no vegetable .
this was .
our and tasty cocktails .
our second visit .
i will not go back .

Hierarchical attention networks for document classification (Yang et al. 2016)

Encoder-Decoder Architectures

General idea:

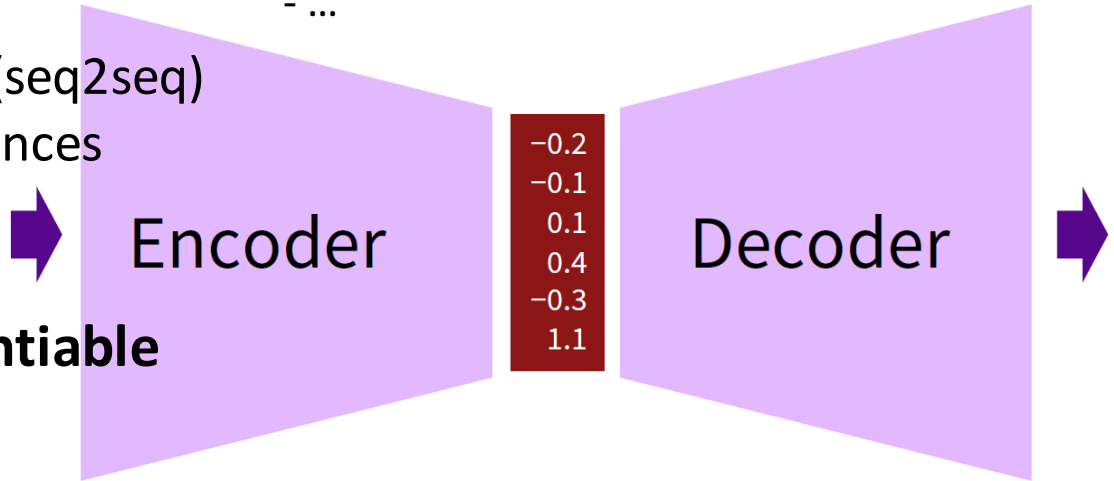
encoder **decoder**

Input -> embedding -> Output

- image
- document
- source language
- …

- caption
- summary
- target language
- …

Known as *sequence-to-sequence* (seq2seq) when input and output are sequences (e.g., NLP applications)



The full architecture is differentiable
=> end-to-end training

What is attention?

Objective

- Relieve the model (e.g., encoder) from the burden of having to embed the input into a single fixed-length vector (lossy). All information can be kept and stored into multiple vectors. The vectors from which information should be retrieved can be selected later on (e.g., by the decoder). (Bahdanau et al. 2014)

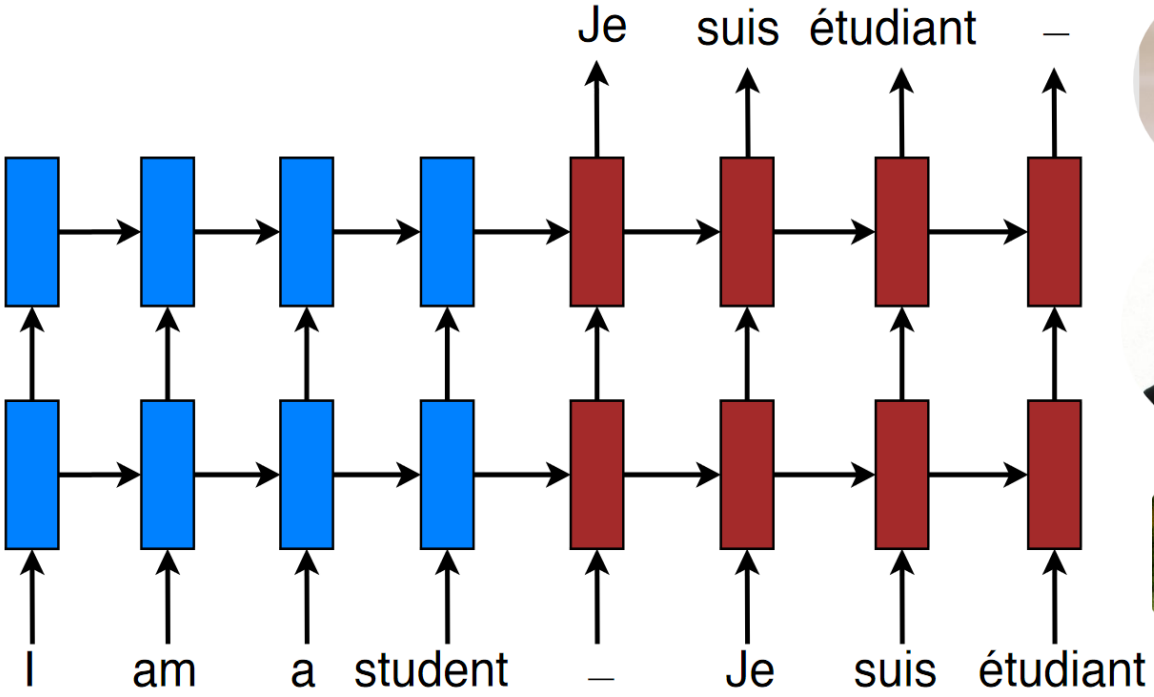
Quick history:

- developed in the context of **encoder-decoder** architectures for neural machine translation (Bahdanau et al. 2014)
- rapidly applied to naturally related tasks like image captioning (Xu et al. 2015) and summarization (*Luong et al. 2015*)
- also proposed for encoders only, e.g. for text classification (Yang et al. 2015) and representation learning (Conneau et al. 2017).
  Known as *self or inner attention* in such cases.

Encoder-Decoder for Neural Machine Translation

Encoder **Decoder**

Target *sentence (generated)*



Source *sentence (input)*

Minh-Thang Luong

Research Scientist at Google
Verified email at google.com - Homepage
Deep Learning Natural Language Processing

Hieu Pham

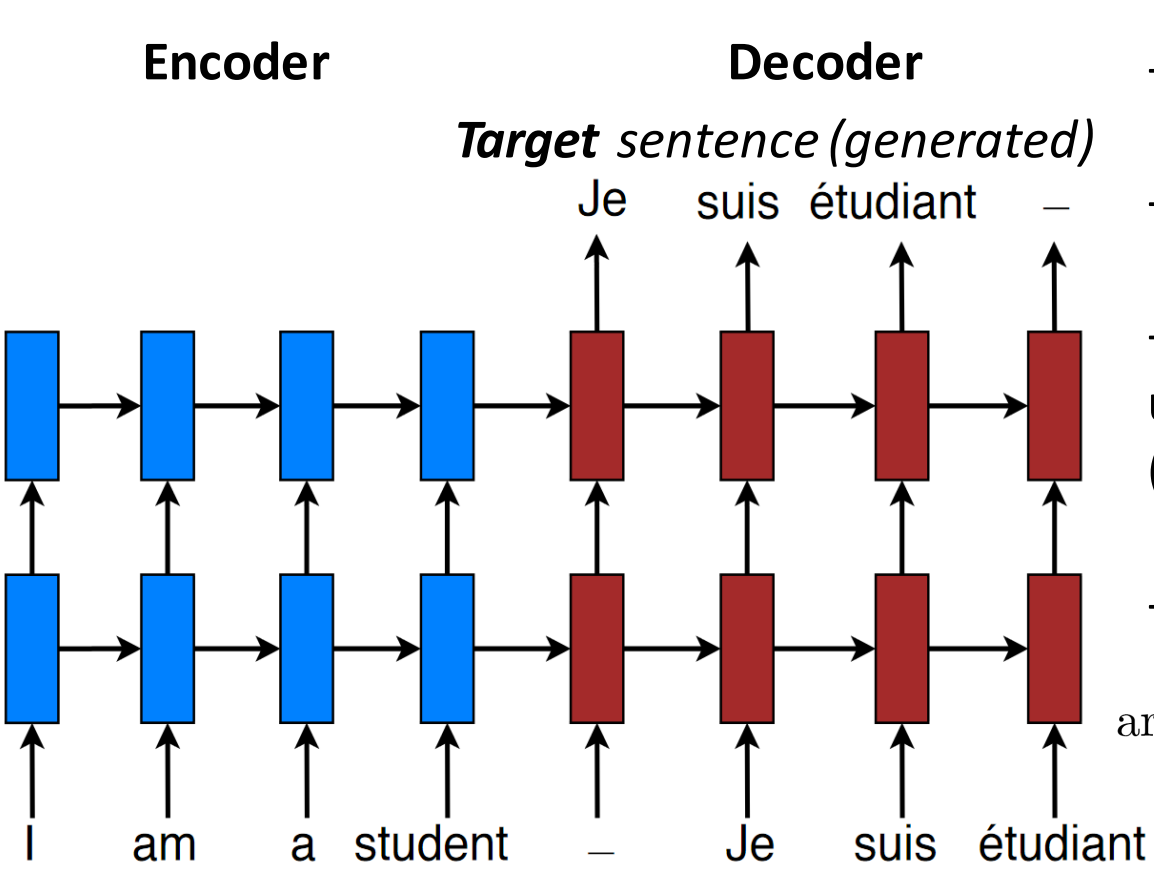
Carnegie Mellon University, Google Brain
Verified email at google.com
Machine Learning

Christopher D Manning

Professor of Computer Science and Linguistics
Verified email at stanford.edu - Homepage
Natural Language Processing

Effective approaches to Attention-Based Neural Machine Translation (2015)

Encoder-Decoder for Neural Machine Translation



- source: $\left(x_1, \ldots, x_{T_x}\right)$

- target: $\left(y_1, \ldots, y_{T_y}\right)$

- Both encoder & decoder are unidirectional deep RNNs (a.k.a. *stacking* RNNs)

- Training objective:

$$\text{argmax}_\theta \left\{ \sum_{(x,y)\in \text{corpus}} \log\, p(y|x;\theta) \right\}$$

Luong et al. (2015)

Encoder-Decoder for Neural Machine Translation

Encoder: usually: CNN, stacking RNN* with LSTM or GRU units…

* unidirectional (Luong et al. 2015) or bidirectional (Bahdanau et al. 2014).

Decoder: unidirectional *RNN* (well suited to text generation), best if deep.

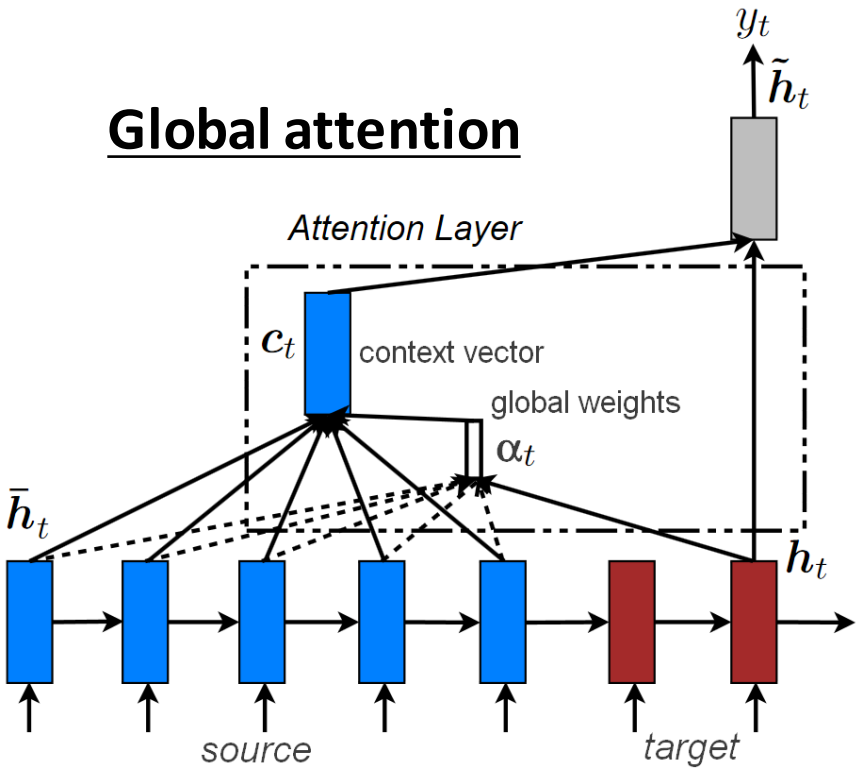
Generates the target sentence $\left(y_1, \cdots, y_{T_y}\right)$ one word at a time:

$$P\left[y_t | \{y_1, \ldots, y_{t-1}\}, c_t\right] = \text{softmax}\left(W_s \tilde{h}_t\right)$$

Luong et al. (2015)

Encoder-**Decoder** for Neural Machine Translation

Global attention



$$P\left[y_t|\{y_1,...,y_{t-1}\},c_t\right] = \text{softmax}\left(W_s\tilde{h}_t\right)$$

attentional hidden state

$$\tilde{h}_t = \tanh\left(W_c\left[c_t;h_t\right]\right)$$

context vector

decoder hidden state

$$c_t = \sum_{i=1}^{T_x} \alpha_{t,i}\bar{h}_i$$

i^{th} encoder hidden state

alignment vector

$$\alpha_{t,i} = \frac{\exp\left(\text{score}\left(h_t,\bar{h}_i\right)\right)}{\sum_{i'=1}^{T_x}\exp\left(\text{score}\left(h_t,\bar{h}_{i'}\right)\right)}$$

$$\text{score}\left(h_t,\bar{h}_i\right) = h_t^\top\bar{h}_i$$

Luong et al. (2015)

Encoder-**Decoder** for Neural Machine Translation

$$P\left[y_t|\{y_1,...,y_{t-1}\},c_t\right] = \text{softmax}\left(W_s\tilde{h}_t\right)$$

attentional hidden state

$$\tilde{h}_t = \tanh\left(W_c\left[c_t;h_t\right]\right)$$

decoder hidden state

$$p_t = T_x \cdot \sigma\left(v_p^\top \tanh\left(W_p h_t\right)\right)$$

context vector

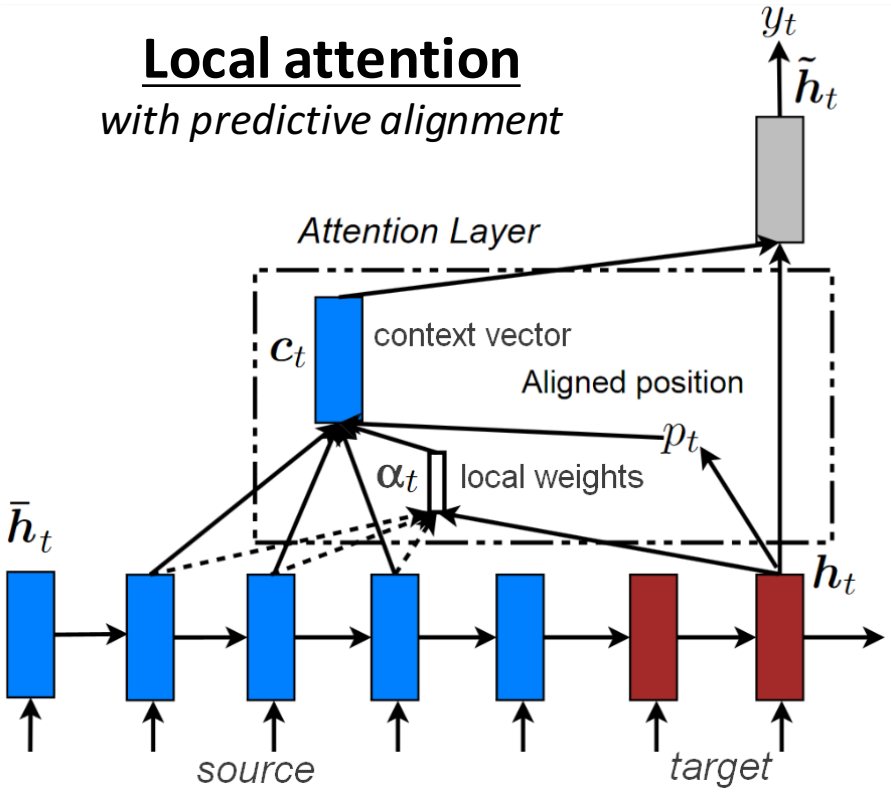
ith encoder hidden state

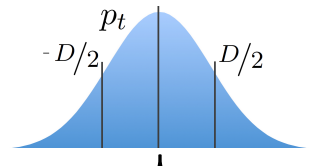
$$c_t = \sum_{i=p_t-D}^{p_t+D} \alpha_{t,i}\bar{h}_i$$

alignment vector

$$\alpha_{t,i} = \frac{\exp\left(\text{score}(h_t,\bar{h}_i)\right)}{\sum_{i'=p_t-D}^{p_t+D}\exp\left(\text{score}(h_t,\bar{h}_{i'})\right)}\exp\left(-\frac{(i-p_t)^2}{2(D/2)^2}\right)$$

$$\text{score}(h_t,\bar{h}_i) = h_t^\top W_\alpha \bar{h}_i$$





Luong et al. (2015)

Encoder-Decoder for Neural Machine Translation

Facts:

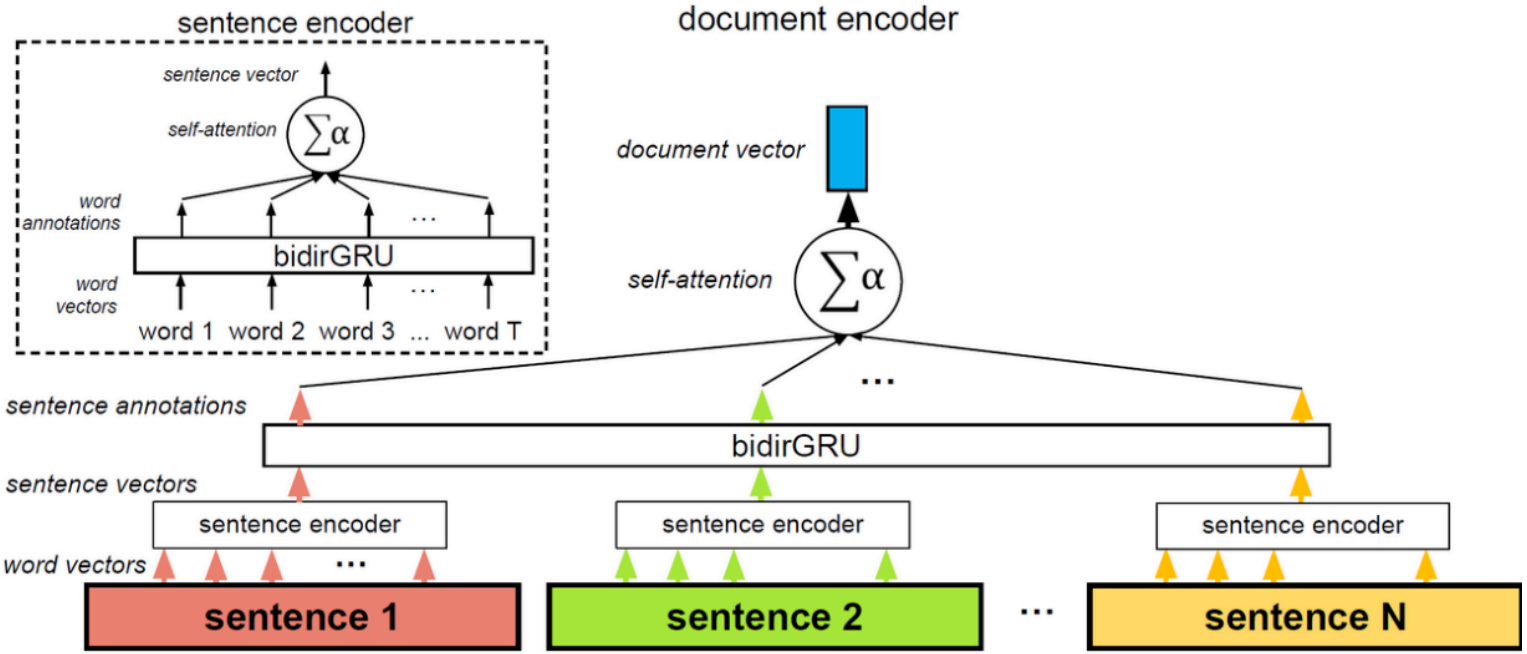
- Tested on the English <-> German task (WMT'14 dataset)
- 4.5M sentence pairs
- Encoder and decoder RNNs feature 4 layers of stacking and 1000-dimensional hidden states
- Window of size *D=10* for local attention
- Trained for 12 epochs (total of 7-10 days on a single GPU, at 1K target words/s)
- New state-of-the-art performance

Lessons learned:

- Local attention with predictive alignment gives better results than global attention
- Dot product ($\text{score}(h_t, \bar{h}_i) = h_t^\top \bar{h}_i$) works well for global attention
- The general formulation ($\text{score}(h_t, \bar{h}_i) = h_t^\top W_\alpha \bar{h}_i$) is better for local attention

Luong et al. (2015)

Self-attention for RNN encoders



Self-attention for RNN encoders

- Input is a sentence $(x_1, \ldots, x_T)$
- We're only interested in getting an embedding s of the sentence for some downstream task (e.g., classification)

encoder hidden state

context vector

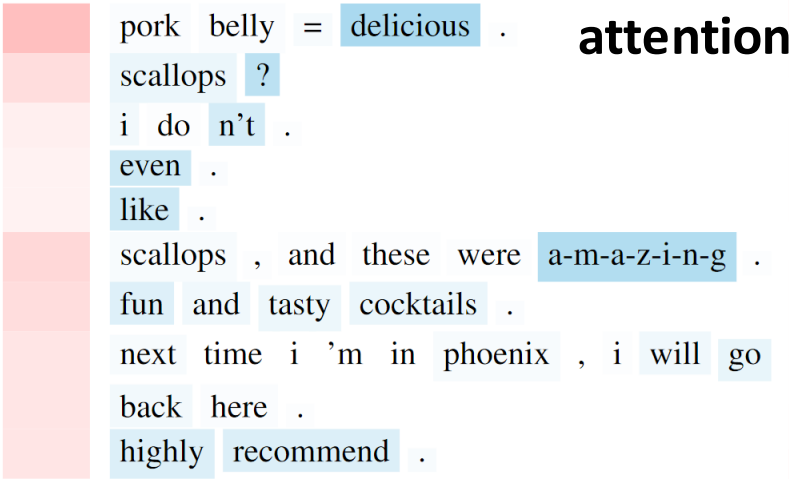
$$u_t = \tanh(W h_t)$$

$$\alpha_t = \frac{\exp(\text{score}(u_t, u))}{\sum_{t'=1}^{T} \exp(\text{score}(u_{t'}, u))}$$

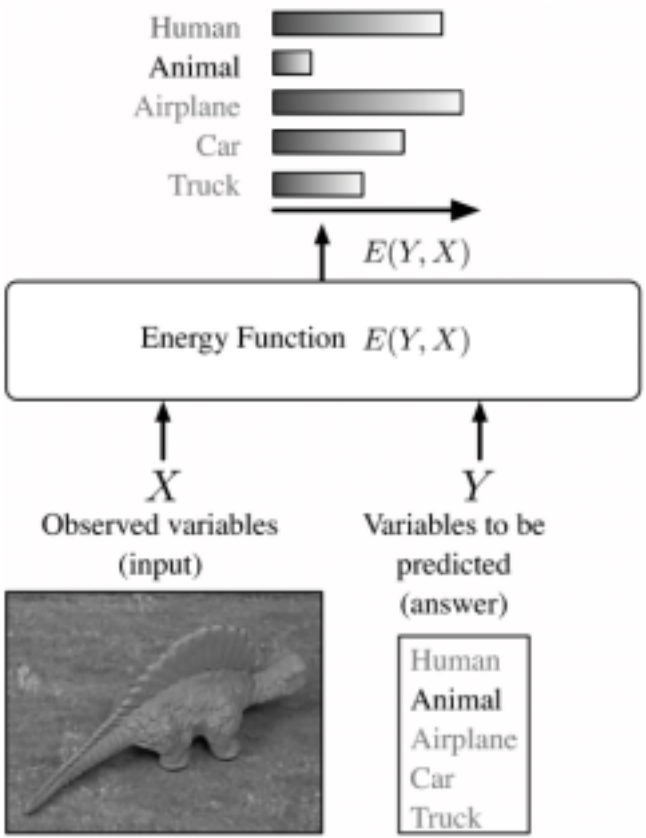
$$s = \sum_{t=1}^{T} \alpha_t h_t$$

Where $\text{score}(u_t, u) = u_t^\top u$

The same process can be repeated over the sentence vectors s -> **hierarchical attention**



Energy-Based Models - EBM for classification



Model: Measures the compatibility between an observed variable X and a variable to be predicted Y through an energy function E(Y,X).

E(Y,X) assigns low energies to correct configurations of X and Y and higher energies to incorrect ones.

Inference process: Search for the Y that minimizes the energy function for a given X. In case of low cardinality, we can use exhaustive search.

$$Y^* = \text{argmin}_{Y \in \mathcal{Y}} E(Y, X).$$

A tutorial on energy-based learning (LeCun et al. 2006)

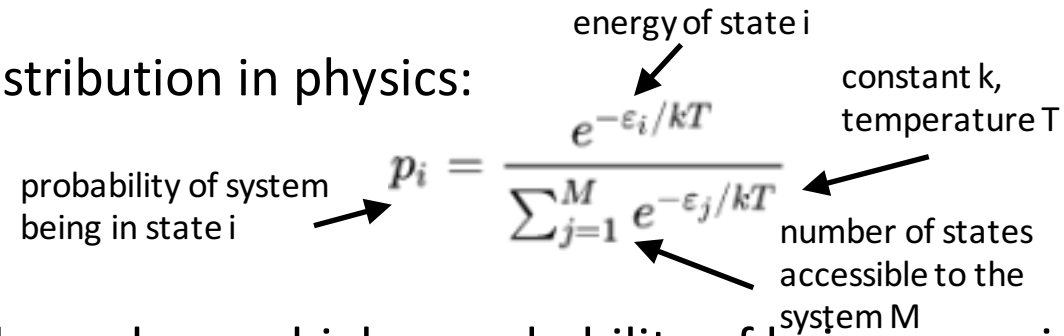
Energy-Based Models （EBMs）

- EBMs associate a scalar energy to each configuration of the variables of interest. Probability distribution is defined through an energy function:

$$P(\mathbf{x}) = \frac{e^{-Energy(\mathbf{x})}}{\sum_{\mathbf{x}} e^{-Energy(\mathbf{x})}} \qquad P(y|\mathbf{x}) = \frac{e^{-Energy(\mathbf{x},y)}}{\sum_{y} e^{-Energy(\mathbf{x},y)}}$$

Learning corresponds to modifying the energy function so that its shape has desirable properties.

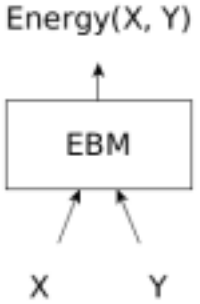
- Related to the Boltzmann distribution in physics:



States with lower energy will always have a higher probability of being occupied than the states with higher energy.

Learning deep architectures for AI (Bengio. 2009)

What Questions Can an EBM Answer?



1. Prediction, Classification & Decision Making:

- Which value of Y is most compatible with X?
- Training: give the lowest energy to the correct answer

2. Ranking:

- Is Y1 or Y2 more compatible with this X?
- Training: produce energies that rank the answers correctly

3. Detection:

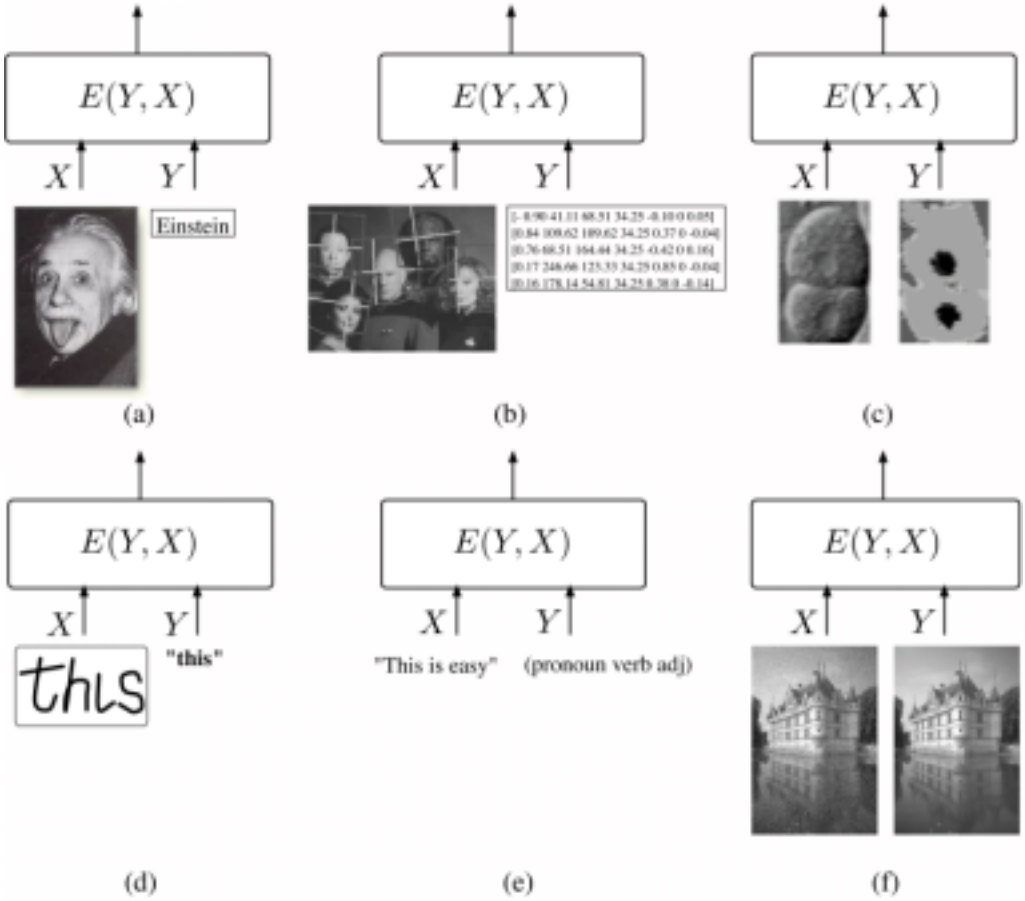
- Is this value of Y compatible with X? (e.g. face detection)
- Training: energies that increase as the image looks less like a face.

4. Conditional Density Estimation:

- What is the conditional distribution P(Y|X)?
- Training: differences of energies must be just so.

A tutorial on energy-based learning (LeCun et al. 2006)

Complex Tasks: Inference is non-trivial



- When the cardinality or dimension of Y is large, exhaustive search is impractical.
- We need to use a "smart" *inference procedure* to find the Y that (globally or locally) minimizes the energy function E(Y, X), such as gradient-based optimization, min-sum, Viterbi algorithms, etc. (depends on the internal structure of the model.)

A tutorial on energy-based learning (LeCun et al. 2006)

EBM Architecture, Training, Inference

- Parameterized energy functions $\mathcal{E} = \{E(W, Y, X) : \ W \in \mathcal{W}\}.$
- Training set $\mathcal{S} = \{(X^i, Y^i) : i = 1 \ldots P\}$
- Training process $W^* = \min_{W \in \mathcal{W}} \mathcal{L}(E, \mathcal{S})$
- Loss functional (*measures the quality of an energy function using the S)*

$$\mathcal{L}(E, \mathcal{S}) = \frac{1}{P} \sum_{i=1}^{P} L(Y^i, E(W, \mathcal{Y}, X^i)) + R(W).$$

Per-sample loss

Desired answer

Energy surface for a given Xi as Y varies

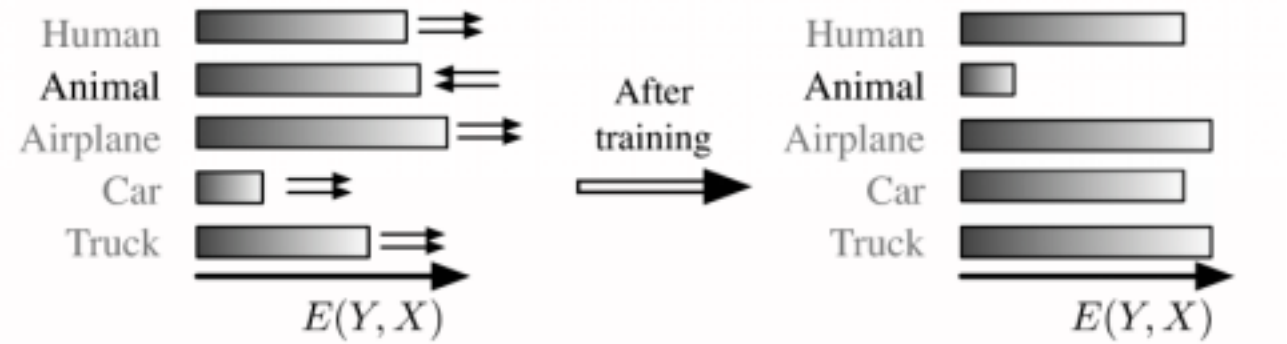
Regularizer

- Inference process

$$Y^* = \text{argmin}_{Y \in \mathcal{Y}} E(Y, X).$$

A tutorial on energy-based learning (LeCun et al. 2006)

Designing a Loss Functional



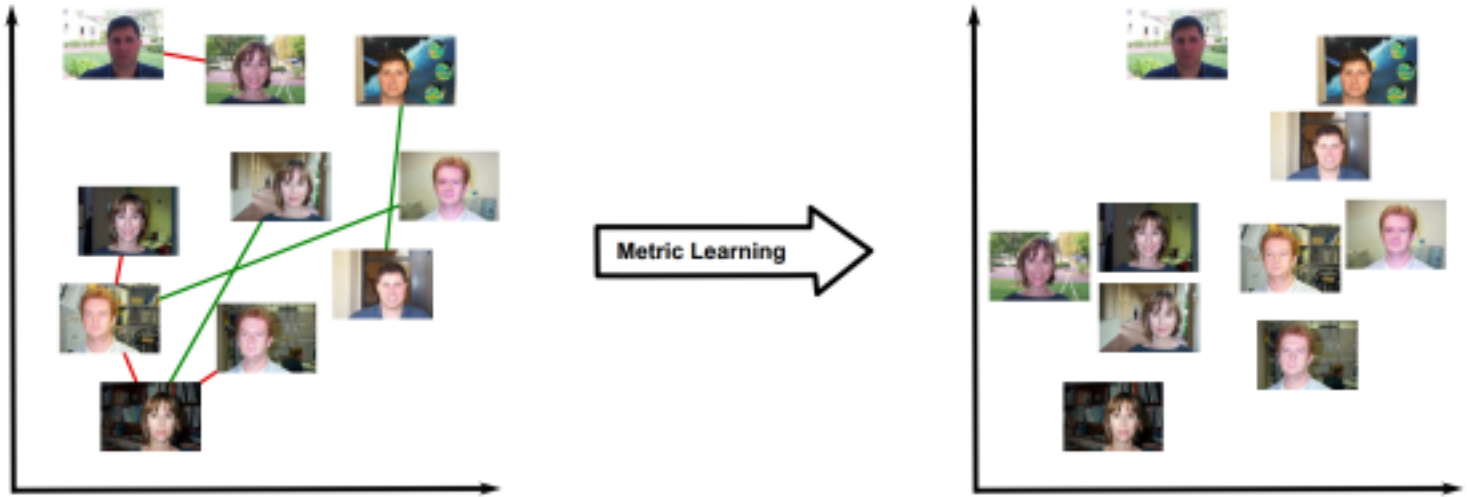
- Correct answer has the lowest energy -> LOW LOSS
- Lowest energy is not for the correct answer -> HIGH LOSS

A tutorial on energy-based learning (LeCun et al. 2006)

Deep Metric Learning

- Metrics are ubiquitous in machine learning, measuring the distance (or similarity) between objects. Each problem has its own semantic notion of similarity, which is often badly captured by standard metrics (e.g., Euclidean distance).

- Learn a **parameterized** distance function $D_w(x, x')$ that assigns small (resp. large) distance to pairs of examples that are *semantically* similar (resp. dissimilar).

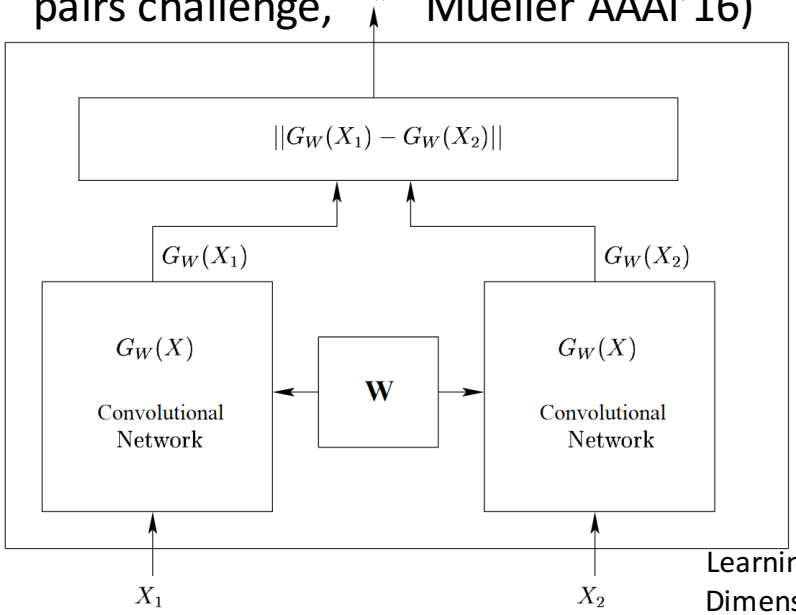


A survey on metric learning for feature vectors and structured data (Bellet. 2013)

Siamese architecture

Overview

- Two encoders **learn the similarity** between training examples passed as pairs
- Weights are *shared* between branches
- Developed in Computer Vision
- Application to learning text similarity (e.g., winners of the 2017 Kaggle Quora question pairs challenge, Mueller AAAI'16)



Sumit Chopra

Imagen Technologies
Verified email at imagen.ai - Homepage

Machine Learning Energy Based Models Deep Learning

Raia Hadsell

Google DeepMind
Verified email at google.com - Homepage

Artificial Intelligence Deep Learning Robotics

Yann LeCun

Chief AI Scientist at Facebook & Silver Professor at the Courant Institute New York University
Verified email at cs.nyu.edu - Homepage

AI machine learning computer vision robotics image compression

genuine (similar)

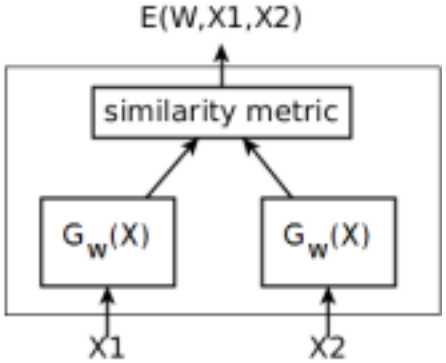
impostor (dissimilar)

Learning a similarity metric discriminatively, with application to face verification (2005)
Dimensionality reduction by learning an invariant mapping (2005)

Deep Metric Learning: Siamese network

- An architecture of EBMs, where variables X1 and X2 are passed through two instances of a same function G_w (dual-branch networks with tied weights).
- Energy function E(W, X1, X2) is defined as *symmetric* similarity metric between G_w(X1) and G_w(X2).
- Training set consists of labeled pairs, when X1 and X2 are similar (e.g. two pictures of the same person), Y = 0; when they are different, Y = 1.
- Training process involves finding an optimal W that assigns low energies to similar pairs and higher energies to dissimilar pairs. (in case of L1 distance)

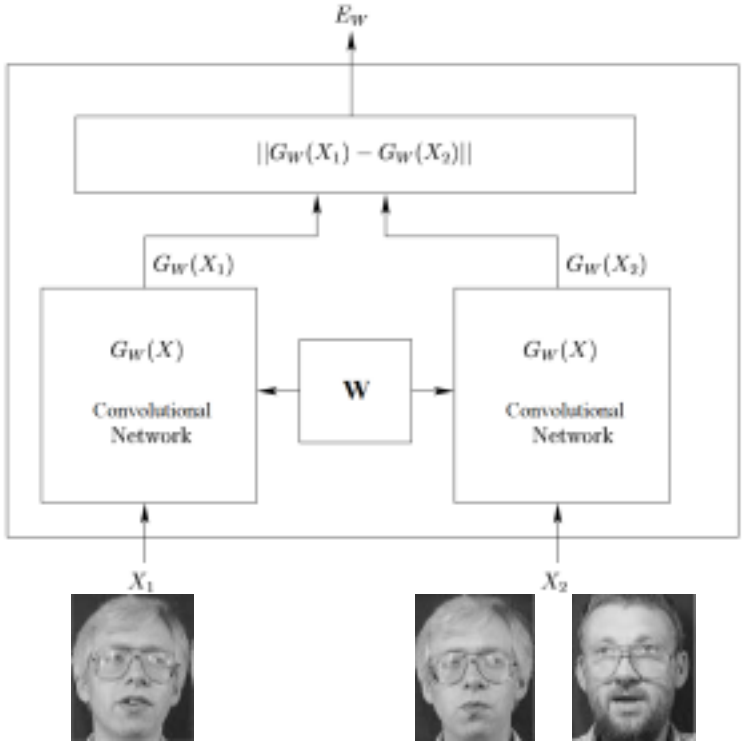
Originally designed for signature verification (Bromley et al., 1993)



Learning a similarity metric discriminatively, with application to face verification (Chopra et al. 2005)

Application in face verification (1/2)

(energy should be low given the *similar* pair, be high given the *dissimilar* pair)



Contrastive loss

An *energy function* E_W computes the L1 distance between the two embeddings $G_W(X_1)$ and $G_W(X_2)$

Images are separately encoded by a shared CNN

Input: pair of images (X_1, X_2) and label Y:

Y = 0 if X_1 and X_2 are similar

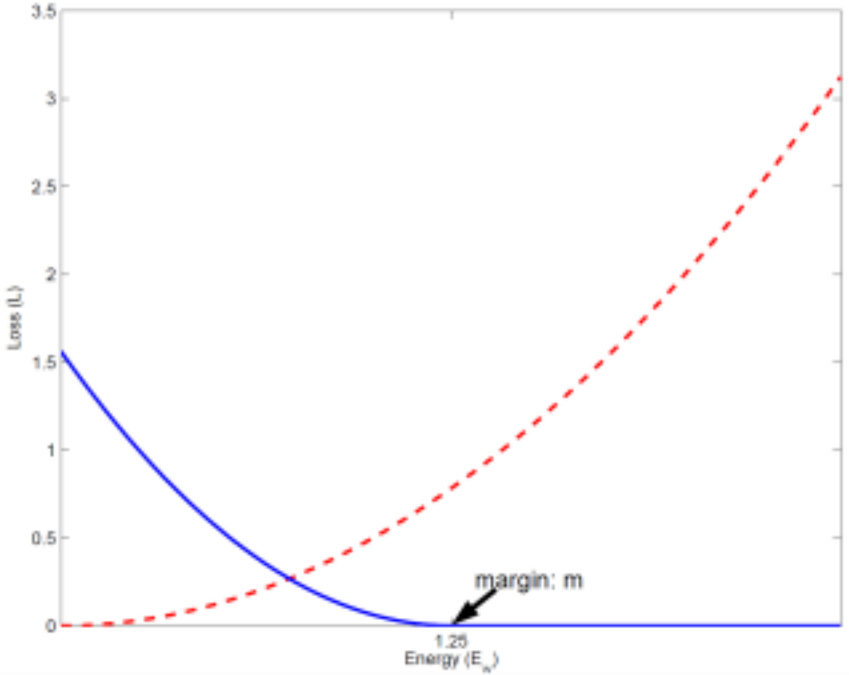
Y = 1 if they are dissimilar

Learning a similarity metric discriminatively, with application to face verification (Chopra et al. 2005)

Application in face verification (2/2)

Contrastive loss:

$$L(W,(Y,X_1,X_2)^i) = (1-Y)L_S\left(E_W^i\right) + YL_D\left(E_W^i\right)$$



L_S: partial loss for similar instances (Y=0)

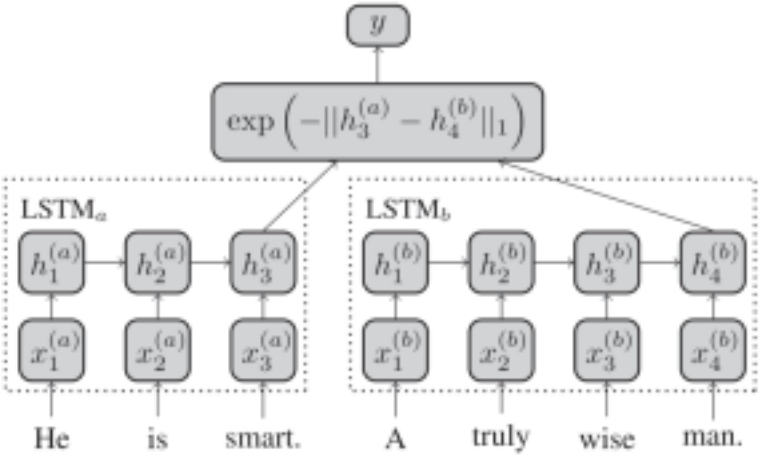
L_D: partial loss for dissimilar instances (Y=1)

- Goal: optimize W so that E_W is **low for similar pairs** and **high for dissimilar pairs**
- attraction/repulsion spring analogy

Learning a similarity metric discriminatively, with application to face verification (Chopra et al. 2005)

Applications in NLP (1/2)

Learning similarity metric between two sentences. (winners of the 2017 Kaggle Quora question pairs challenge)



Mean squared error loss

An *energy function* E_W computes the exponent negative manhattan distance between the two embeddings $G_W(X_1)$ and $G_W(X_2)$

Sentence are separately encoded by a shared LSTM

Input: pair of sentences (X_1, X_2) (represented as a sequence of word vectors) and label Y:

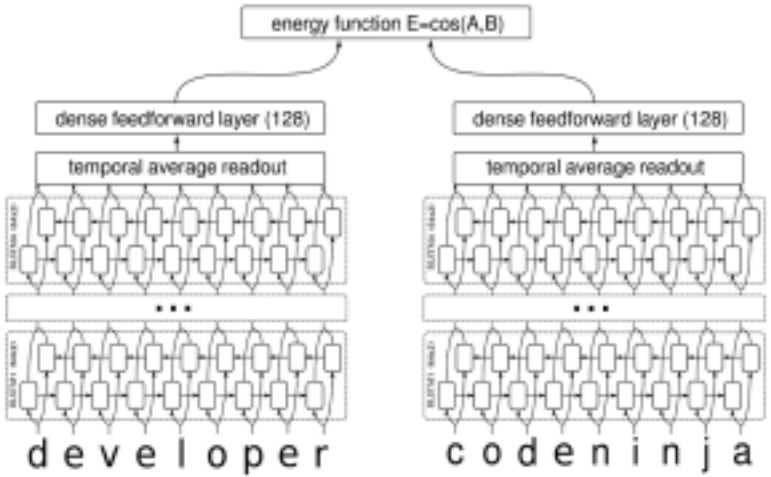
Y = 1 if X_1 and X_2 are similar

Y = 0 if they are dissimilar

Siamese Recurrent Architectures for Learning Sentence Similarity (Mueller and Thyagarajan. 2016)

Applications in NLP (1/2)

Learning similarity metric between two sequences of characters (job titles).



Contrastive loss

An *energy function* E_W computes the cosine similarity between the two embeddings $G_W(X_1)$ and $G_W(X_2)$

Words are separately encoded by shared four layers of Bidirectional LSTM nodes and one densely connected feedforward layer.

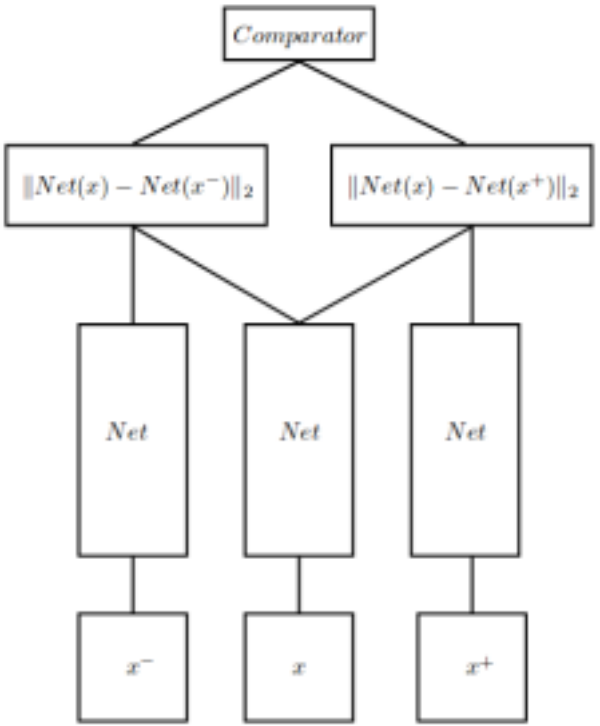
Input: pair of words (X_1, X_2) and label Y:

$Y = 1$ if X_1 and X_2 are similar

$Y = 0$ if they are dissimilar

Learning text similarity with siamese recurrent networks (Neculoiu et al. 2016)

Deep Metric Learning: Triplet network



A Triplet network is comprised of 3 instances of the same *Net(x)* (with shared parameters).

Training is performed by feeding the network with triplets (x, x^+, x^-) where x and x^+ are of the same class, and x^- is of different class.

Intermediate values consist of distances between each of x^+ and x^- against the reference x.

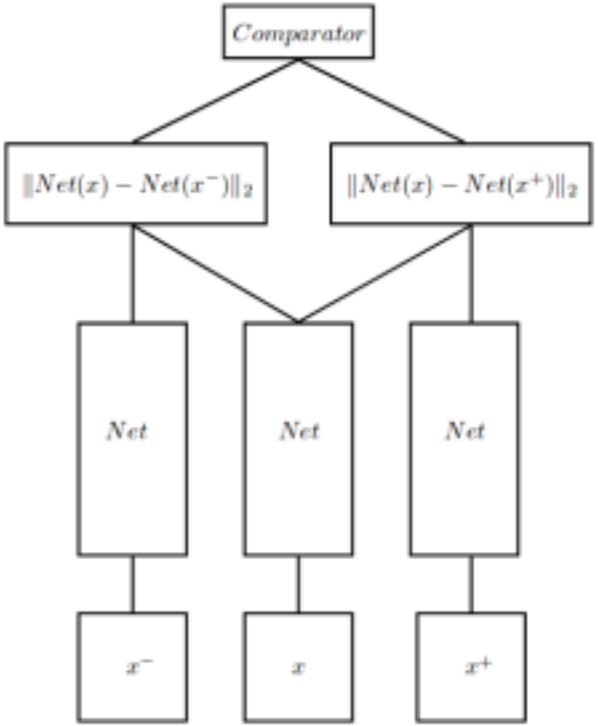
$$TripletNet(x, x^-, x^+) = \begin{bmatrix} \|Net(x) - Net(x^-)\|_2 \\ \|Net(x) - Net(x^+)\|_2 \end{bmatrix} \in \mathbb{R}^2_+$$

Siamese or Triplet ? Depending on data, training strategies, network design, ...

Deep metric learning using triplet network (Hoffer and Ailon. 2005)

Deep Metric Learning: Triplet network

The objective is to learn a metric embedding, the label determines which example is *closer* to x.



Loss function is MSE on the softmax result, compared to the (0, 1) vector:

$$Loss(d_+, d_-) = \|(d_+, d_- - 1)\|_2^2 = const \cdot d_+^2$$

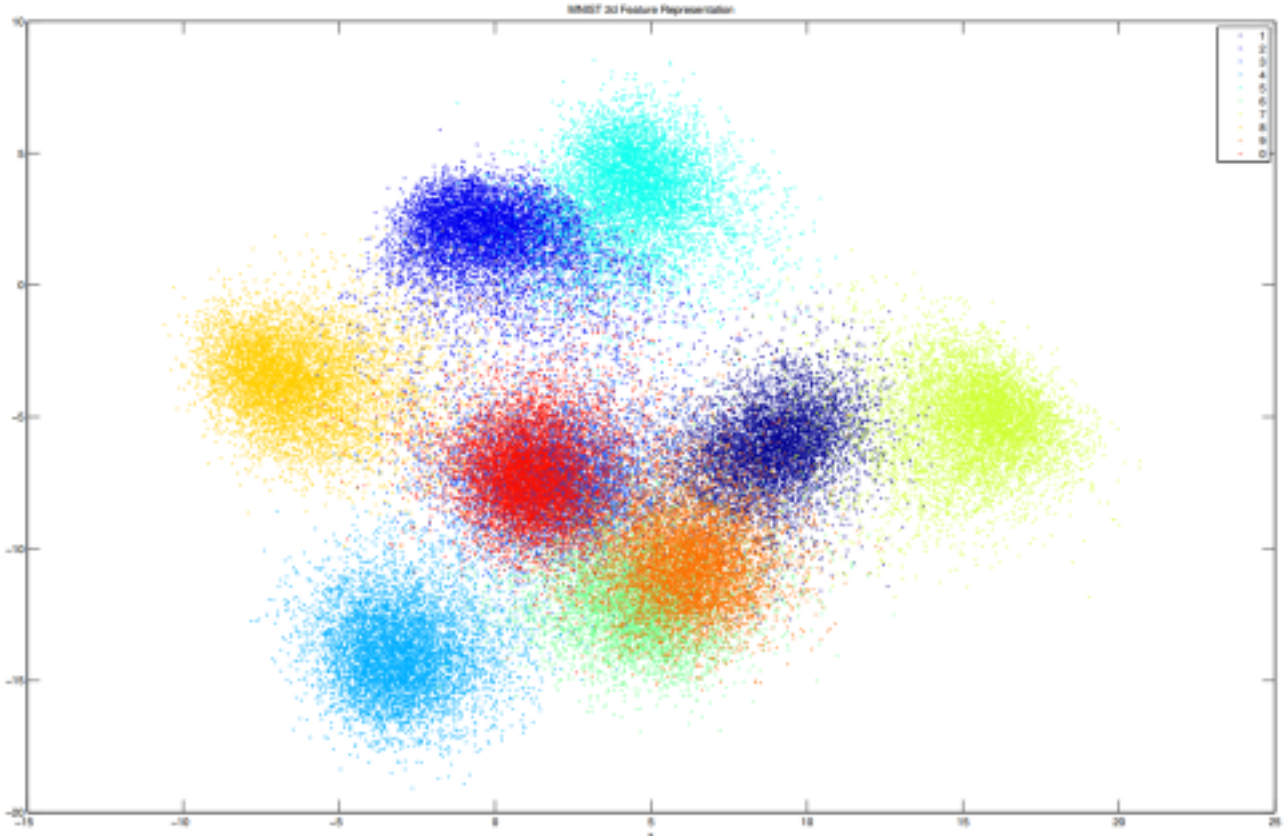
$$d_+ = \frac{e^{\|Net(x)-Net(x^+)\|_2}}{e^{\|Net(x)-Net(x^+)\|_2} + e^{\|Net(x)-Net(x^-)\|_2}}$$

$$d_- = \frac{e^{\|Net(x)-Net(x^-)\|_2}}{e^{\|Net(x)-Net(x^+)\|_2} + e^{\|Net(x)-Net(x^-)\|_2}}$$

FaceNet: A Unified Embedding for Face Recognition and Clustering (Schroff et al.. 2015)

Deep metric learning using triplet network (Hoffer and Ailon. 2015)

Application on MNIST dataset



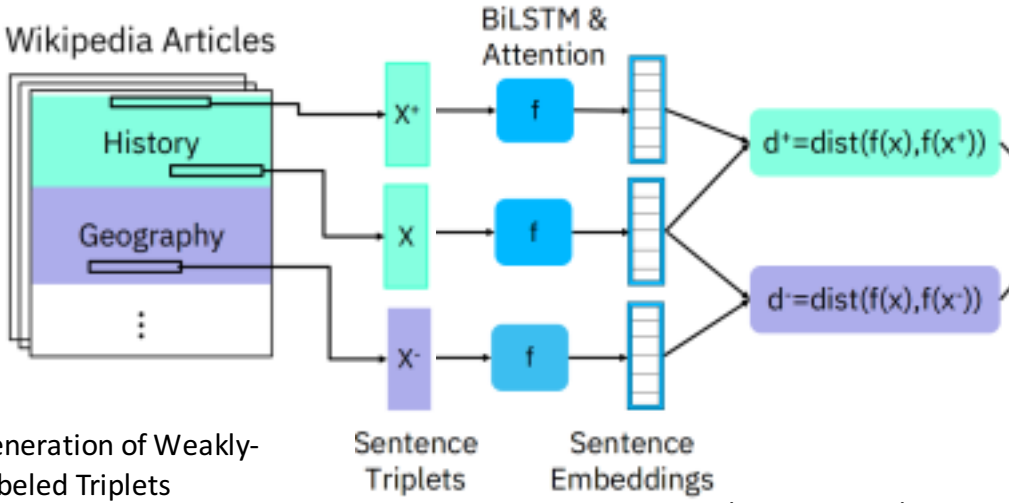
MNIST - Euclidean representation of embedded test data, projected onto top two singular vectors

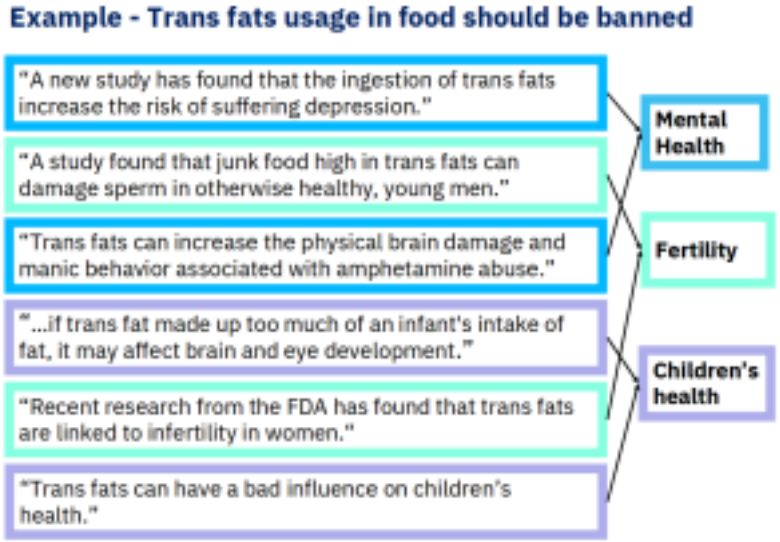
Deep metric learning using triplet network (Hoffer and Ailon. 2015)

Application in NLP

Argumentation mining:
Organize arguments into themes to generate a compelling argumentative narrative.

Multi-document summarization:
Organize selected sentences into meaningful sections and paragraphs.





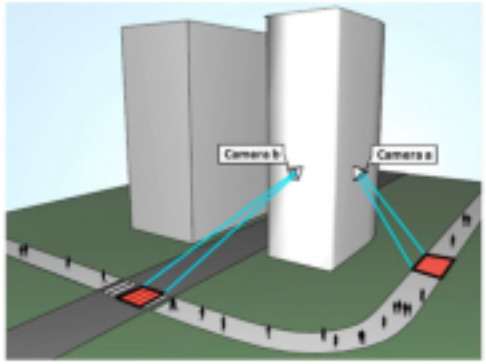
Learning a similarity metric between sentences dedicated to thematic clustering

Learning Thematic Similarity Metric from Article Sections Using Triplet Networks (Dor et al. 2018)

Deep Metric Learning: Quadruplet network

Probe Rank List
(groundtruth in green frames)

Results of triplet loss on Testing Set



Observation: The trained model (on the left figure) with triplet loss has low generalization ability on testing set.

- Some false positives are more similar to the probe image.
- These false positives never show up in training set.

Beyond triplet loss: a deep quadruplet network for person re-identification (Chen et al. 2017)

Deep Metric Learning: Quadruplet network

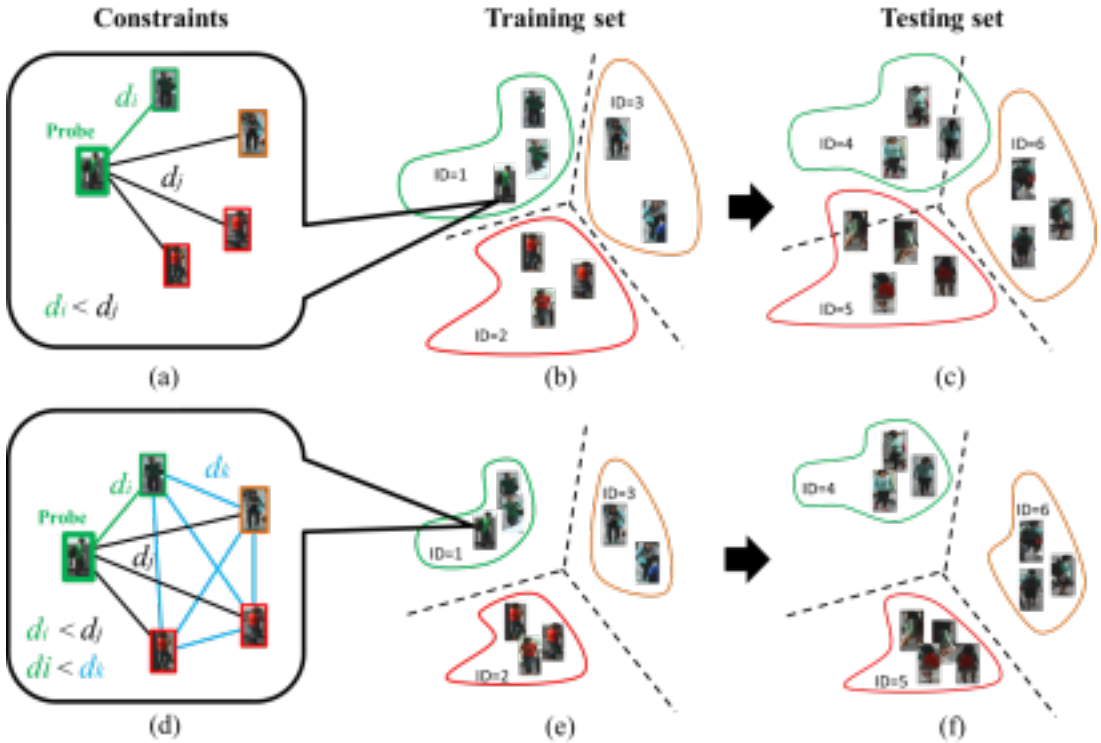
$$L_{trp} = \sum_{i,j,k}^{N} [g(x_i, x_j)^2 - g(x_i, x_k)^2 + \alpha_{trp}]_+$$

g(xi, xj) is leaned metric
[z]+ = max(z, 0)
The threshold α is a margin that is enforced between positive and negative pairs

$$L_{quad} = \sum_{i,j,k}^{N} [g(x_i, x_j)^2 - g(x_i, x_k)^2 + \alpha_1]_+ + \sum_{i,j,k,l}^{N} [g(x_i, x_j)^2 - g(x_l, x_k)^2 + \alpha_2]_+$$

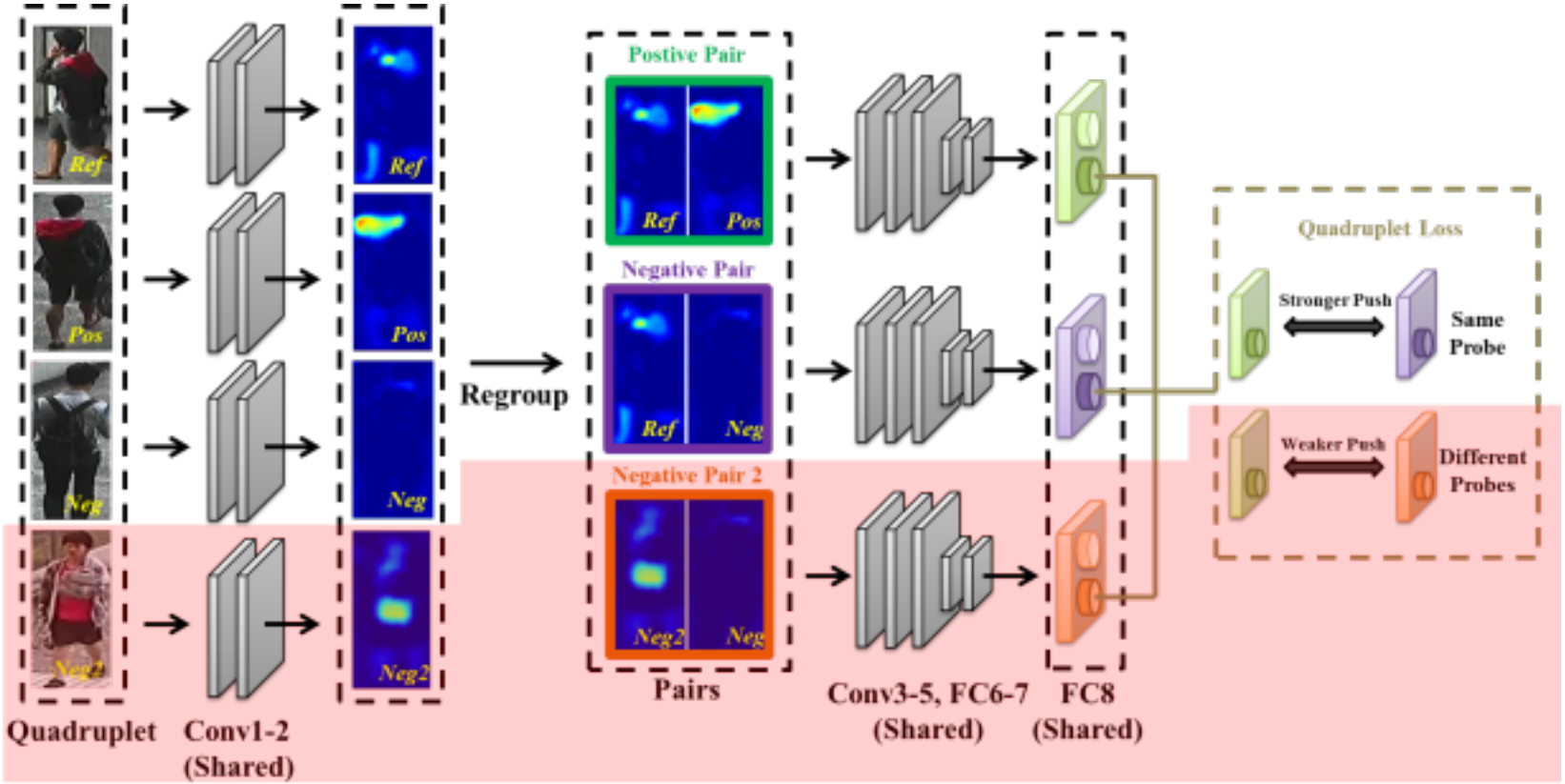
$$s_i = s_j, s_l \neq s_k, s_i \neq s_l, s_i \neq s_k$$

where α1 and α2 are the values of margins in two terms and si refers to the person ID of image xi



Beyond triplet loss: a deep quadruplet network for person re-identification (Chen et al. 2017)

Deep Metric Learning: Quadruplet network



Beyond triplet loss: a deep quadruplet network for person re-identification (Chen et al. 2017)

References (1/3)

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., & Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. arXiv preprint arXiv:1705.02364.

Chopra, Sumit, Raia Hadsell, and Yann LeCun. "Learning a similarity metric discriminatively, with application to face verification." *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE, 2005.

Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., & Wierstra, D. (2015). DRAW: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*.

Hadsell, Raia, Sumit Chopra, and Yann LeCun. "Dimensionality reduction by learning an invariant mapping." *Computer vision and pattern recognition, 2006 IEEE computer society conference on*. Vol. 2. IEEE, 2006.

Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning. "Effective approaches to attention-based neural machine translation." arXiv preprint arXiv:1508.04025 (2015).

Mueller, J., & Thyagarajan, A. (2016, February). Siamese Recurrent Architectures for Learning Sentence Similarity. In *AAAI* (pp. 2786-2792).

References (2/3)

Neculoiu, Paul, Maarten Versteegh, and Mihai Rotaru. "Learning text similarity with siamese recurrent networks." *Proceedings of the 1st Workshop on Representation Learning for NLP*. 2016.

Rush, Alexander M., Sumit Chopra, and Jason Weston. "A neural attention model for abstractive sentence summarization." arXiv preprint arXiv:1509.00685 (2015).

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... & Bengio, Y. (2015, June). Show, attend and tell: Neural image caption generation with visual attention. In International Conference on Machine Learning (pp. 2048-2057).

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016). Hierarchical attention networks for document classification. NAACL 2016 (pp. 1480-1489).

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., & Huang, F. (2006). A tutorial on energy-based learning. *Predicting structured data*, *1*(0).

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, *2*(1), 1-127.

References (3/3)

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., & Shah, R. (1994). Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems* (pp. 737-744).

Neculoiu, P., Versteegh, M., & Rotaru, M. (2016). Learning text similarity with siamese recurrent networks. In *Proceedings of the 1st Workshop on Representation Learning for NLP* (pp. 148-157).

Hoffer, E., & Ailon, N. (2015, October). Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition* (pp. 84-92). Springer, Cham.

Schroff, F., Kalenichenko, D., & Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 815-823).

Dor, L. E., Mass, Y., Halfon, A., Venezian, E., Shnayderman, I., Aharonov, R., & Slonim, N. (2018). Learning Thematic Similarity Metric from Article Sections Using Triplet Networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (Vol. 2, pp. 49-54).

Bellet, A., Habrard, A., & Sebban, M. (2013). A survey on metric learning for feature vectors and structured data.

Chen, W., Chen, X., Zhang, J., & Huang, K. (2017, July). Beyond triplet loss: a deep quadruplet network for person re-identification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Vol. 2, No. 8).