Deep Learning

Michèle Sebag
TAO

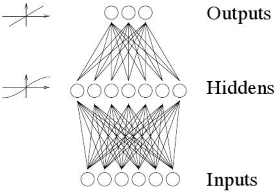
Université Paris-Saclay

Jan. 21st, 2016

Credit for slides: Yoshua Bengio, Yann Le Cun, Nando de Freitas, Christian Perone, Honglak Lee, Ronan Collobert, Tomas Mikolov, Rich Caruana

Overview

Neural Nets

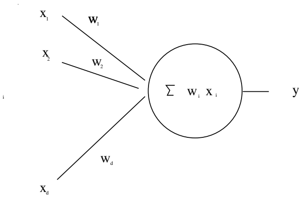


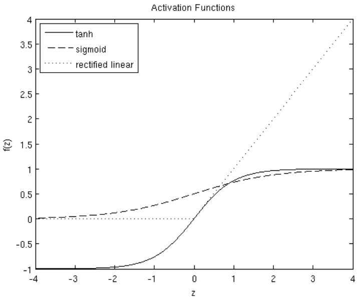
(C) David McKay - Cambridge Univ. Press

History

1943	A neuron as a computable function $y = f(\mathbf{x})$ Intelligence $\rightarrow$ Reasoning $\rightarrow$ Boolean functions	Pitts, McCullough
1960	Connexionism + learning algorithms	Rosenblatt
1969	AI Winter	Minsky-Papert
1989	Back-propagation	Amari, Rumelhart & McClelland, LeCun
1995	Winter again	Vapnik
2005	Deep Learning	Bengio, Hinton

One neuron: input, weights, activation function





$$\mathbf{x} \in \mathbb{R}^d \qquad z = \sum_i w_i x_i \qquad f(z) \in \mathbb{R}$$

Activation functions

- Thresholded — 0 if $z < threshold$, 1 otherwise
- Linear — z
- Sigmoid — $1/(1+e^{-z})$
- Tanh — $\frac{e^z - e^{-z}}{e^z + e^{-z}}$
- Radius-based — e^{-z^2/σ^2}
- Rectified linear (ReLU) — $\max(0, z)$

Learning the weights

An optimization problem: Define a criterion

- Supervised learning — classification, regression

$$\mathcal{E} = \{(\mathbf{x}_i, y_i), x_i \in \mathbb{R}^d, y_i \in \mathbb{R}, i = 1 \ldots n\}$$

- Reinforcement learning

$$\pi : \text{ State space } \mathbb{R}^d \mapsto \text{ Action space } \mathbb{R}^{d'}$$

Mnih et al., 2015

Main issues

- Requires a differentiable / continuous activation function
- Non convex optimization problem

Back-propagation, 1

Notations

Input $\mathbf{x} = (x_1, \ldots x_d)$

From input to the first hidden layer

$$z_j^{(1)} = \sum w_{jk} x_k$$

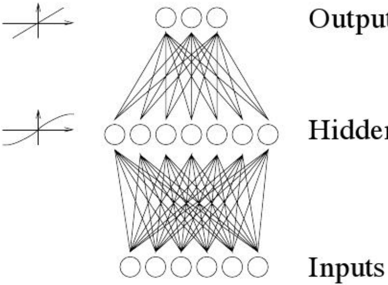
$$x_j^{(1)} = f(z_j^{(1)})$$

From layer i to layer $i+1$

$$z_j^{(i+1)} = \sum w_{jk}^{(i)} x_k^{(i)}$$

$$x_j^{(i+1)} = f(z_j^{(i+1)})$$

(f: e.g. sigmoid)



Back-propagation, 2

Input $(\mathbf{x}, y)$, $\mathbf{x} \in \mathbb{R}^d$, $y \in \{-1, 1\}$
Phase 1 Propagate information forward

- For layer $i = 1 \dots \ell$

  For every neuron j on layer i

  $z_j^{(i)} = \sum_k w_{j,k}^{(i)} x_k^{(i-1)}$

  $x_j^{(i)} = f(z_j^{(i)})$

Phase 2 Compare the target output (y) to what you get $(x_1^{(\ell)})$

assuming scalar output for simplicity

- Error: difference between $\hat{y} = x_1^{(\ell)}$ and y.
  Define

  $$e^{output} = f'(z_1^{\ell})[\hat{y} - y]$$

  where $f'(t)$ is the (scalar) derivative of f at point t.

Back-propagation, 3

Phase 3 retro-propagate the errors

$$e_j^{(i-1)} = f'(z_j^{(i-1)}) \sum_k w_{kj}^{(i)} e_k^{(i)}$$

Phase 4: Update weights on all layers

$$\Delta w_{ij}^{(k)} = \alpha e_i^{(k)} x_j^{(k-1)}$$

where α is the learning rate < 1.

Adjusting the learning rate is a main issue

Properties of NN

Good news

- MLP, RBF: universal approximators

For every decent function f ($= f^2$ has a finite integral on every compact of $\mathbb{R}^d$)
for every $\epsilon > 0$,
there exists some MLP/RBF g such that $||f - g|| < \epsilon$.

Bad news

- Not a constructive proof (the solution exists, so what ?)
- Everything is possible $\rightarrow$ no guarantee (overfitting).

Very bad news

- A non convex (and hard) optimization problem
- Lots of local minima
- Low reproducibility of the results

The curse of NNs

Le Cun 2007

- **The NIPS community has suffered of an acute convexivitis epidemic**
  - ML applications seem to have trouble moving beyond logistic regression, SVMs, and exponential-family graphical models.
  - For a new ML model, convexity is viewed as a virtue
  - Convexity is sometimes a virtue
  - But it is often a limitation

  - ML theory has essentially never moved beyond convex models
    - the same way control theory has not really moved beyond linear systems

  - Often, the price we pay for insisting on convexity is an unbearable increase in the size of the model, or the scaling properties of the optimization algorithm [O(n^2), O(n^3)...]

http://videolectures.net/eml07_lecun_wia/

Old Key Issues (many still hold)

Model selection

- Selecting number of neurons, connexion graph — **More $\not\Rightarrow$ Better**
- Which learning criterion — avoid overfitting

Algorithmic choices — a difficult optimization problem

- Enforce stability through relaxation

$$\mathbf{W}_{neo} \leftarrow (1-\alpha)\mathbf{W}_{old} + \alpha \mathbf{W}_{neo}$$

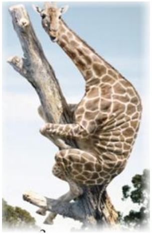
- Decrease the learning rate α with time
- Stopping criterion — early stopping

Tricks

- Normalize data
- Initialize **W** small !

Overview

Toward deeper representations



Invariances matter

- The label of an image is invariant through small translation, homothety, rotation...
- Invariance of labels $\rightarrow$ Invariance of model

$$y(x) = y(\sigma(x)) \rightarrow h(x) = h(\sigma(x))$$

Enforcing invariances

- by augmenting the training set:

$$\mathcal{E} = \{(x_i, y_i)\} \bigcup \{(\sigma(x_i), y_i)\}$$

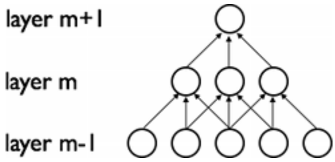
- by structuring the hypothesis space

Convolutional networks

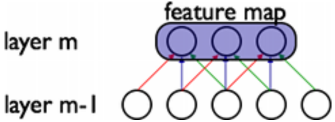
Hubel & Wiesel 1968

Visual cortex of the cat

- cells arranged in such a way that
- ... each cell observes a fraction of the visual field — receptive field
- ... their union covers the whole field



- Layer m: detection of local patterns — (same weights)

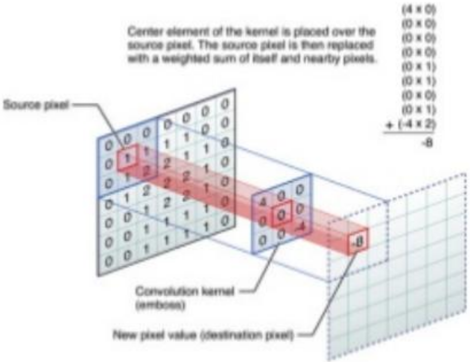


- Layer $m+1$: non linear aggregation of output of layer m

Ingredients of convolutional networks

1. Local receptive fields

(aka kernel or filter)



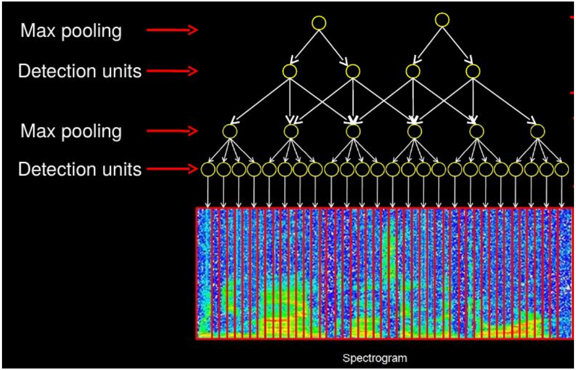
2. Sharing weights

through adapting the gradient-based update: the update is averaged over all occurrences of the weight.

Reduces the number of parameters by several orders of magnitude

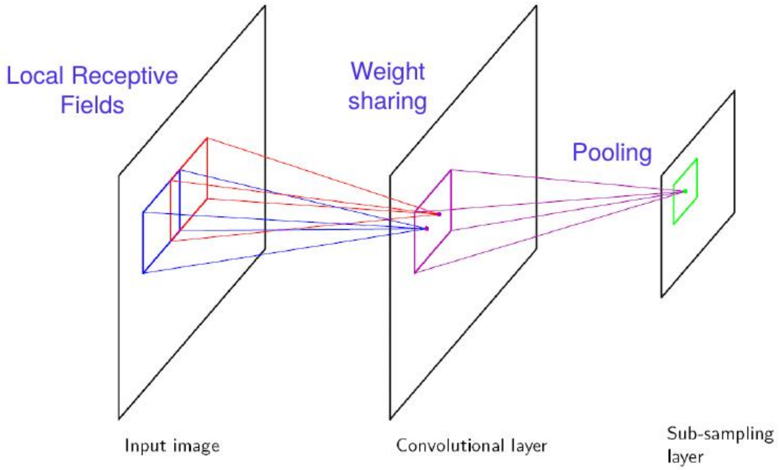
Ingredients of convolutional networks, 2

3. Pooling: reduction and invariance



- Overlapping / non-overlapping regions
- Return the max / the sum of the feature map over the region
- Larger receptive fields (see more of input)

Convolutional networks, summary

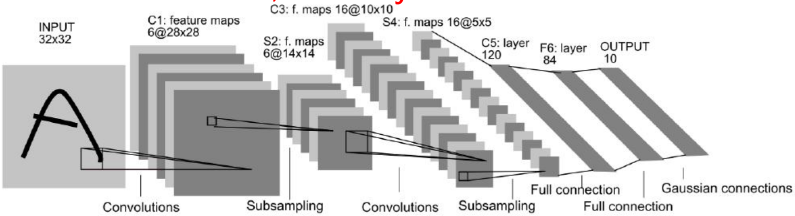


LeCun 1998

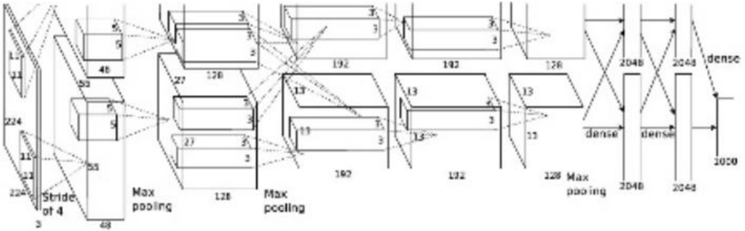
Properties

- Invariance to small transformations (over the region)
- Reducing the number of weights

Convolutional networks, summary



LeCun 1998



Kryzhevsky et al. 2012

Properties

- Invariance to small transformations (over the region)
- Reducing the number of weights
- Usually many convolutional layers

Overview

Manifesto for Deep

Bengio, Hinton 2006

1. Grand goal: AI

2. Requisites
   - Computational efficiency
   - Statistical efficiency
   - Prior efficiency: architecture relies on human labor

3. Abstraction is mandatory

Manifesto for Deep

Bengio, Hinton 2006

1. Grand goal: AI

2. Requisites
   - Computational efficiency
   - Statistical efficiency
   - Prior efficiency: architecture relies on **student** labor

3. Abstraction is mandatory

Manifesto for Deep

Bengio, Hinton 2006

1. Grand goal: AI

2. Requisites
   - Computational efficiency
   - Statistical efficiency
   - Prior efficiency: architecture relies on **student** labor
3. Abstraction is mandatory
4. Compositionality principle:

Manifesto for Deep

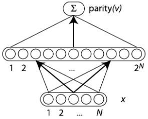
Bengio, Hinton 2006

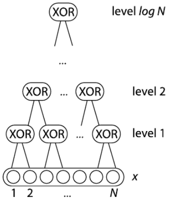
1. Grand goal: AI

2. Requisites
   - Computational efficiency
   - Statistical efficiency
   - Prior efficiency: architecture relies on **student** labor
3. Abstraction is mandatory
4. Compositionality principle:
   build skills on the top of simpler skills **Piaget**

The importance of being deep

A toy example: *n*-bit parity Hastad 1987





Pros: efficient representation

Deep neural nets are (exponentially) more compact

Cons: poor learning

- More layers $\rightarrow$ more difficult optimization problem
- Getting stuck in poor local optima.

Overcoming the learning problem

Long Short Term Memory

- Jurgen Schmidhuber (1997).
  Discovering Neural Nets with Low Kolmogorov Complexity and High Generalization Capability.
  Neural Networks.

Deep Belief Networks

- Geoff. Hinton and S. Osindero and Yeh Weh Teh (2006).
  A fast learning algorithm for deep belief nets.
  Neural Computation.

Auto-Encoders

- Yoshua Bengio and P. Lamblin and P. Popovici and H. Larochelle (2007).
  Greedy Layer- Wise Training of Deep Networks.
  Advances in Neural Information Processing Systems

Auto-encoders

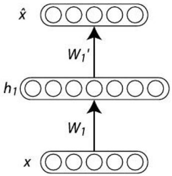
$$\mathcal{E} = \{(\mathbf{x}_i, y_i), x_i \in \mathbb{R}^d, y_i \in \mathbb{R}, i = 1 \dots n\}$$

First layer

$$\mathbf{x} \longrightarrow h_1 \longrightarrow \hat{\mathbf{x}}$$

- An auto-encoder:

$$\text{Find } W^* = \arg\min_W \left(\sum_i ||W^t o W(\mathbf{x}_i) - x_i||^2 \right)$$



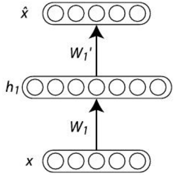
(*) Instead of min squared error, use cross-entropy loss:

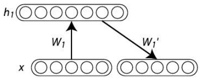
$$\sum_j \mathbf{x}_{i,j} \log \hat{\mathbf{x}}_{i,j} + (1 - \mathbf{x}_{i,j}) \log (1 - \hat{x}_{i,j})$$

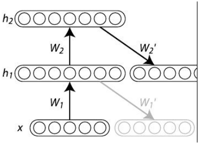
Auto-encoders, 2

First layer $\mathbf{x} \longrightarrow \mathbf{h}_1 \longrightarrow \hat{\mathbf{x}}$

Second layer $\mathbf{h}_1 \longrightarrow \mathbf{h}_2 \longrightarrow \hat{\mathbf{h}_1}$ same, replacing $\mathbf{x}$ with $\mathbf{h}_1$







Discussion

Layerwise training

- Less complex optimization problem (compared to training all layers simultaneously)
- Requires a local criterion: e.g. reconstruction
- Ensures that layer i encodes same information as layer $i+1$
- But in a more abstract way:
  layer 1 encodes the patterns formed by the (descriptive) features
  layer 2 encodes the patterns formed by the activation of the previous patterns
- When to stop ? trial and error.

Discussion

Layerwise training

- Less complex optimization problem (compared to training all layers simultaneously)
- Requires a local criterion: e.g. reconstruction
- Ensures that layer i encodes same information as layer $i + 1$
- But in a more abstract way:
  layer 1 encodes the patterns formed by the (descriptive) features
  layer 2 encodes the patterns formed by the activation of the previous patterns
- When to stop ? trial and error.

Now pre-training is almost obsolete

Gradient problems better understood

- Initialization
- New activation — ReLU
- Regularization
- Mooore data
- Better optimization algorithms

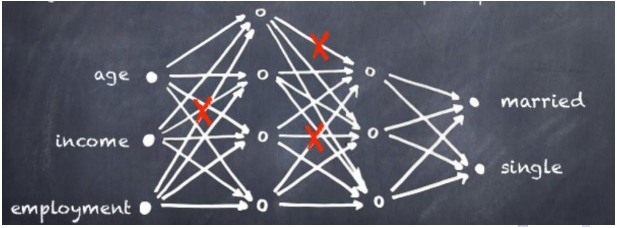
Dropout

Why

- Ensemble learning is effective
- But training several Deep NN is too costly
- The many neurons in a large DNN can "form coalitions".
- Not robust !

How

- During training
  - For each hidden neuron, each sample, each iteration
  - For each input (of this hidden neuron)
  - with probability p (.5), zero the input
  - (double the # iterations needed to converge)
- During validation/test
  - use all input
  - rescale the sum ($\times p$) to preserve average



Recommendations

Ingredients

as of 2015

- ReLU non-linearities
- Cross-entropy loss for classification
- Stochastic Gradient Descent on minibatches
- Shuffle the training samples
- Normalize the input variables (zero mean, unit variance)
- If you cannot overfit, increase the model size; if you can, regularize.

Regularization

- L_2 penalizes large weights
- L_1 penalizes non-zero weights

Adaptive learning rate

- adjusted per neuron to fit the moving average of the last gradients

Hyper-parameters

- Grid search
- Continue training the most promising model

More: Neural Networks, Tricks of the Trade (2012 edition) G. Montavon, G. B. Orr, and K-R Mller eds.

Not covered

- Long Short Term Memory
- Restricted Boltzman Machines
- Natural gradient

Overview

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton,
Advances in Neural Information Processing Systems 2012

ImageNet

- 15M images
- 22K categories
- Images collected from Web
- Human labelers (Amazons Mechanical Turk crowd-sourcing)
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010)
    - 1K categories
    - 1.2M training images (1000 per category)
    - 50,000 validation images
    - 150,000 testing images
- RGB images with variable-resolution

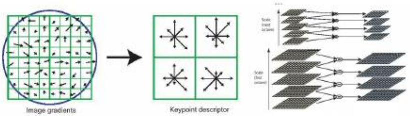
ImageNet

Evaluation

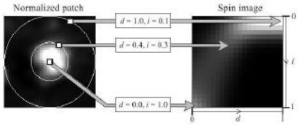
- Guess it right — top-1 error
- Guess the right one among the top 5 — top-5 error

What is new ?

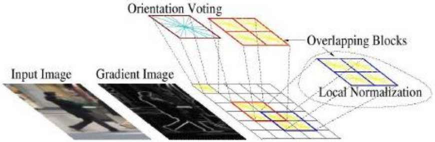
Former state of the art



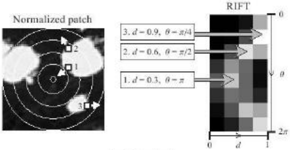
SIFT



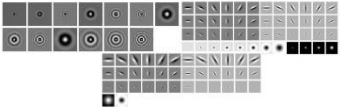
Spin image



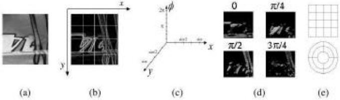
HoG



RIFT



Textons



GLOH

SIFT: scale invariant feature transform
HOG: histogram of oriented gradients

Textons: “vector quantized responses of a linear filter bank”

What is new, 2

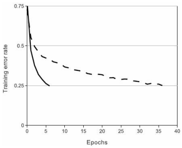
Traditional approach

→ Manually crafted features → Trainable classifier

Deep learning

→ Trainable feature extractor → Trainable classifier

DNN. 1, Tractability



4 layers convolutional

Activation function

- On CIFAR-10: Relu 6 times faster than tanh

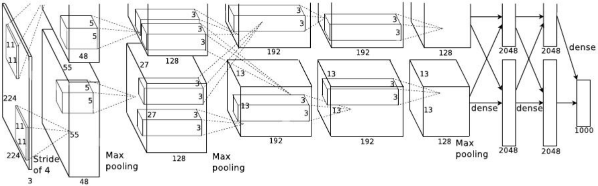
Data augmentation

learn 60 million parameters; 650,000 neurons

- Translation and horizontal symmetries
- Alter RGB intensities
  - PCA, with (p, λ) eigen vector, eigen value
  - Add: $(p_1, p_2, p_3) \times (\alpha\lambda_1, \alpha\lambda_2, \alpha\lambda_3)^t$ to each image, with $\alpha \sim U[0, 1]$

DNN. 2, Architecture

- 1st layer: 96 kernels ($11 \times 11 \times 3$; stride 3)
- Normalized, pooled
- 2nd layer: 256 kernels ($5 \times 5 \times 48$).
- Normalized, pooled
- 3rd layer: 384 kernels ($3 \times 3 \times 256$)
- 4th layer: 384 kernels ($3 \times 3 \times 192$)
- 5th layer: 256 kernels ($3 \times 3 \times 192$)
- followed by 2 fully connexted layers, 4096 neurons each



DNN. 3, Details

Pre-processing

- Variable-resolution images $\to$ i) down-sampling; ii) rescale
- subtract mean value for each pixel

Results on the test data

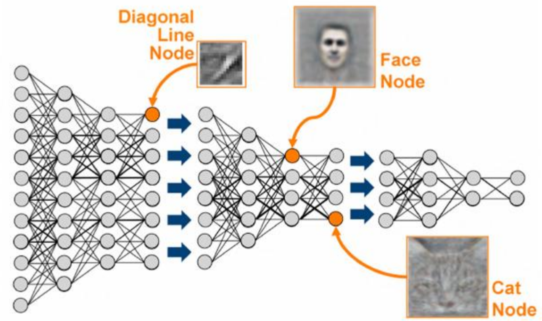
- top-1 error rate: 37.5%
- top-5 error rate: 17.0%

Results on ILSVRC-2012 competition

- 15.3% accuracy
- 2nd best team: **26.2% accuracy**

"Understanding" the result

Interpreting a neuron:
Plotting the input (image) which maximally excites this neuron.



20 millions image from YouTube

"Understanding" the result, 2

Interpreting the representation: Plotting the induced topology

http://cs.stanford.edu/people/karpathy/cnnembed/

Overview

Natural Language Processing

Dimensionality: 20K (speech) 50K (Penn TB) 500K (big vocab) 13M (Google)

Bag-of-words

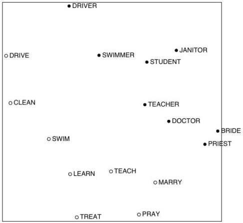
motel [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0] AND
hotel [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0] = 0

Latent representations

Latent Semantic analysis

- Input: matrix (documents $\times$ words)
- *You know a word by the company it keeps* — Firth 57
- Dimensionality reduction

– high dimensional, sparsity issue, scales quadratically, update problematic



also, something non additive is needed: not bad $\neq$ not + bad

NLP: which learning criterion ?

Criterion for learning

1. predict a linguistic label
2. predict a class — opinion mining
3. predict the neighborhood of words

The labelling cost

- 1, 2 requires labels — ground truth
- 3: can be handled in an unsupervised way — tons of data !

Criterion for evaluation

- evaluate relationships

Continuous language models

Bengio et al. 2001

Principle

- Input: 10,000-dim boolean input (words)
- Hidden layer: 500 continuous neurons
- Output: from a text window $(w_i \ldots w_{i+2k})$, predict
  - The grammatical tag of the central word w_{i+k}
  - Other: see next

Trained embeddings

- Hidden layer defines a mapping from a text window onto $\mathbb{R}^{500}$
- Applicable to any discrete space

Continuous language models

The window approach

- Fixed size window works fine for some tasks
- Does not deal with long-range dependencies

The sentence approach

- Feed the whole sentence to the network
- Convolutions to handle variable-length inputs
- Convert network outputs into probabilities softmax $p(i) = \frac{exp(f(i,x,\theta))}{\sum_j exp(f(j,x,\theta))}$
- Maximize log likelihood

$$\text{Find } \theta^* = \arg\max log(p(y|\mathbf{x},\theta))$$

Results

- Small improvements
- 15% of most frequent words in the dictionary are seen 90% of the time...

Going unlabelled

Collobert et al. 08

Idea: a lesion study

- Take a sentence from Wikipedia: label *true*
- Replace middle word with random word: label *false*
- Tons of labelled data, 0-cost labels
- Captures semantics and syntax

Training Language Model

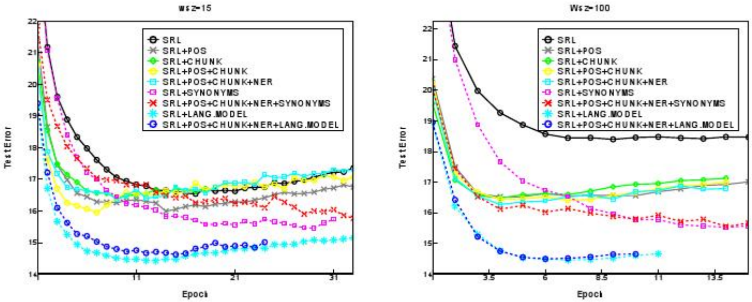
- Two window approach (11) networks (100HU) trained on two corpus:
  - ⋆ LM1: Wikipedia: **631M** of words
  - ⋆ LM2: Wikipedia+Reuters RCV1: **631M+221M=852M** of words
- Massive dataset: cannot afford classical training-validation scheme
- Like in biology: breed a couple of network lines
- Breeding decisions according to 1M words validation set
- LM1
  - ⋆ order dictionary words by frequency
  - ⋆ increase dictionary size: 5000, $10,000$, $30,000$, $50,000$, $100,000$
  - ⋆ 4 weeks of training
- LM2
  - ⋆ initialized with LM1, dictionary size is $130,000$
  - ⋆ 30,000 additional most frequent Reuters words
  - ⋆ 3 additional weeks of training

Unsupervised Word Embeddings

france	jesus	xbox	reddish	scratched	megabits
454	1973	6909	11724	29869	87025
austria	god	amiga	greenish	nailed	octets
belgium	sati	playstation	bluish	smashed	mb/s
germany	christ	msx	pinkish	punched	bit/s
italy	satan	ipod	purplish	popped	baud
greece	kali	sega	brownish	crimped	carats
sweden	indra	psNUMBER	greyish	scraped	kbit/s
norway	vishnu	hd	grayish	screwed	megahertz
europe	ananda	dreamcast	whitish	sectioned	megapixels
hungary	parvati	geforce	silvery	slashed	gbit/s
switzerland	grace	capcom	yellowish	ripped	amperes

Continuous language models, Collobert et al. 2008

MTL: Semantic Role Labeling



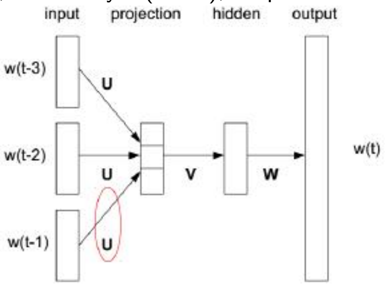
- We get: 14.30%. State-of-the-art: 16.54% – Pradhan et al. (2004)
- 250× faster than state-of-the-art. ∼ 0.01s to label a WSJ sentence.

Word to Vec

Mikolov et al., 13, 14
https://code.google.com/p/word2vec/

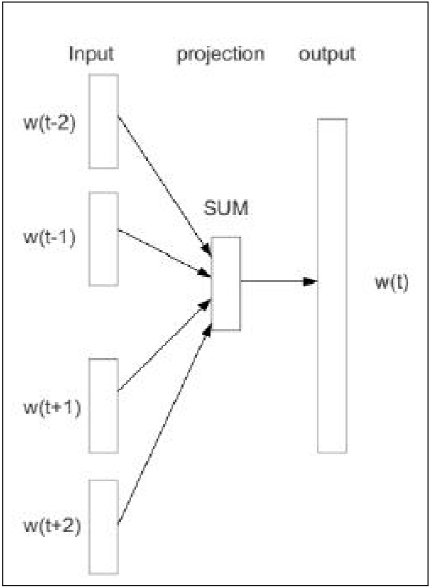
Continuous bag of words model

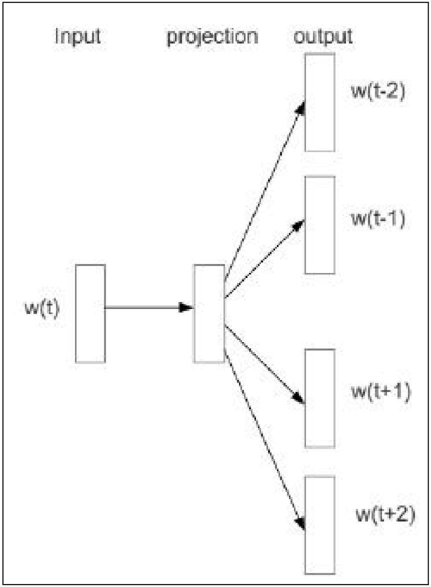
- input, projection layer, hidden layer (linear), output



- Adds input from window to predict the current word
- Shares the weights for different positions
- Very efficient

Word to Vec, two models





Computational aspects

Model	*Vector Dimensionality*	*Training Words*	*Training Time*	*Accuracy [%]*
Collobert NNLM	50	660M	2 months	11
Turian NNLM	200	37M	few weeks	2
Mnih NNLM	100	37M	7 days	9
Mikolov RNNLM	640	320M	weeks	25
Huang NNLM	50	990M	weeks	13
Skip-gram (hier.s.)	1000	6B	hours	66
CBOW (negative)	300	1.5B	**minutes**	**72**

Tricks

- Undersample frequent words (the, is, ...)
- Linear hidden layer
- Negative sampling: only the output neuron that represents the positive class + few randomly sampled neurons are evaluated
- output neurons: independent logistic regression classifiers
- $\rightarrow$ training speed independent of vocabulary size

Word vectors – nearest neighbors

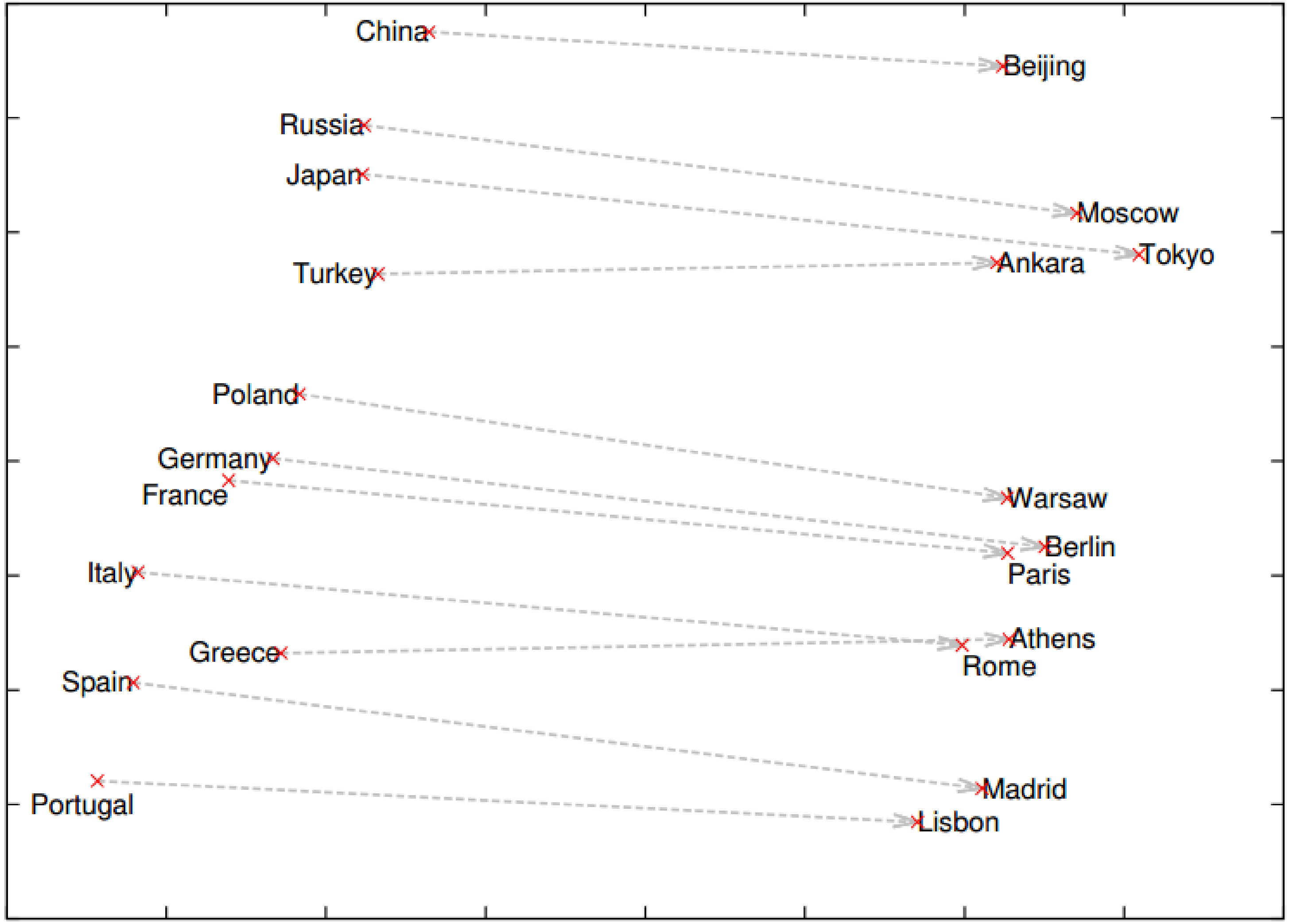
	Redmond	Havel	graffiti	capitulate
Collobert NNLM	conyers lubbock keene	plauen dzerzhinsky osterreich	cheesecake gossip dioramas	abdicate accede rearm
Turian NNLM	McCarthy Alston Cousins	Jewell Arzu Ovitz	gunfire emotion impunity	- - -
Mnih NNLM	Podhurst Harlang Agarwal	Pontiff Pinochet Rodionov	anaesthetics monkeys Jews	Mavericks planning hesitated
Skip-gram (phrases)	Redmond Wash. Redmond Washington Microsoft	Vaclav Havel president Vaclav Havel Velvet Revolution	spray paint grafitti taggers	capitulation capitulated capitulating

- More training data helps the quality a lot!

Word vectors – more examples

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

Word vectors – visualization using PCA

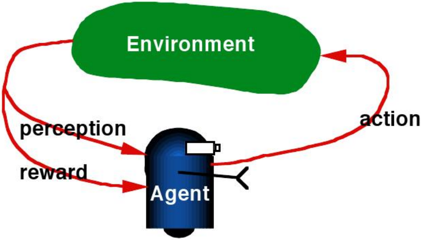


Overview

Deep Reinforcement Learning

Reinforcement Learning in one slide

- State space $\mathcal{S}$
- Action space $\mathcal{A}$
- Transition model $p(s, a, s') \mapsto [0, 1]$
- Reward $r(s)$ bounded



Value functions and policies

$$V^{\pi}(s) = r(s) + \gamma \sum_{s'} p(s, \pi(s), s') V^{\pi}(s')$$

$$V^{*}(s) = \max_{\pi} V^{\pi}(s')$$

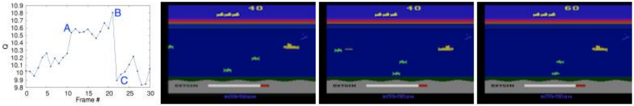
$$\pi^{*}(s) = \underset{a \in \mathcal{A}}{argmax} \left\{ \sum_{s'} p(s, a, s') V^{*}(s') \right\}$$

Playing Atari

Mnih et al. 2015

Input: 4 consecutive frames

- 84 $\times$ 84 (reduced, gray-scaled) pixels $\times$ 4 (last four frames)



Architecture

- 1st hidden layer : 16 8 $\times$ 8 filters with stride 4, ReLU
- 2nd hidden layer : 32 4 $\times$ 4 filters with stride 2, ReLU
- last hidden layer, fully connected, 256 ReLU
- output layer: fully connected, one output per valid action $\#\mathcal{A}$ in 4..18
- decision: select action with max. output

Playing Atari, 2

Training

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation
  end for
end for

$$y = Q(s, a, \theta)$$

$$Q(s, a, \theta) = \mathbb{E}\left[r(s, a) + \arg\max_{a \in \mathcal{A}}\{Q(s', a', \theta)\}\right]$$

Playing Atari, 3

Tricks

- Experience replay: store $\{(s_t, a_t, r_t, s_{t+1}\}$
- Inner loop, minibatch of 32 uniformly drawn samples (avoids correlated updates)
- All positive rewards = 1; negative = -1
- Select an action every 4 time frames and apply it for 4 time frames

Results

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Table 1: The upper table compares average total reward for various learning methods by running an ϵ-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an ϵ-greedy policy with $\epsilon = 0.05$.

Playing Atari, 4

What is impressive

- Several games
- Single architecture
- Same hyper-parameters !!!

Overview

Do Deep Nets Really Need To Be Deep ?

Caruana
http://research.microsoft.com/apps/video/dl.aspx?id=232373

Principle

- Train an ensemble of deep NN
- Use the ensemble as teacher
- Find (optimize) a shallow NN to approximate the teacher

TIMIT

Contents

TIMIT speech corpus: 462 speakers in the training set, 50 speakers in validation set, 24 speakers in test set.

Pre-processing

The raw waveform audio data were pre-processed using 25ms Hamming window shifting by 10ms to extract Fourier-transform-based filter-banks with 40 coefficients (plus energy) distributed on a mel-scale, together with their first and second temporal derivatives. We included +/- 7 nearby frames to formulate the final 1845 dimension input vector. The data input features were normalized by subtracting the mean and dividing by the standard deviation on each dimension. All 61 phoneme labels are represented in tri-state, i.e., three states for each of the 61 phonemes, yielding target label vectors with 183 dimensions for training. At decoding time these are mapped to 39 classes as in [13] for scoring.

Results

NNs

- DNN, three fully connected feedforward hidden layers (2000 rectified linear units per layer).
- CNN convolutional architecture
- Shallow neural nets with 8000, 50 000, and 400 000 hidden units.

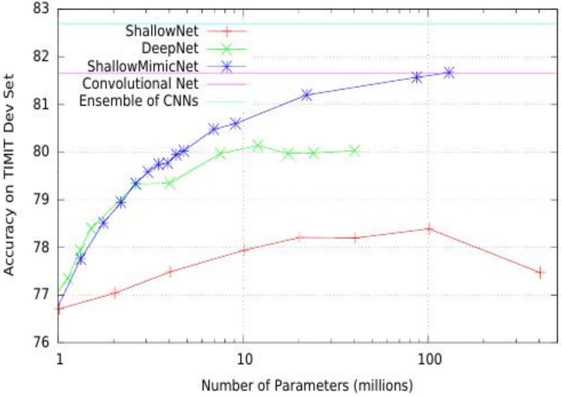
Architecture of shallow NN

- A linear bottleneck followed by a non-linear layer

	Architecture	# Param.	# Hidden units	PER
SNN-8k	8k + dropout trained on original data	~12M	~8k	23.1%
SNN-50k	50k + dropout trained on original data	~100M	~50k	23.0%
SNN-400k	250L-400k + dropout trained on original data	~180M	~400k	23.6%
DNN	2k-2k-2k + dropout trained on original data	~12M	~6k	21.9%
CNN	c-p-2k-2k-2k + dropout trained on original data	~13M	~10k	**19.5%**
ECNN	ensemble of 9 CNNs	~125M	~90k	**18.5%**
SNN-MIMIC-8k	250L-8k no convolution or pooling layers	~12M	~8k	**21.6%**
SNN-MIMIC-400k	250L-400k no convolution or pooling layers	~180M	~400k	**20.0%**

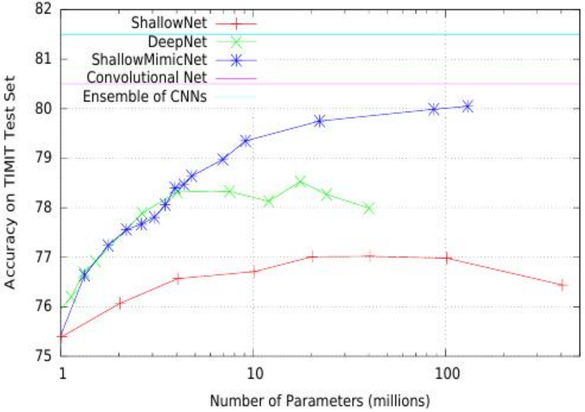
What matters is not the deep architecture, after all...

On the validation set



What matters is not the deep architecture, after all...

On the test set



Why does this work ?

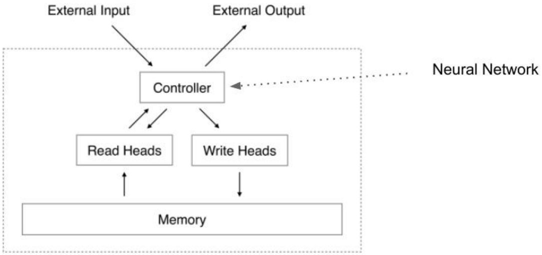
Why does it work

- Label much more informative: input $(\mathbf{x}, p)$ with p the log probability of each class (before the softmax).
  This gives much more information than the softmax.
- Data augmentation: teacher can label anything, no extra label cost.

Not covered

Neural Turing Machines Alex Graves

http://msrvideo.vo.msecnd.net/rmcvideos/260037/dl/260037.mp4



Not covered, 2

Morphing of representations

Leon Gatys

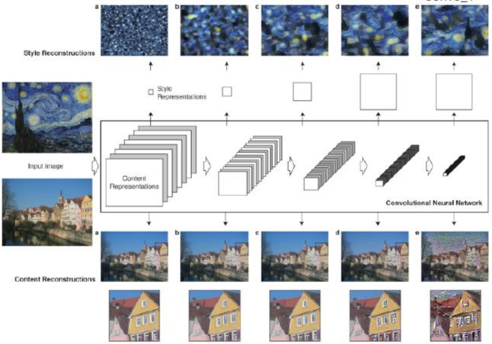
conv1_1

conv1_1
conv2_1

conv1_1
conv2_1
conv3_1

conv1_1
conv2_1
conv3_1
conv4_1

conv1_1
conv2_1
conv3_1
conv4_1
conv5_1

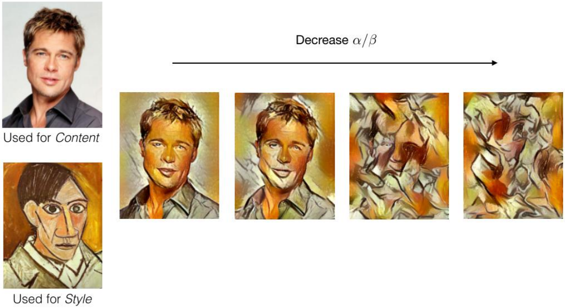


conv1_1 conv2_1 conv3_1 conv4_1 conv5_1

Not covered, 2

Morphing of representations

Leon Gatys



Not covered, 3

Spatial transformer

Jaderberg et al., 15

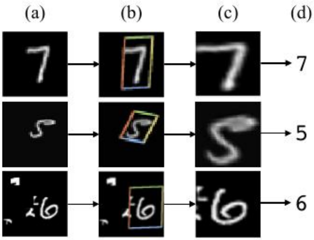


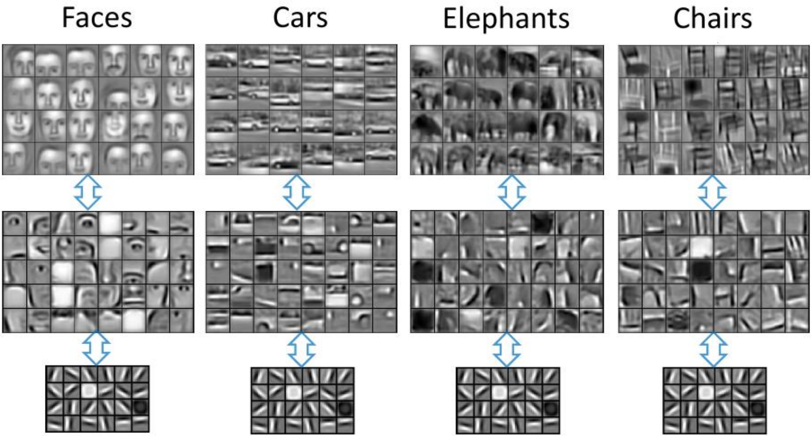
Figure 1: The result of using a spatial transformer as the first layer of a fully-connected network trained for distorted MNIST digit classification. (a) The input to the spatial transformer network is an image of an MNIST digit that is distorted with random translation, scale, rotation, and clutter. (b) The localisation network of the spatial transformer predicts a transformation to apply to the input image. (c) The output of the spatial transformer, after applying the transformation. (d) The classification prediction produced by the subsequent fully-connected network on the output of the spatial transformer. The spatial transformer network (a CNN including a spatial transformer module) is trained end-to-end with only class labels – no knowledge of the groundtruth transformations is given to the system.

Overview

DNN as a representation builder

Features learned from large datasets — e.g. ImageNet

- Can be useful for many other problems
- As initialization for another DNN
  - Higher layers are more specific: can be tuned on *your* data while reusing general features from lower layers (e.g. edge detectors)
- As indices for a large db — see Locally Sensitive Hashing
- As a feature layer for e.g. SVMs



DNN as a massive computer science technology

DNN training is made possible

- With tons of data
- With specific computational platforms

The entry ticket is expensive

- See TensorFlow

DNN as a functional primitive

Huge models

Published source	Application	Params
Hinton et al., 2006	Digit images	1.6mn
Hinton & Salakhutdinov	Face images	3.8mn
Salakhutdinov & Hinton	Sem. hashing	2.6mn
Ranzato & Szummer	Text	3mn
Using GPU (Raina et al., 2009)		100mn

Model	*Vector Dimensionality*	*Training Words*	*Training Time*	*Accuracy [%]*
Collobert NNLM	50	660M	2 months	11
Turian NNLM	200	37M	few weeks	2
Mnih NNLM	100	37M	7 days	9
Mikolov RNNLM	640	320M	weeks	25
Huang NNLM	50	990M	weeks	13
Skip-gram (hier.s.)	1000	6B	hours	66
CBOW (negative)	300	1.5B	**minutes**	**72**

Next frontiers

Questions

- Interpretation
- Do we still need (relational) logic ?

Next applications

- Signal processing ?