Deep Learning

Shallow vs Deep

Understanding Machine Learning

Recap: Goal of ML

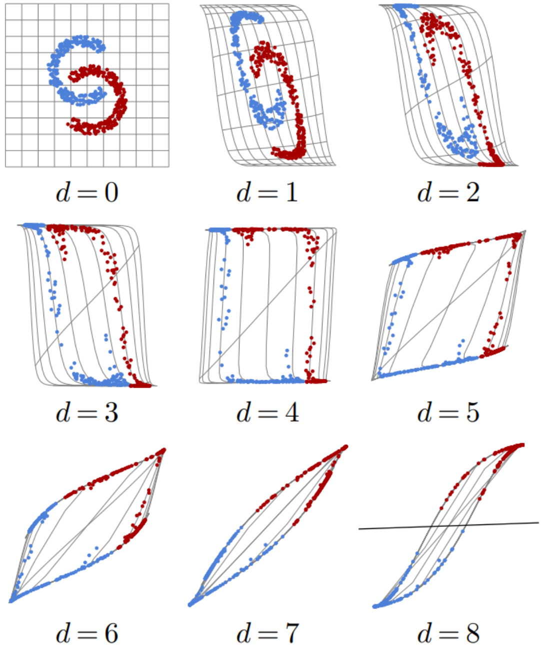
generalization from optimization on training data set (approximation of true data generating probability distribution by empirical risk minimization)

- fitting: complex function approximation
- for generalization: learning of good abstraction/representation of data/concepts

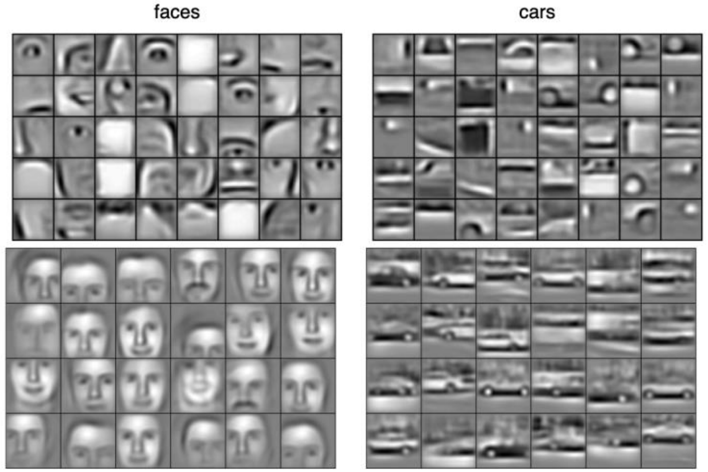
→ deep learning methods (MLP, CNN, …) optimal candidates

e.g., CNNs can learn hierarchical representation by means of many convolutional and pooling layers

the deeper the better (accuracy, hierarchical representation)



source



source

Deep Learning

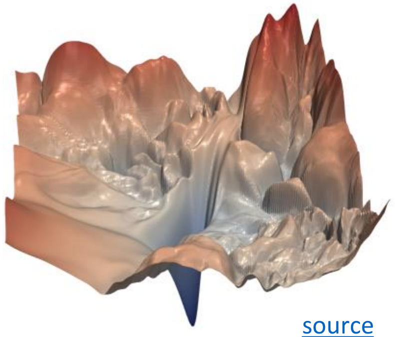
But … How to Train Deep Neural Networks?

optimization and regularization difficult

- non-convex optimization problem (e.g., local vs global minima, saddle points), easily overfitting
- many hyperparameters to tune

many methods to get it working in practice (despite partly patchy theoretical understanding)

typical loss surface:



source

optimization

- activation and loss functions
- weight initialization
- stochastic gradient descent
- adaptive learning rate
- batch normalization

explicit regularization

- weight decay
- dropout
- data augmentation
- weight sharing

implicit regularization

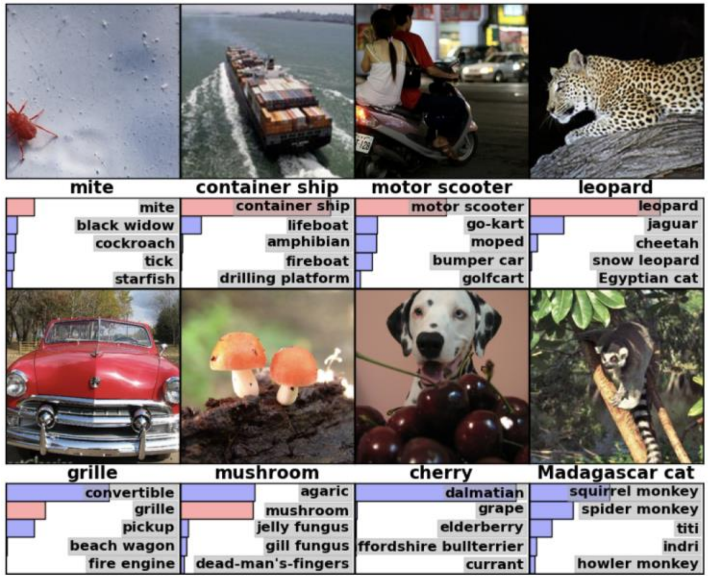
- early stopping
- batch normalization
- stochastic gradient descent

History: Rise of Deep Learning

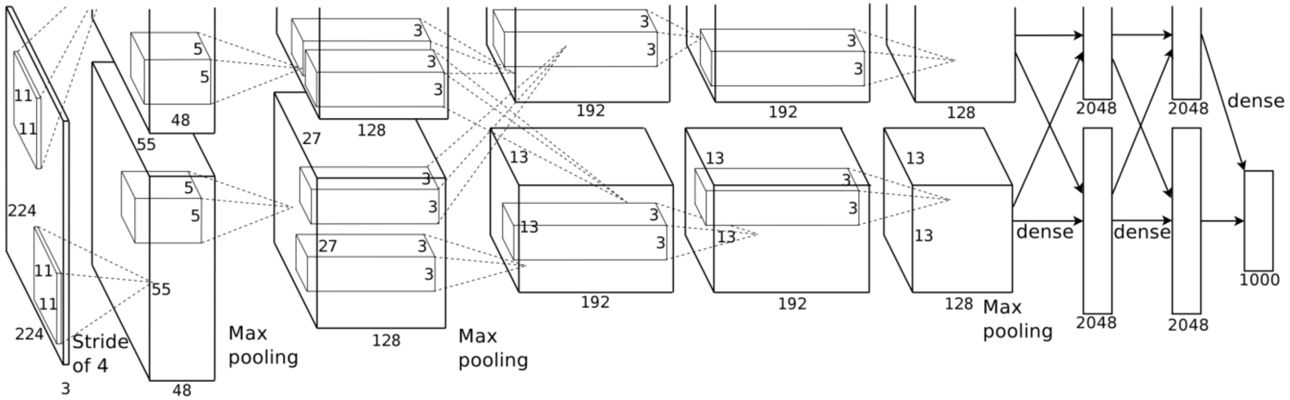
a little bit oversimplified:

deep learning = lots of training data + parallel computation + smart algorithms

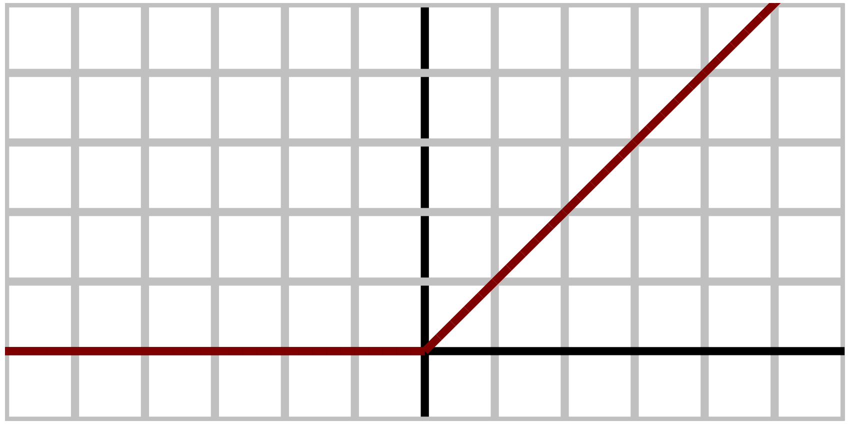
AlexNet: ImageNet (with data augmentation) + GPUs + ReLU, dropout, SGD



source



Rectified Linear Unit (ReLU)



reminder: activation function non-linear transformation of summed weighted input of a node (linear), output to be used as input for nodes of subsequent layer

needs to be differentiable for back-propagation (ReLU at 0 no issue, just set to 0 or 1)

neural network model with ReLU activation can be interpreted as exponential number of linear models that share parameters

main advantages (leading to enablement of deeper networks by better optimization):

- unlike sigmoid or $\tanh$ (predominantly used before) activation, no issue with vanishing gradients from saturation effects
- very efficient computation: constant gradients of 0 and 1 below and above input of zero
- sparse activation: many hidden nodes deactivated (output 0) → information disentangling

Weight Initialization

starting values for weights crucial for convergence of deep learning trainings

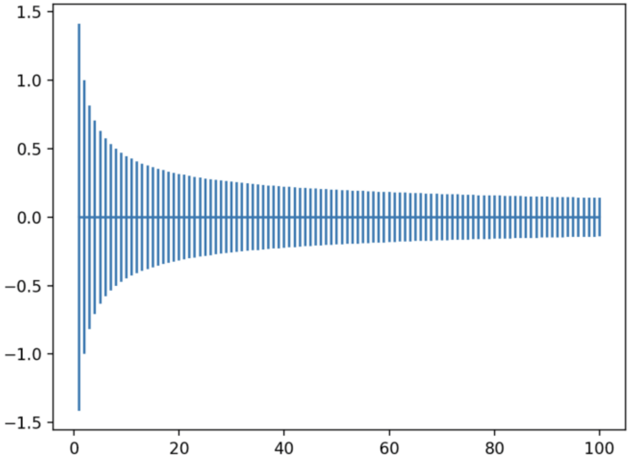
most important: need to break symmetry between different nodes in a hidden layer (same initial weights lead to identical weight updates) ← early issue in back-propagation

→ small random numbers (from Gaussian or uniform distribution) work (only bias weights set to zero by default)

but specific heuristics using information on activation function and number of inputs to a node can improve optimization (remain same expected variance between layers)

for ReLU, He initialization works well: randomly draw from zero-mean Gaussian with standard deviation $\sqrt{2/n}$

number of inputs (hidden nodes in previous layer)



Plot of Range of He Weight Initialization With Inputs From One to One Hundred

source

(Stochastic) Gradient Descent

using gradient of cost (objective) function with respect to weights: $\nabla_{\hat{\boldsymbol{w}}} J(\hat{\boldsymbol{w}})$

updates $\hat{\boldsymbol{w}} \leftarrow \hat{\boldsymbol{w}} - \eta \nabla_{\hat{\boldsymbol{w}}} J(\hat{\boldsymbol{w}})$ can be done with whole training data set (n observations) or small random sample:

- $J(\hat{\boldsymbol{w}}) = \frac{1}{n} \sum_{i=1}^{n} J_i(\hat{\boldsymbol{w}})$ batch (or deterministic) gradient descent
- $J(\hat{\boldsymbol{w}}) = J_i(\hat{\boldsymbol{w}})$ stochastic gradient descent (single example)
- $J(\hat{\boldsymbol{w}}) = \frac{1}{m} \sum_{i=1}^{m} J_i(\hat{\boldsymbol{w}})$ mini-batch stochastic gradient descent (size m)

implicit regularization: (mini-batch) SGD follows gradient of true generalization error, if no examples are repeated (but usually many epochs in training)

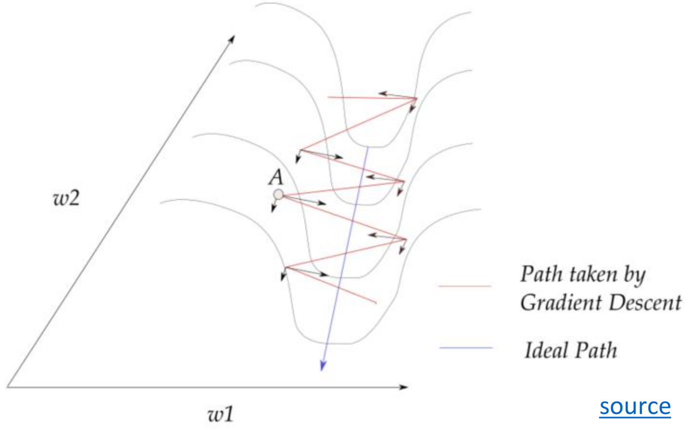
Mini-Batch Sizes

trade-off:

- larger batches give more accurate gradient estimates → allowing for higher learning rate
- smaller batches have (implicit) regularization effect and better convergence

in practice, also need to consider memory limitations and run times

Adaptive Learning Rate



source

strategies for gradient descent learning rate: constant, decaying, with momentum (escape from local minima and saddle points)

better convergence by adapting learning rate for each weight: lower/higher learning rates for weights with large/small updates (avoid direction of oscillations)

popular methods ($\hat{w}$ here denotes individual weight → $g_{\hat{w}}$ component of gradient):

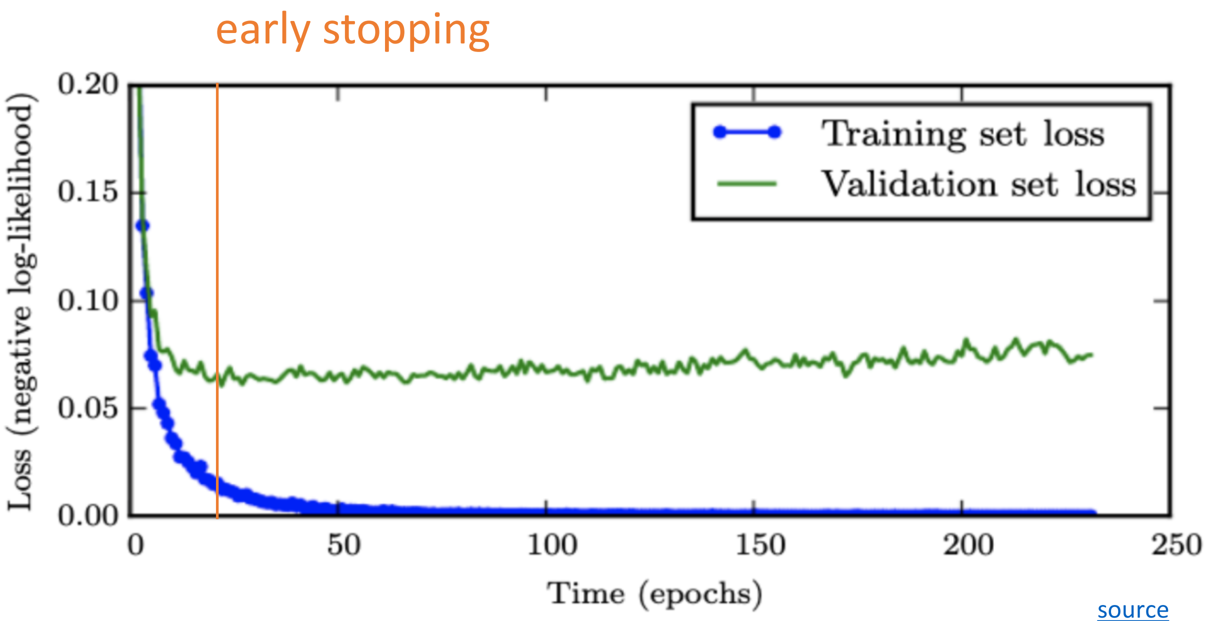
- Adagrad: $\hat{w} \leftarrow \hat{w} - \frac{\eta}{\sqrt{\sum_{\tau=1}^{t} g_{\hat{w},\tau}^2}} g_{\hat{w}}$ with t, τ denoting current and past iterations

  (issue: sum in denominator grows with more iterations → danger of stucking)
- RMSProp: $\hat{w} \leftarrow \hat{w} - \frac{\eta}{\sqrt{v(\hat{w})}} g_{\hat{w}}$ with $v(\hat{w}) \leftarrow \gamma\, v(\hat{w}) + (1-\gamma) g_{\hat{w}}^2$
- Adam (Adaptive Moment Optimization): combines RMSProp with momentum

Early Stopping

loss independently measured on validation set

halting training when overfitting begins to occur



source

(suppresses double descent)

Skip/Residual Connections

issue: degradation of training and test errors when adding more and more layers → not due to overfitting (but reason controversial)

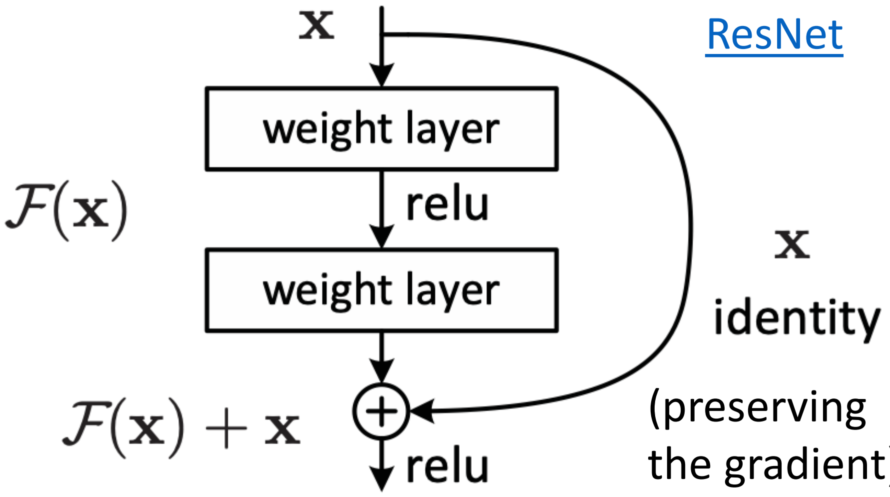
solution: learning of residuals by means of skip connections (resulting in combination of different paths through computational graph)

→ produces loss functions that train easier

together with batch normalization (avoiding exploding gradients), skip connections enable extremely deep networks (~1000 layers) without degradation

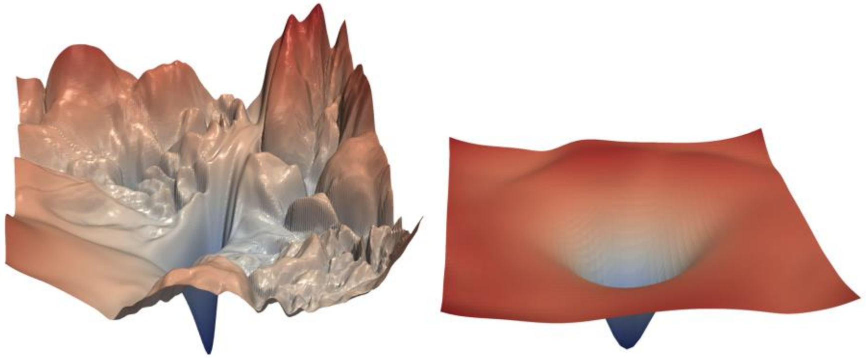
residual mapping (special kind of skip/shortcut connections):

ResNet



(preserving the gradient)

loss surface:



(a) without skip connections (b) with skip connections

source

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

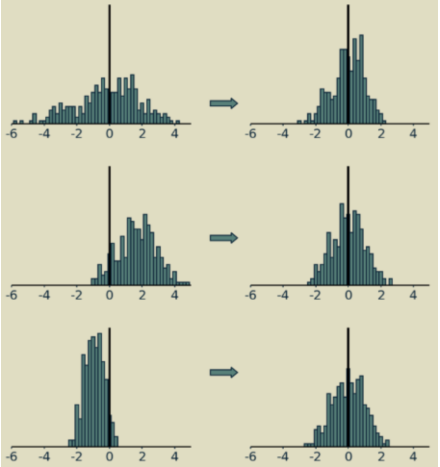
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

source



source

adaptive reparameterization of inputs to a network layer (before or after activation)

independently for each input/feature

(not to confuse with weight normalization: decoupling of length and direction of weight vectors)

to maintain expressive power (optional): γ, β learned together with weights via back-propagation

Benefits from Batch Normalization

- allows higher learning rates
- reduces importance of weight initialization
- alleviates vanishing/exploding gradients
- (implicit) regularization effect: introducing both additive and multiplicative noise, sometimes making dropout (multiplicative noise) unnecessary

reason why batch normalization improves optimization still controversial

most plausible explanation: smoothening of loss landscape (similar to skip connections)

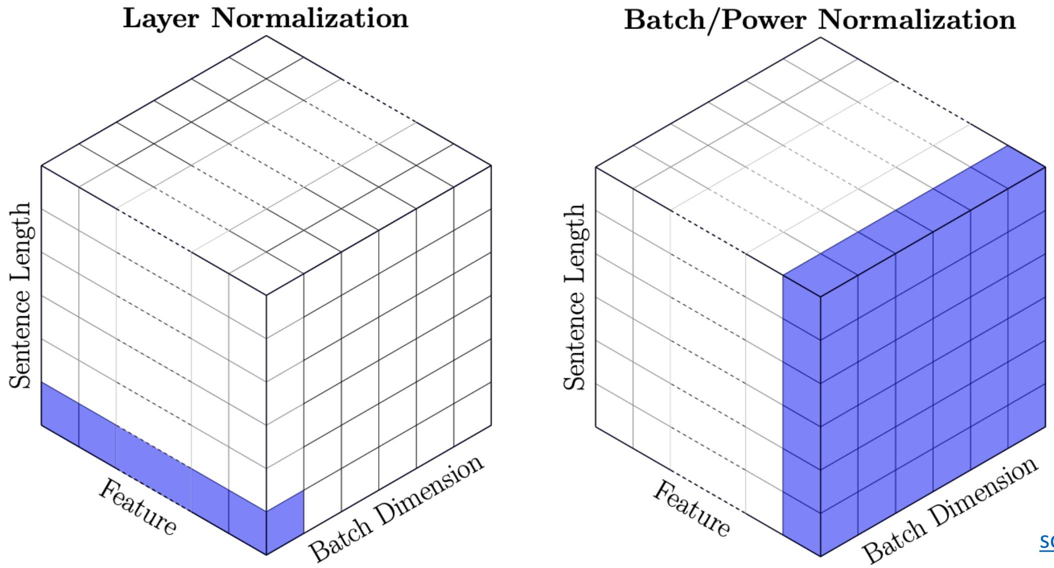
Layer Normalization

layer normalization

normalization over inputs/features, independently for each data sample

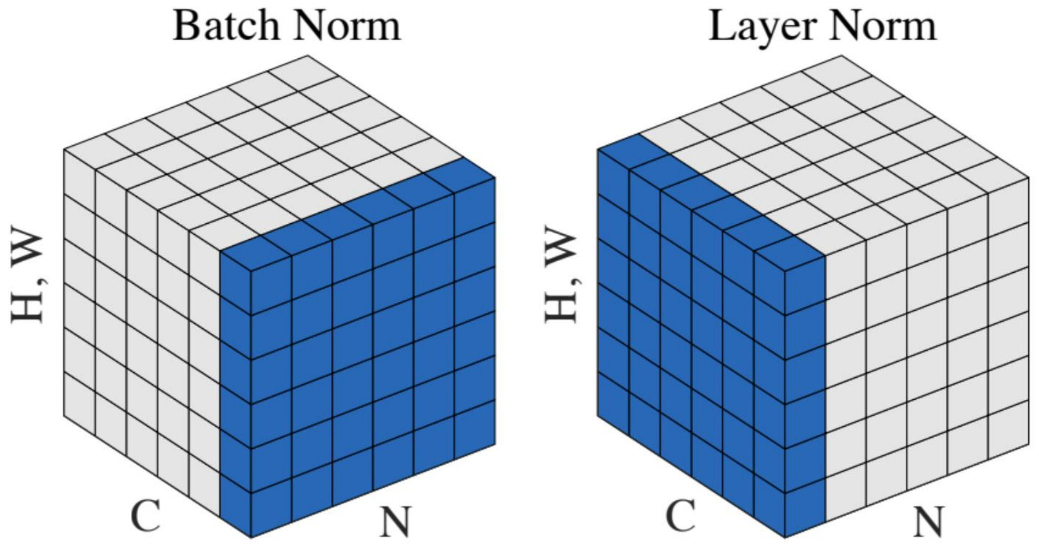
→ mean and variance shared over all hidden nodes of a network layer

batch norm often in computer vision (CNN), layer norm in NLP (variable-sized inputs)



source

transformer:



source

Comparison to Shallow Methods

Feature Engineering vs Feature Learning

shallow learning:

representation encoded in features

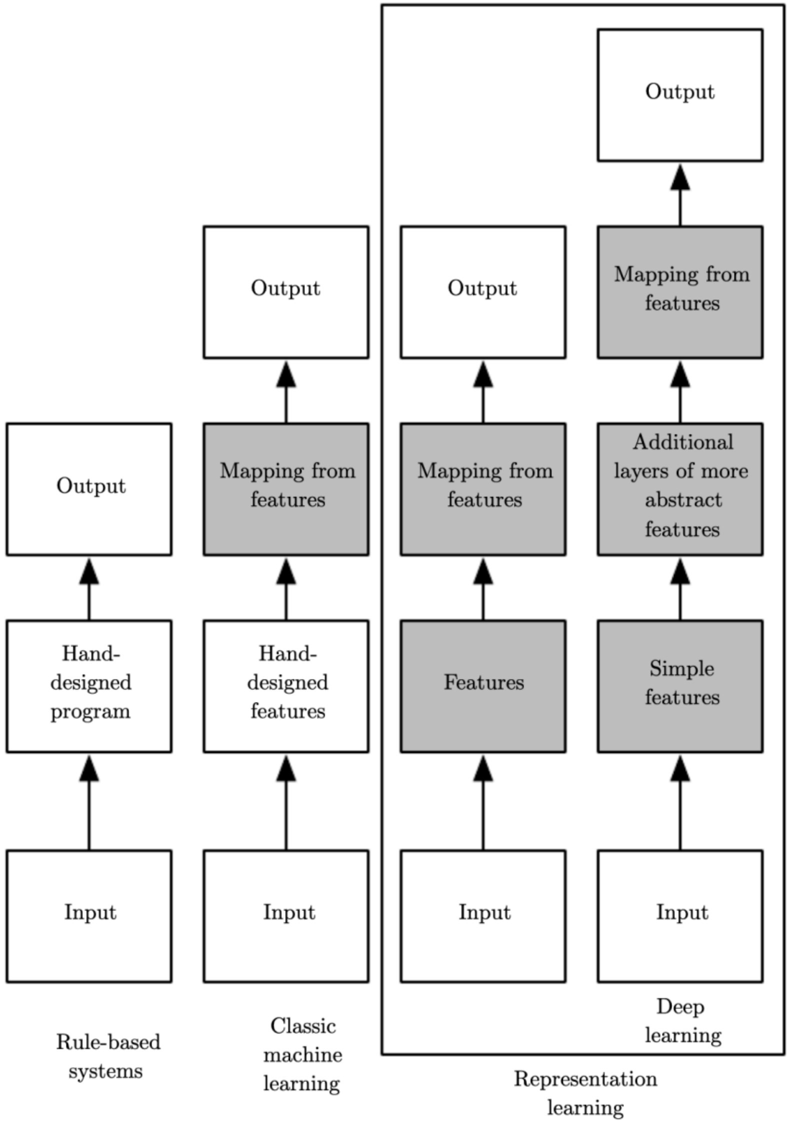
→ feature engineering

deep learning:

representation encoded in network

→ feature/representation learning

(hierarchy of concepts learned from raw data in deep graph with many layers)



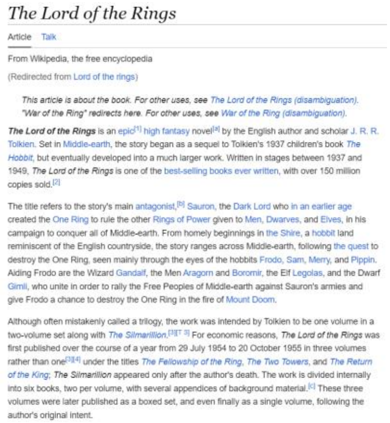
source

Structured/Tabular vs Unstructured Data

unstructured data: homogenous

→ deep learning rules

ImageNet



The Lord of the Rings

Article Talk

From Wikipedia, the free encyclopedia

(Redirected from Lord of the rings)

This article is about the book. For other uses, see The Lord of the Rings (disambiguation).
"War of the Ring" redirects here. For other uses, see War of the Ring (disambiguation).

The Lord of the Rings is an epic[1] high fantasy novel[a] by the English author and scholar J. R. R. Tolkien. Set in Middle-earth, the story began as a sequel to Tolkien's 1937 children's book *The Hobbit*, but eventually developed into a much larger work. Written in stages between 1937 and 1949, *The Lord of the Rings* is one of the best-selling books ever written, with over 150 million copies sold.[2]

The title refers to the story's main antagonist,[b] Sauron, the Dark Lord who in an earlier age created the One Ring to rule the other Rings of Power given to Men, Dwarves, and Elves, in his campaign to conquer all of Middle-earth. From homely beginnings in the Shire, a hobbit land reminiscent of the English countryside, the story ranges across Middle-earth, following the quest to destroy the One Ring, seen mainly through the eyes of the hobbits Frodo, Sam, Merry, and Pippin. Aiding Frodo are the Wizard Gandalf, the Men Aragorn and Boromir, the Elf Legolas, and the Dwarf Gimli, who unite in order to rally the Free Peoples of Middle-earth against Sauron's armies and give Frodo a chance to destroy the One Ring in the fire of Mount Doom.

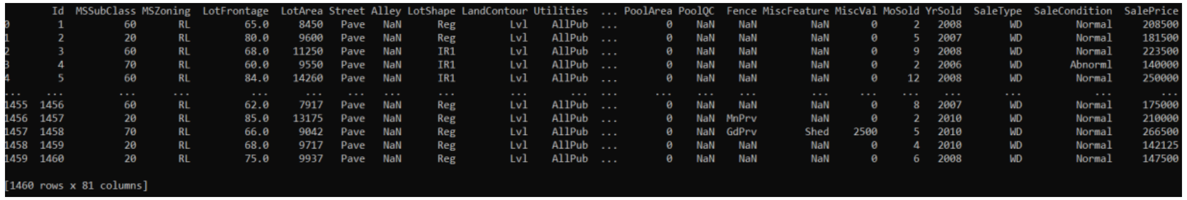
Although often mistakenly called a trilogy, the work was intended by Tolkien to be one volume in a two-volume set along with *The Silmarillion*.[3][T 3] For economic reasons, *The Lord of the Rings* was first published over the course of a year from 29 July 1954 to 20 October 1955 in three volumes rather than one[3][4] under the titles *The Fellowship of the Ring*, *The Two Towers*, and *The Return of the King*; *The Silmarillion* appeared only after the author's death. The work is divided internally into six books, two per volume, with several appendices of background material.[c] These three volumes were later published as a boxed set, and even finally as a single volume, following the author's original intent.

structured data: heterogenous

→ feature engineering needed

→ deep learning loses its advantage over shallow methods

→ e.g., gradient boosting still prominent



	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	8	2007	WD	Normal	175000
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0	2	2010	WD	Normal	210000
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	GdPrv	Shed	2500	5	2010	WD	Normal	266500
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	4	2010	WD	Normal	142125
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	6	2008	WD	Normal	147500

[1460 rows x 81 columns]

Categorical Variables

tabular data usually heterogenous, often with sparse categorical variables (like color of an object)

→ need for an encoding for categorical variables

different possibilities:

- ordinal encoding (introduces artificial order to unordered categories)
- leave-one-out encoding (use mean of target for given category excluding current row, used in CatBoost)
- one-hot encoding (can suffer from curse of dimensionality)
- embeddings (can also alleviate issue of non rotationally-invariant data)

Embeddings

Vector Representations

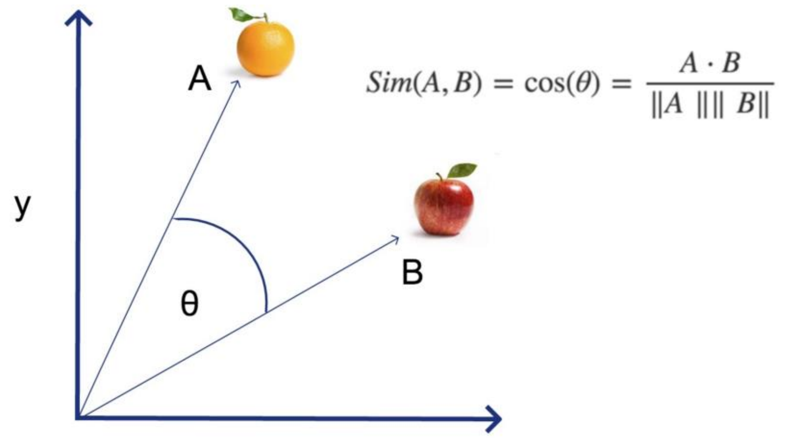
embeddings: representation of entities by vectors

similarity between embeddings by, e.g., cosine similarity → semantic similarity

most famous application: word embeddings → associations (natural language processing)

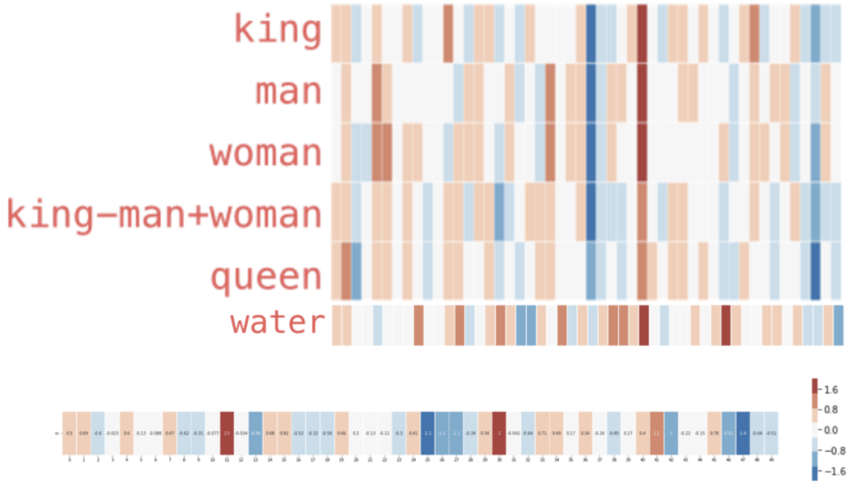
but general concept: embeddings of (categorical) features (e.g., products in recommendation engines)

learned via co-occurrence (e.g., word2vec)



but also direction of difference vectors interesting (analogies):

king - man + woman ~= queen



source

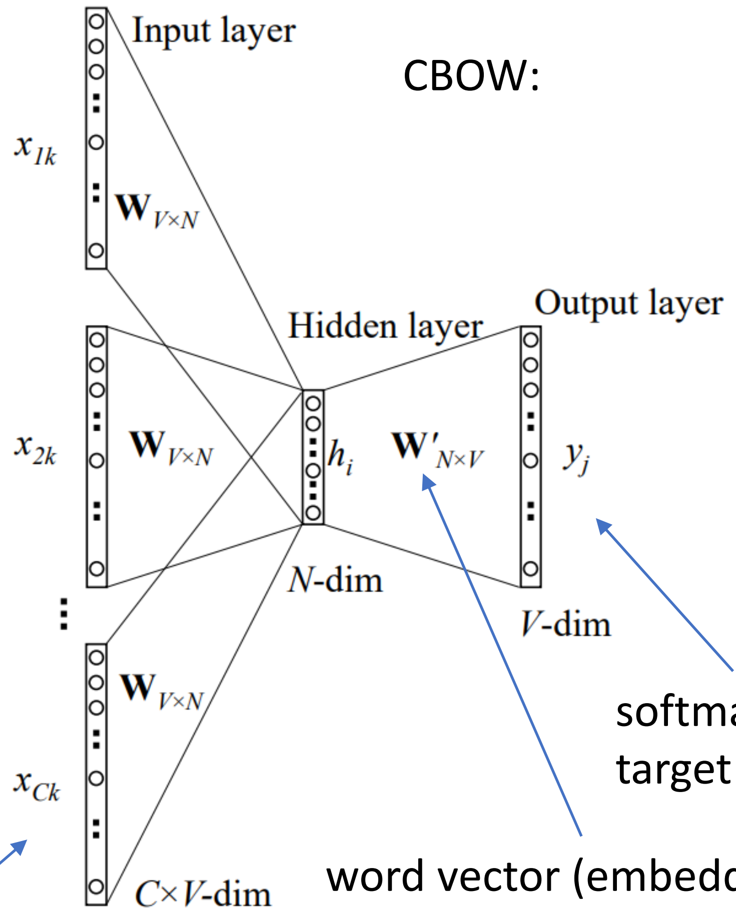
Some Thoughts on Word Embeddings

can be implemented as

- neural network with single hidden layer (linear activation)
- using, e.g., bag-of-words approach (predict masked word from its surroundings)

→ not context-aware (need for attention or RNN)

CBOW:



source

softmax of target word

word vector (embedding) by multiplication with one-hot vector

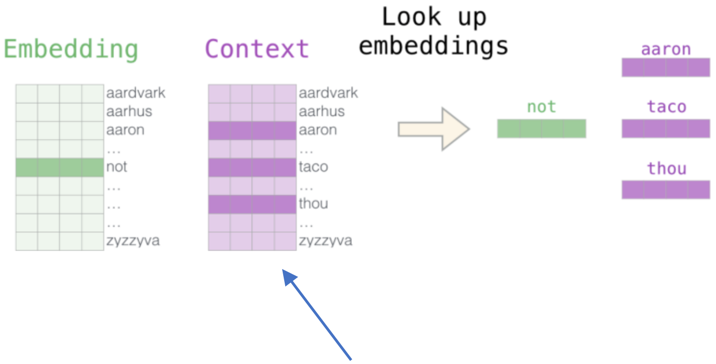
one-hot vectors of context words

word2vec

focus on generating word embeddings, not entire language model → negative sampling:

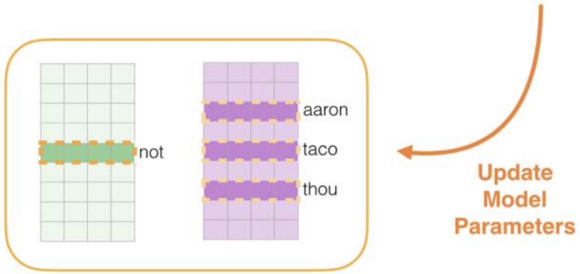
- use input and output words of language model as features, binary target if neighbors (dropping expensive projection to output vocabulary → much faster)
- include random negative samples (samples of words that are not neighbors)

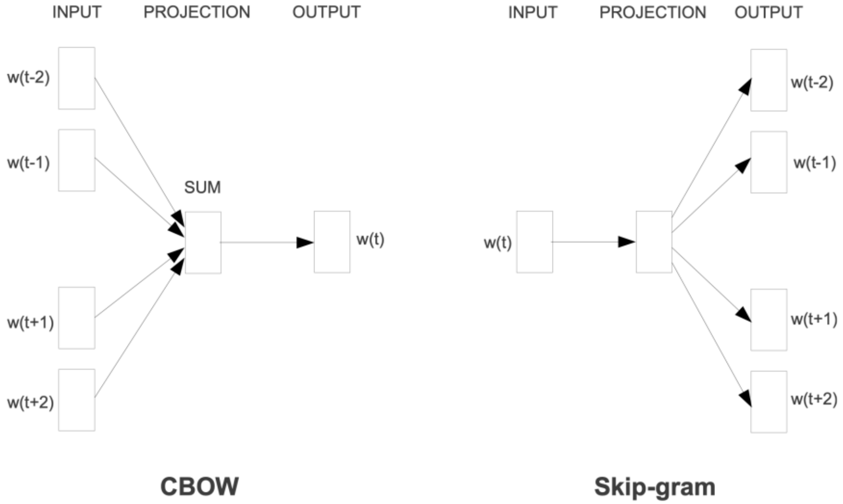
not a deep neural network (just single hidden layer)



“helper” matrix

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68





Continuous Bag Of Words

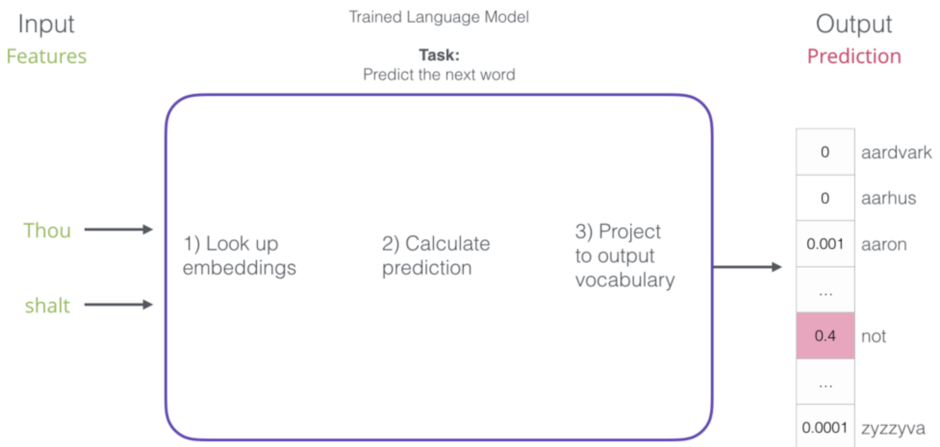
source

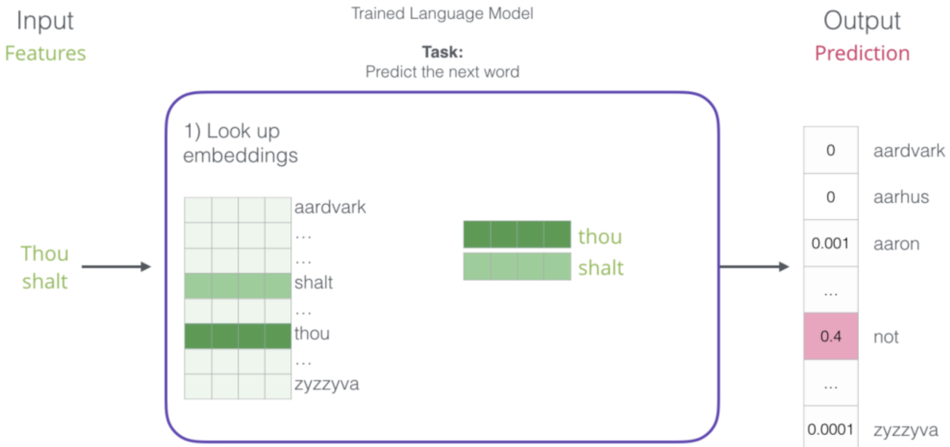
Word Embeddings as Part of Language Model

language models contain embedding matrix as part of learned parameters

- can be extracted and subsequently used as pre-trained embeddings for other task
- typically several hundred dimensions for word vectors (to be compared with vocabulary sizes of many thousands)
- trained on huge data sets

next-word prediction:





source

Neural Language Models

using neural networks: learning distributed representation of words

self-supervised learning: sliding (with some sliding window) over text to generate training data set

learning of embeddings: kind of feature learning (dimensions of embeddings vectors)

contextually-meaningful embeddings can be learned by means of sequence models (RNN/LSTM, transformer)

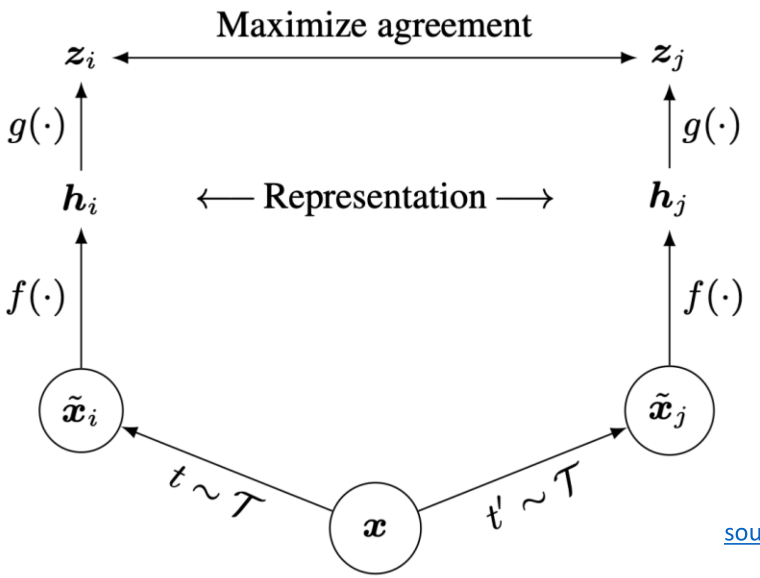
Contrastive Learning

goal: create embedding space in which similar samples are close to each other and dissimilar ones are far apart

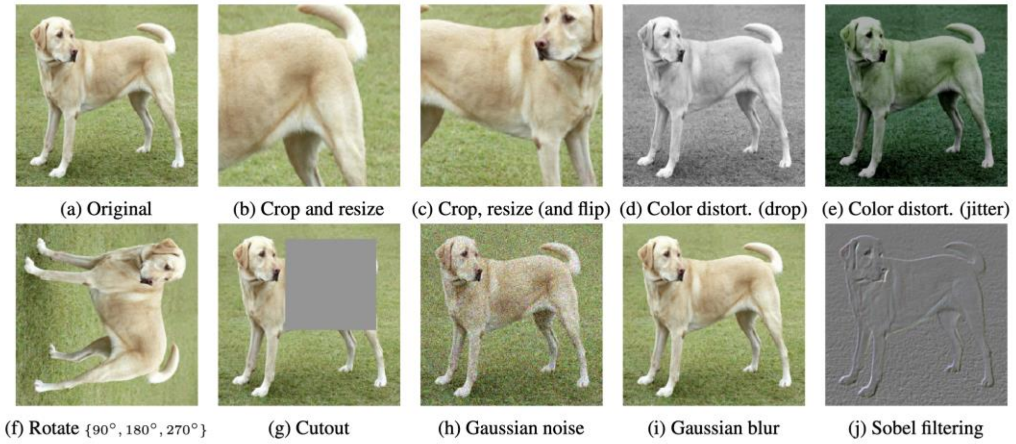
often learned in a self-supervised way

examples:

- natural language processing: word2vec
- computer vision: SimCLR, SimCLRv2 (learning of image representations)



source



Autoencoders

Representation Learning

autoencoders as prime example of representation learning

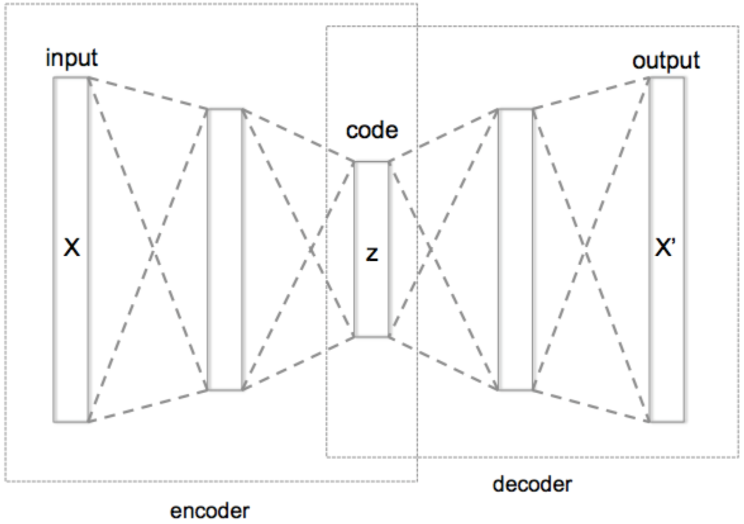
combination of

- encoder: converting input data into different representation (code)
- decoder: converting learned representation back into original format

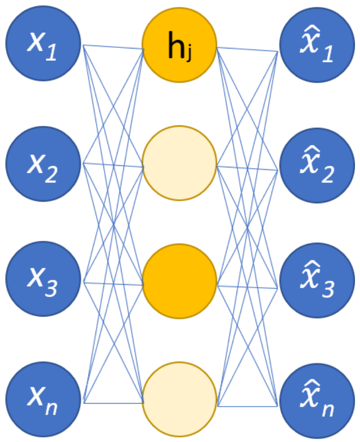
possibilities to avoid simple duplication:

- undercomplete autoencoders: code with smaller dimension (less nodes) than input (generalized PCA: autoencoder is PCA if encoder and decoder are linear transformations)
- sparse autoencoders: sparsity penalty to deactivate hidden nodes (e.g., with help from ReLU activation)

learned in the same way as feed-forward neural networks



from wikipedia



History: Unsupervised Pre-Training

breakthrough in effectiveness of deep learning training in 2006:

Deep Belief Networks introduced idea of greedily initializing each layer by unsupervised learning

(using an energy-based method called Restricted Boltzmann Machine)

→ commonly seen as actual starting point of deep learning wave

only later, ReLU activation functions (and other improvements) enabled effective deep learning without unsupervised pre-training

but unsupervised pre-training still beneficial in context of semi-supervised learning, using large amounts of unlabeled data

Stacked Autoencoders

besides dimensionality reduction, autoencoders can also be used (instead of Restricted Boltzmann Machines) for unsupervised pre-training

→ feature learning (internal distributed representations, high-level abstractions of input data), initializing weights in region near a good local minimum

stacking of autoencoders (or RBMs) → industry adoption:

- object recognition ([cat paper](#))
- [speech recognition in industry](#)

stimuli for cat neuron:

[source](#)

Recurrent Neural Networks (RNN)

Sequential Structures

speech recognition, natural language processing, time series, …

problem: need to generalize across different points in time (or multiple positions of words within a sentence) and learn from context

idea: parameter sharing across different parts of model

CNNs apply parameter sharing (convolutions) on grid-like structures (including 1-D grids like time series or speech), but this is limited to neighboring inputs (per layer).

→ need for approach to learn sequential structures (process input as stream, e.g., speech, rather than one batch, e.g., image)

Back-Propagation through Time

context

RNN: output of node directed back to input

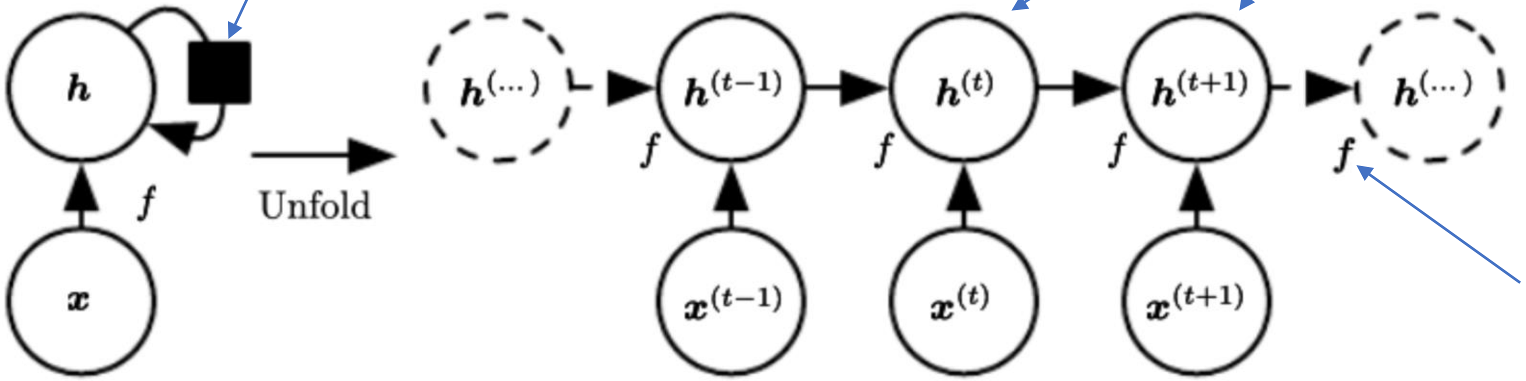
activation of recurrent nodes at that point in time

interaction with delay of single time step

computational graph with repeated pieces:

hidden layer: several recurrent neurons

input layer: e.g., one-hot encoded words or embeddings



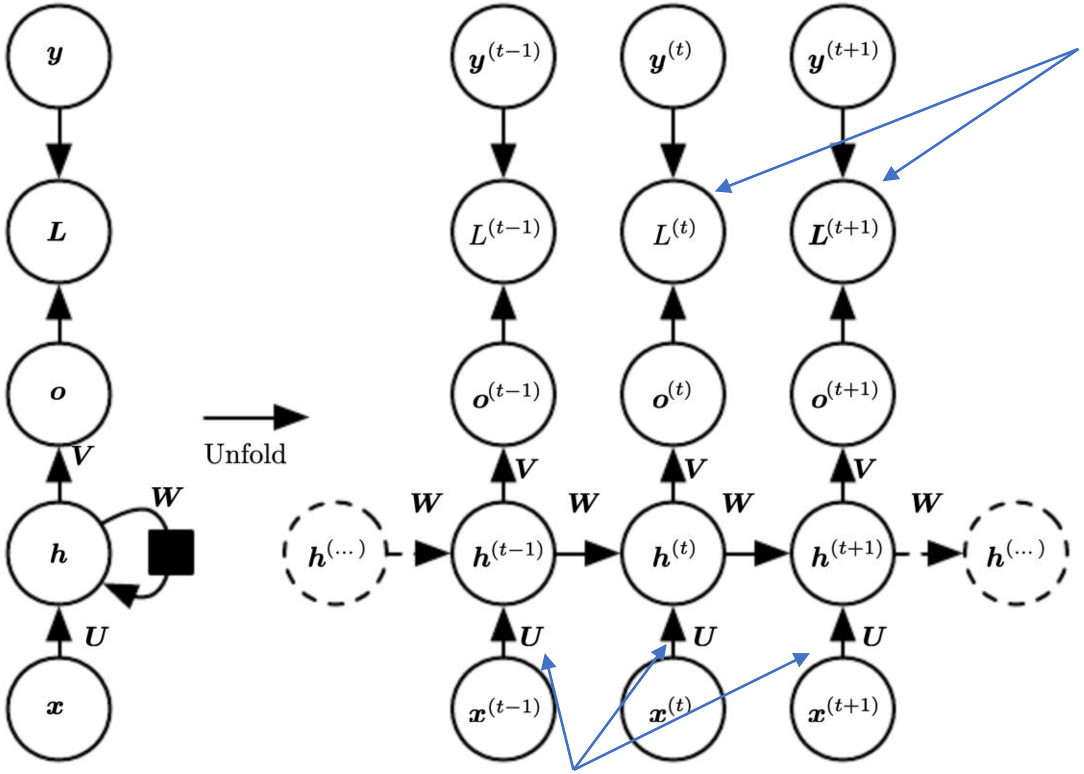
single model operating on all time steps

source

different kind of depth: one recurrence for each sequential step (e.g., word)

Weight Sharing across Time

example: input-output mapping at each time step with recurrent hidden nodes



total loss for sequence: sum of losses over time steps

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)}),
\end{aligned}
$$

Markov dependence on previous hidden state

more popular than ReLU here (exploding gradients)

discrete outputs

parameter/weight sharing over time

source

Further Examples

summarization of sequence:

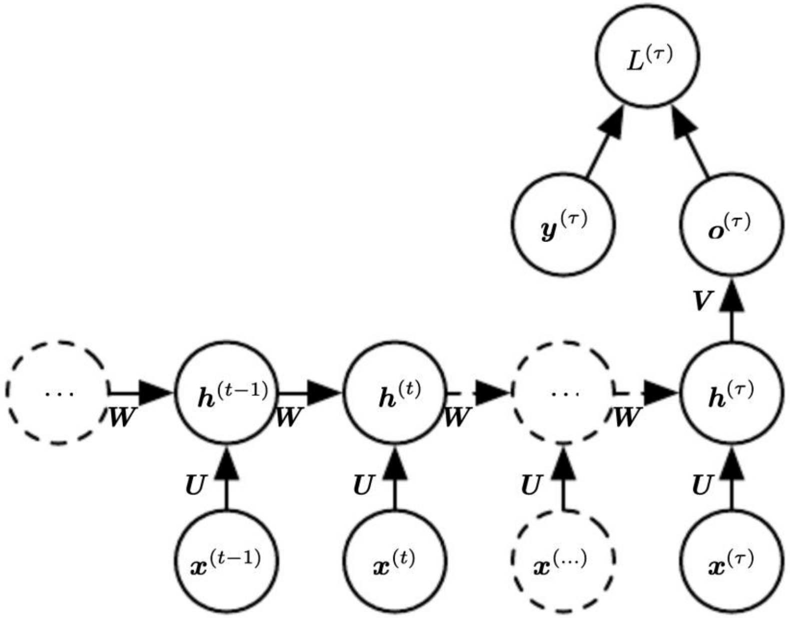
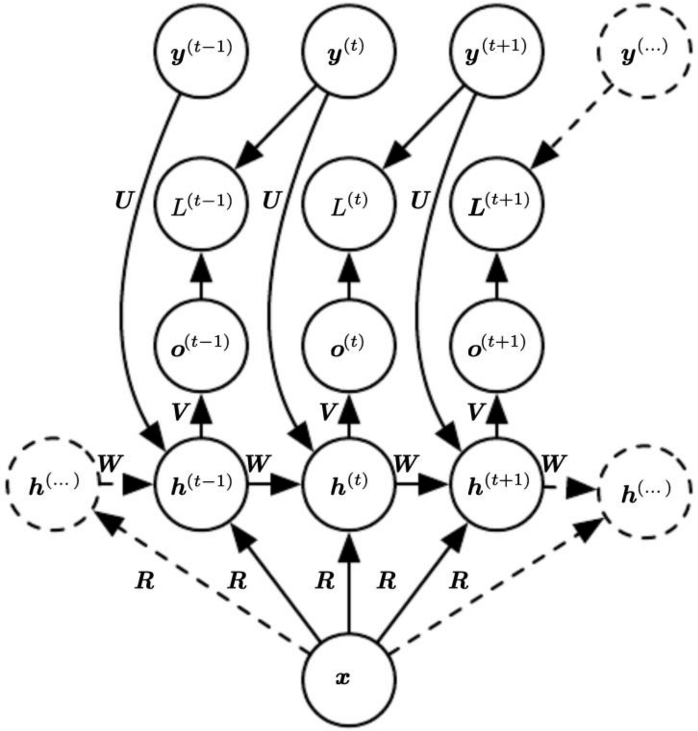
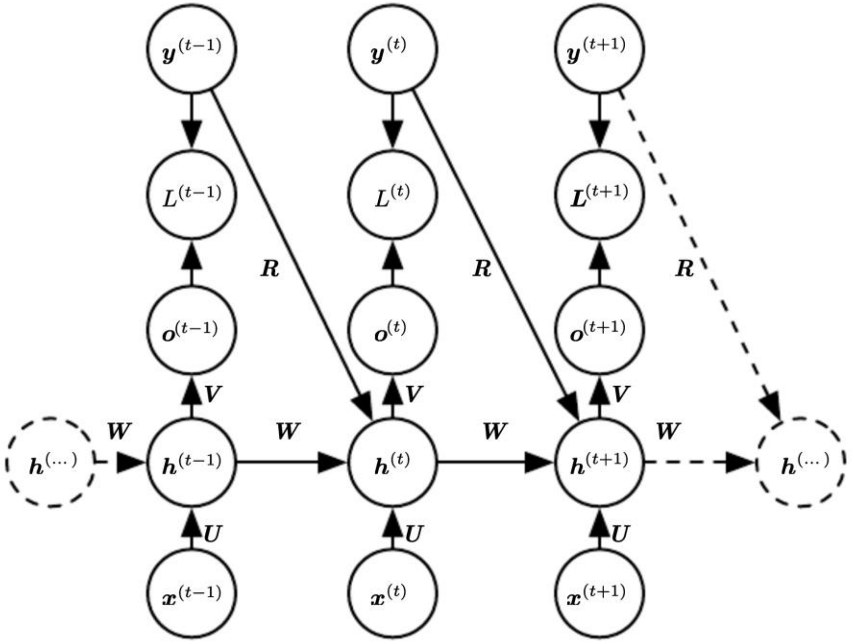


image captioning:

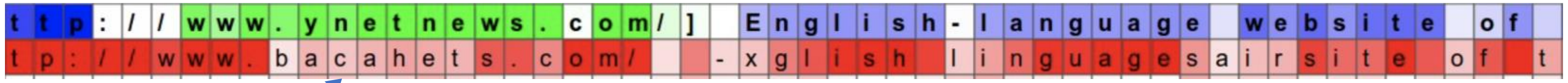


conditional sequence:



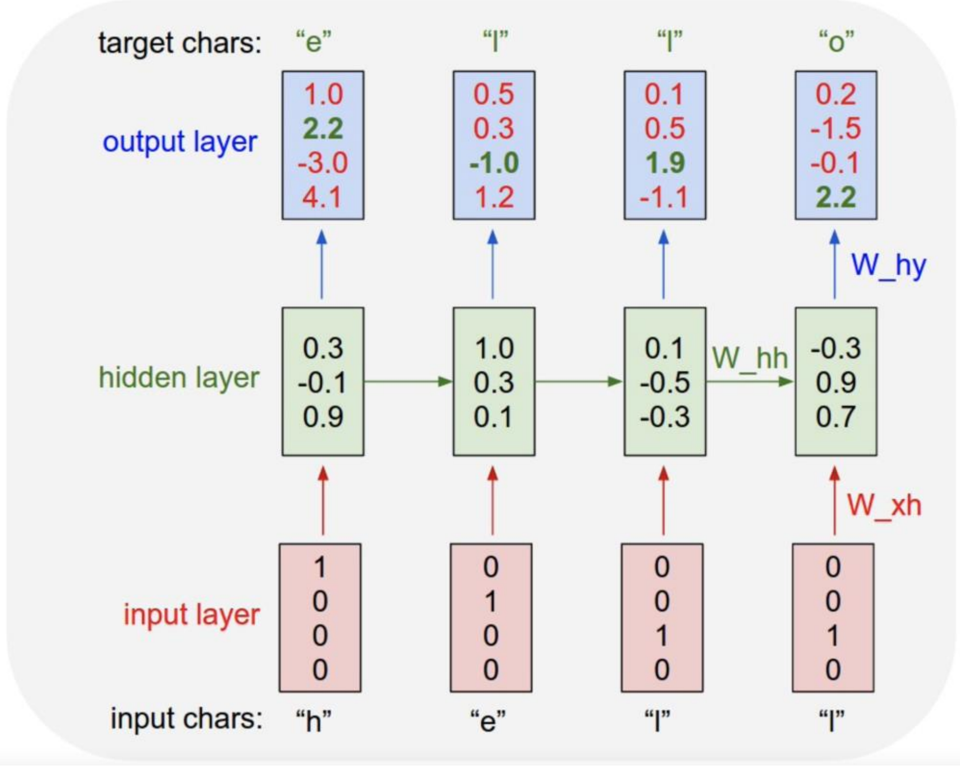
Visualization

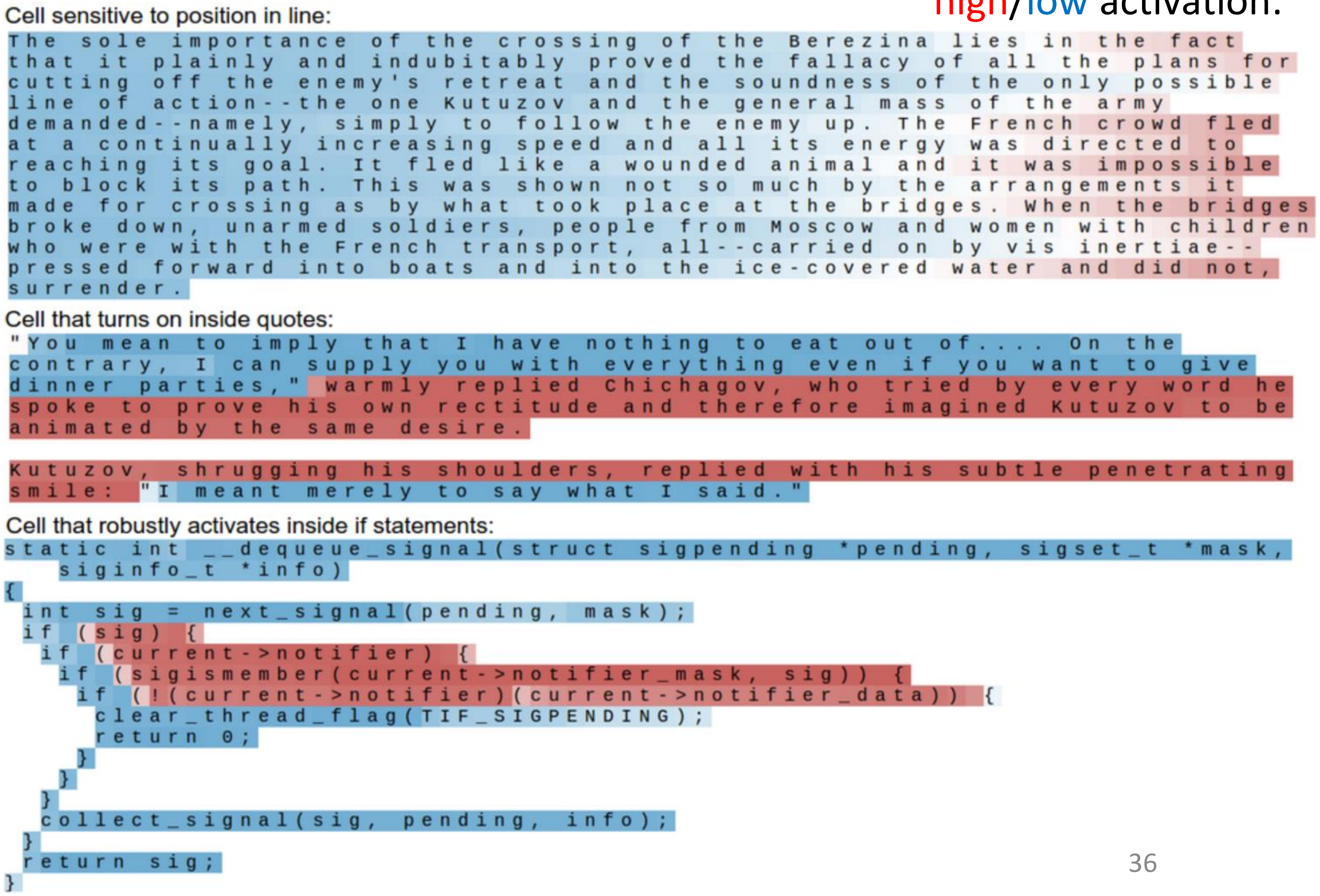
neuron getting excited inside URLs (excited, not excited):



next character prediction

high/low activation:

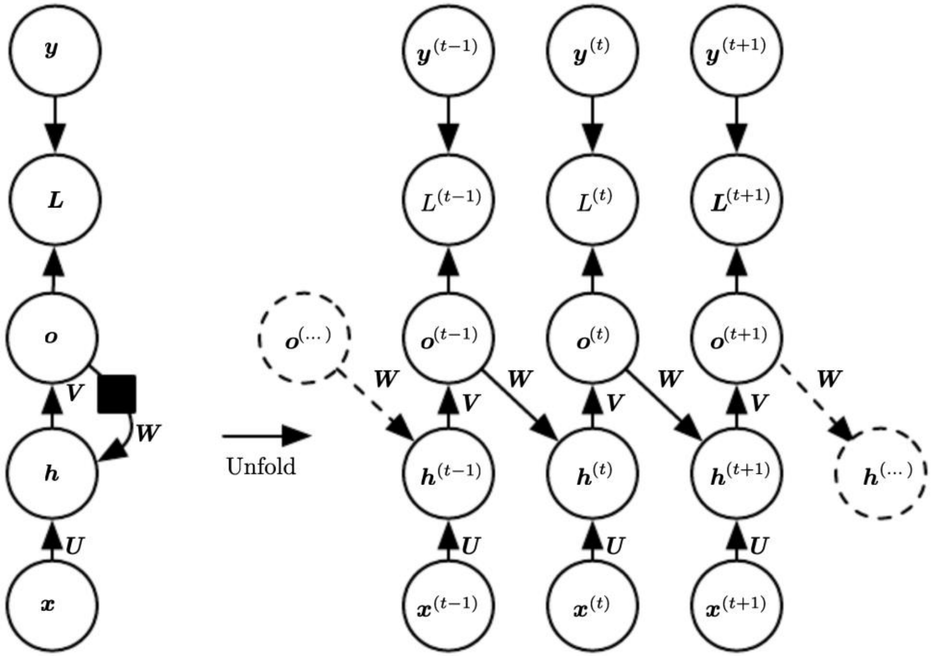


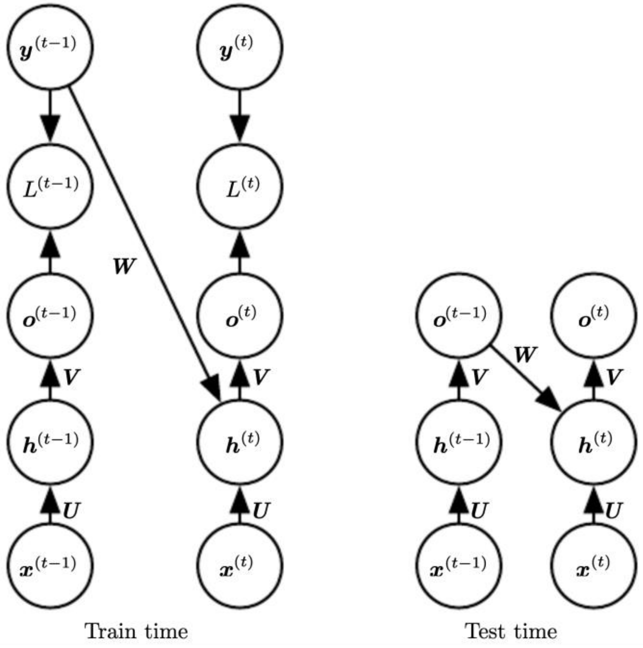


source

Teacher Forcing

for feedback from output to hidden layer: instead of feeding model output back into itself, use target values directly





source

long-term memory: weights
short term memory: activation patterns (context)

Gated RNNs

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)} \right)$$

$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)}$$

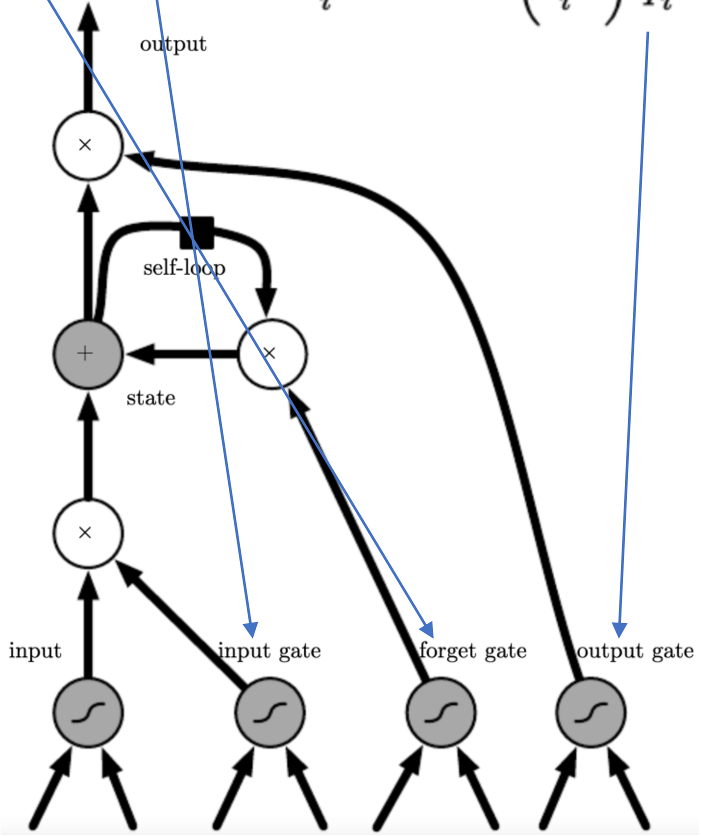
issue with long chain of gradients through recurrences: vanishing (mainly) and exploding gradients in back-propagation

→ need to focus on important sequence elements

replace usual recurrent hidden nodes with long short-term memory (LSTM) cells with internal recurrence (self-loop)

- linear self-loop in addition to outer recurrence of RNN: error carousel preserving (→ long) short-term memory (i.e., activation patterns)
- sigmoid activations with independent weights for input, forget, and output gates

other prominent gated RNN: gated recurrent unit (GRU)



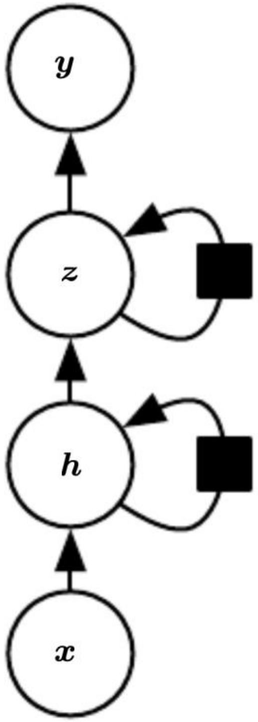
source

Issues with RNNs

- especially LSTMs very computationally expensive (with lots of parameters)
- transfer learning (using pre-trained network layers on new task, e.g., popular for CNNs) difficult

(hierarchical) representation learning: go deep by stacking several layers of recurrent nodes (e.g., important for speech recognition)
→ worsening efficiency even more

(self-)attention and transformers to the rescue …



source

Literature

easily digestible overview:

[The Little Book of Deep Learning]

papers:

- Deep Learning overview
- ResNet
- A Neural Probabilistic Language Model
- word2vec
- training of Deep Belief Nets

blogs:

- The Illustrated Word2vec
- Karpathy on RNNs

Black-Box Models

To build trust in AI systems, individual predictions/actions need to be fully transparent, i.e., explainable.

Unfortunately, complex models like deep learning methods are difficult to interpret.

→ need for model-agnostic methods to explain black-box models

examples: local surrogates (LIME), Shapley values (SHAP)

overview