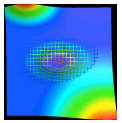


Tutorial on Deep Learning

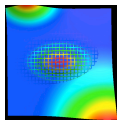
Brief introduction to Deep Neural Networks

Bálint Daróczy

Informatics Laboratory

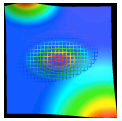
MTA SZTAKI

07.07.2017



Overview

- Brief introduction
- Generalization theory
- Deep vs. shallow models
- Feed-Forward networks: MLP, CNN
- Recurrent Neural Networks: simple RNN, LSTM, GRU
- Generative Neural Networks: VAE, GAN
- Example 1: Image caption via Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) with Word Embedding (WE)
- Example 2: Object detection
- Some unsolved problems



Classification

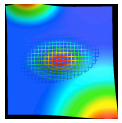
Let be a finite set $X=\{x_1,..,x_T\}$ in R^d and for each point a label $y=\{y_1,..,y_T\}$ usually in {-1,1}. The problem of binary classification is to find a particular f(x) which approximate y over X.

How to measure the performance of the approximation?
How to choose the function class?
How to find a particular element in the chosen function class?
How to generalize?

E.g. the problem of learning a half-space or a linear separator. The task is to find a d-dimensional vector w, if one exists, and a threshold b such that

$$w \cdot x_i > b \text{ for each } x_i \text{ labelled} +1$$
$$w \cdot x_i < b \text{ for each } x_i \text{ labelled} -1$$

A vector-threshold pair, (w, b), satisfying the inequalities is called a linear separator -> dual problem: high dimensional learning via kernels (inner products)



Vapnik-Chervonenkis theorem 1971

The theorem explains the connection between generalisation, training set selection and model selection.

Generalisation gap: the performance difference between the training set (empirical risk) and the distribution (true risk):

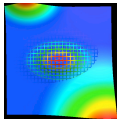
$$P(\sup_{f \in \mathcal{F}} \mid R_{emp}(f) - R_{true}(f) \mid > \epsilon) \leq 8\mathcal{S}(\mathcal{F}, T)\mathrm{e}^{-\frac{T\epsilon^2}{32}}$$

By VC-theorem the gap is upper bounded and depends only on the size of the training set (T) and the separating capability of the chosen function class (F) measured by the shattering coefficient S(F,T): the maximum number of different labelings the function class F can realize over T samples (in binary classification ideally 2^T).

The VC-dimension of a function class VC(F) is the cardinality of the largest set in the d-dimensional space which can be separated correctly (or shattered) with any label set. According to Sauer's lemma [Sauer, 1972] the shattering coefficient is upper bounded as

$$\mathcal{S}(\mathcal{F}, T) \leq (1 + T)^{VC(\mathcal{F})}$$

Linear separator: VCdim = d+1 (Radon theorem). Is it a sharp bound?



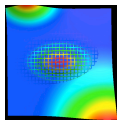
Vapnik-Chervonenkis theorem

1) Optimize for low empirical risk on the largest possible training set
2) Choose a function class with low shattering coefficient (low VC-dimension)
3) Evaluate on a separate validation set! Let us take a disjoint test set, then according to the proof in [Devroye et al., 1996]

$$P(\sup_{f \in \mathcal{F}} \mid R_{emp}(f) - R_{true}(f) \mid > \epsilon) \leq 2P(\sup_{f \in \mathcal{F}} \mid R_{emp}(f) - R'_{emp}(f) \mid > \frac{\epsilon}{2}) / \quad (4)$$

If we evaluate on a separate test set we have a new upper bound!
Limitations?

1) Fixed distribution … (e.g.?)
2) Really high for complex models ☹ or…?
3) Are we even close to the optimal during optimization?
   A very good start: [Bottou & Bousquet, 2007]



Deep vs. shallow models

Hypothesis: deep, hierarchical models can be exponentially more efficient than a shallow one [Bengio et al. 2009, Le Roux and Bengio, 2010, Delalleau and Bengio, 2011 etc.]

[Delalleau and Bengio, 2011]: deep sum-product network may require exponentially less units to represent the same function compared to a shallow sum-product network.

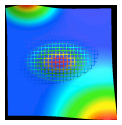
[Lin & Tegmark, 2016]: efficient "flattening" of deep architectures is exponentially expensive even for simple cases

What is the depth of a Neural Network (NN)?

In case of feed forward networks, the number of multiple nonlinear layers between the input and the output layer.

We will see, in case of recurrent NN this definition does not apply.

But before: What is NN?



Neural Networks

Key ingredients:

- Wiring: units and connections

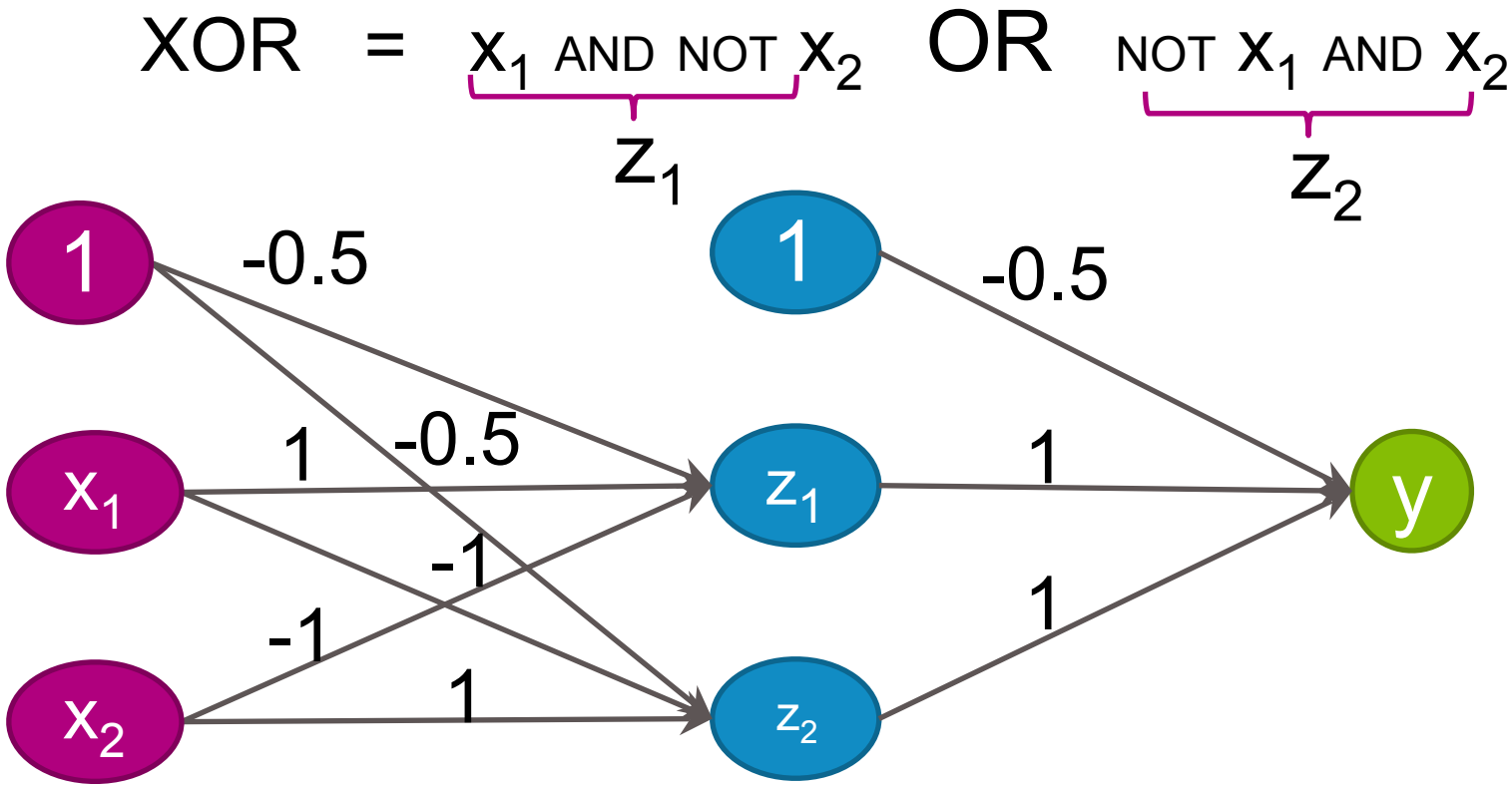
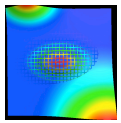


Fig.: Danny Bickson



Activation functions

Identity	$f(x) = x$	$f'(x) = 1$
Binary step	$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH	$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan	$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear	$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus	$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$
Bent identity	$f(x) = \frac{\sqrt{x^2+1}-1}{2} + x$	$f'(x) = \frac{x}{2\sqrt{x^2+1}} + 1$
SoftExponential	$f(\alpha, x) = \begin{cases} -\frac{\log_e(1-\alpha(x+\alpha))}{\alpha} & \text{for } \alpha < 0 \\ x & \text{for } \alpha = 0 \\ \frac{e^{\alpha x}-1}{\alpha} + \alpha & \text{for } \alpha > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \frac{1}{1-\alpha(\alpha+x)} & \text{for } \alpha < 0 \\ e^{\alpha x} & \text{for } \alpha \geq 0 \end{cases}$
Sinusoid	$f(x) = \sin(x)$	$f'(x) = \cos(x)$
Sinc	$f(x) = \begin{cases} 1 & \text{for } x = 0 \\ \frac{\sin(x)}{x} & \text{for } x \neq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x = 0 \\ \frac{\cos(x)}{x} - \frac{\sin(x)}{x^2} & \text{for } x \neq 0 \end{cases}$
Gaussian	$f(x) = e^{-x^2}$	$f'(x) = -2xe^{-x^2}$

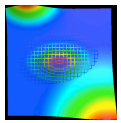
Usually:
f : R^d -> R

Output of a unit

- linear/ non-linear
- bounded/ unbounded
- usually monotonic, but not all

Why so rigid?

Fig.: wikipedia



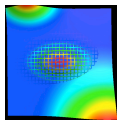
Feed-forward Neural Networks

As Bayesian Networks [Pearl, 2011]:

- The nodes are either input, output or hidden
- Connections between the nodes: directed edges
- usual presumption: finite set of nodes -> finite set of layers (are there any layers?)
- no directed cycles -> directed acyclic graphs (DAG)!
- usual posteriors:
  - linear combination (edge weights) of inputs
  - Activation function

$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$

where $z_i^{(l+1)}$ is the linear combination of the i-th element in the (l+1)-th hidden layer, and f is the non-linear transformation (common: f: R -> R ! When is it not?)



Feed-forward neural networks

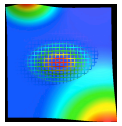
Some common restrictions:

- Disjoint set of nodes -> layers
- Exists an ordering of layers (so ordering of nodes!!)
  - “Causality”: previous layer “causes” the next one
  - each node is connected to
    - Nodes in the previous layer (input nodes)
    - Nodes in the next layer
- continuously differentiable activation functions
- Optimization via previously determined loss function (CDF?)

If it is fully connected:

- Each node in the previous layer is connected
- Each node in the next layer is connected

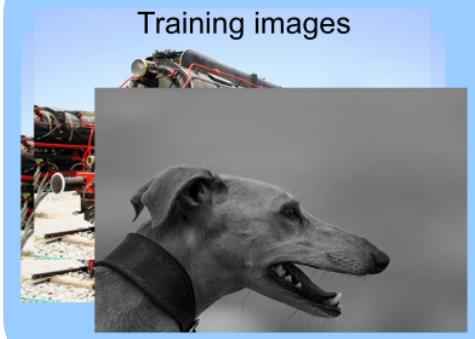
If so, the Network is called Multi-Layer Perceptron (in short MLP)



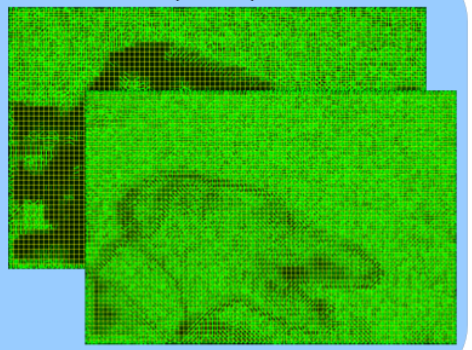
Convolutional Neural Network (CNN)

Histogram of Oriented Gradients (HOG)

Training images

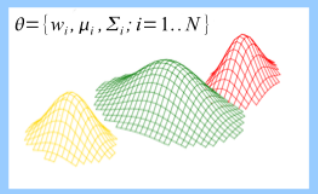


I_X, I_Y



$X = x_1, .., x_T, Y = y_1, .., y_T$

Gaussian Mixture Model



Fisher score

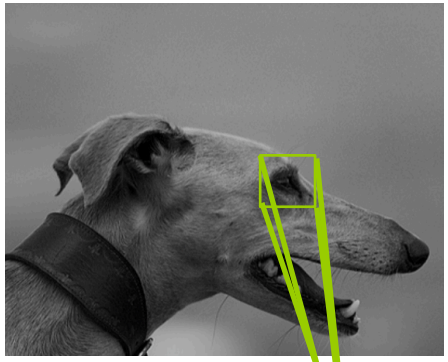
$$U_X = \nabla_\theta \sum_t^T \log P(x_t|\theta), U_Y = \nabla_\theta \sum_t^T \log P(y_t|\theta)$$

Fisher vector

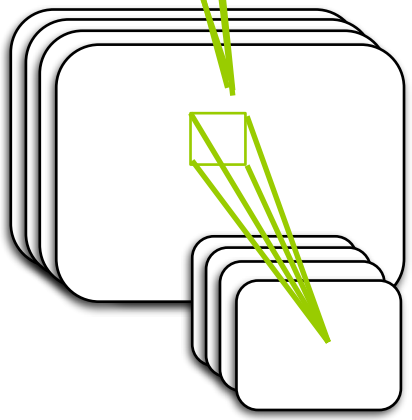
$$G_X = F^{-1/2} U_X, G_Y = F^{-1/2} U_Y$$

Support Vector Machine (SVM)

$$K(X, Y) = U_X^T F^{-1} U_Y = G_X^T G_Y$$



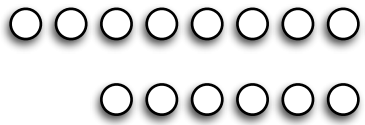
Receptive field



Conv. Layer

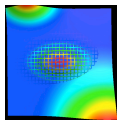
Sub-sampling

....



Fully conn.

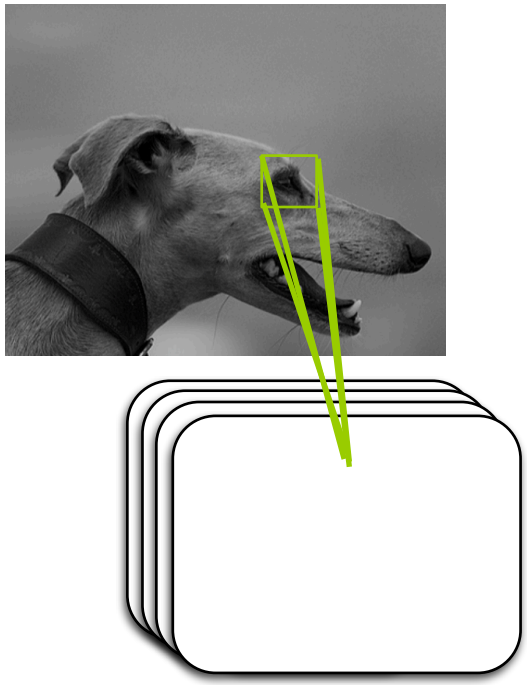
Traditional pattern recognition vs. CNN



Feed-forward Neural Networks

MLP vs. CNN

- Each node is adopted on a subset of the input, but all over the image



It can be interpreted

- Either as a lot of fully connected node with zero weights and there weights where they are non-zero is shared
- Or leave this complicated definition and just simply define it as a convolution over the input:

$$(f*g)(x) = \int f(t)\, g(x-t)dt$$

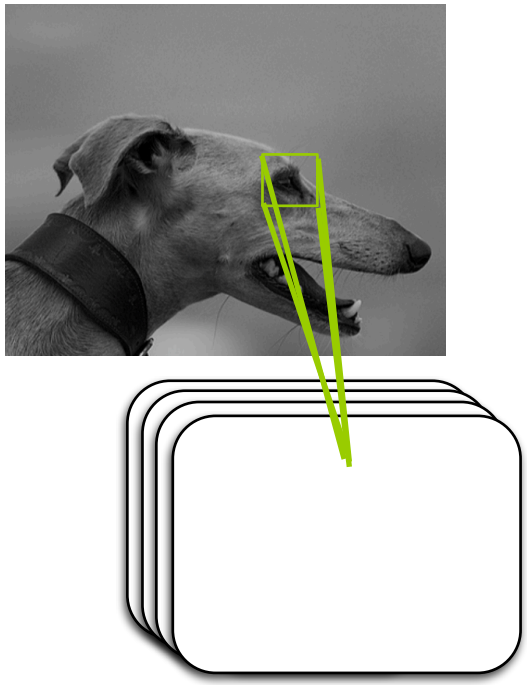
Usually we think of it as a discrete convolution and
in case of images, it is 2D/3D or XD convolution.
What kind of input can we think for 1D?

Feed-forward Neural Networks

MLP vs. CNN

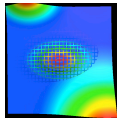
- Each node is adopted on a subset of the input, but all over the image

Example:



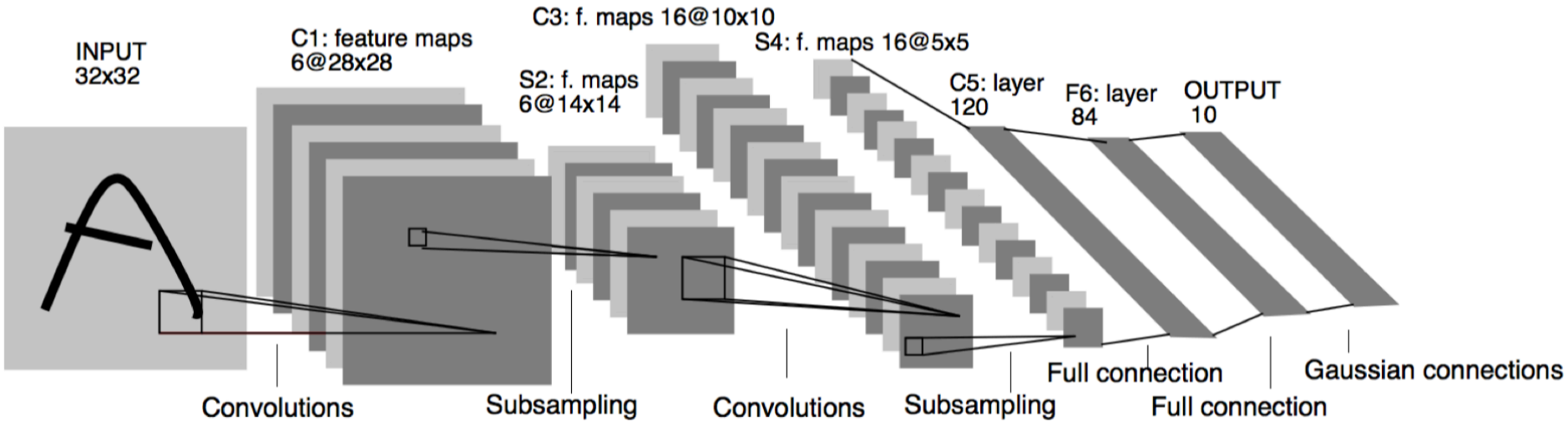
1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

What does it do? What are we changing during optimization?
The main advantages of the CNN over MLP (in practice) is the highly reduced size of the parameter set: 32x32x128 vs. 3x3x128 (128 hidden node and 32x32 input) so the VCdim will be significantly lower ☺️



LeNet-5

LeNet-5 for handwriting recognition in [LeCun et al. 1998]



Key advantages:

- Fixed feature extraction vs. learning the kernel functions
- Spatial structure through sampling
- “Easier to train” due much lesser connection than fully connected

Training: back propagation
By definition it is a feed forward deep neural network.

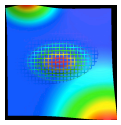
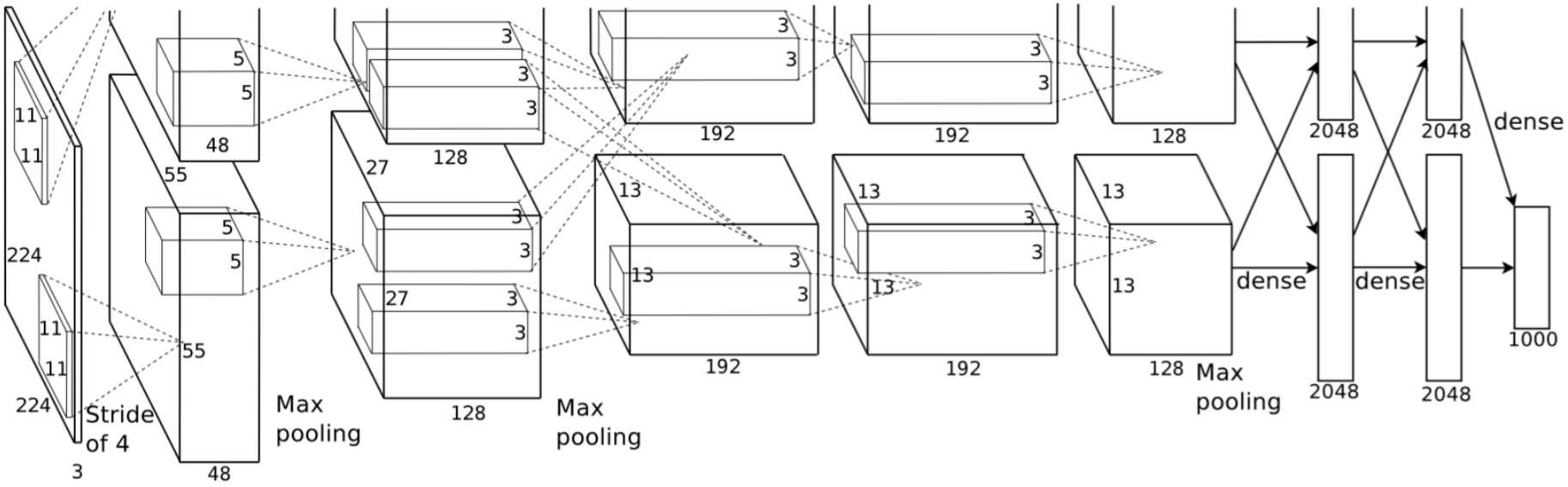


Image classification with CNN

[Krizhevsky et al. 2012]



Advantages over LeNet:

- Local response normalization (normalize over the kernel maps at the same position) over ReLU (-1.2%..1.4% in error rate)
- Overlapping pooling (-0.3..-0.4% in error rate)
- traditional image tricks: augmentation as horizontal flipping, subsampling, PCA over the RGB and noise (-1% in error rate)
- Dropout

Image classification with CNN

[Krizhevsky et al. 2012]

ImageNet: 150k test set and 1.2 million training images with 1000 labels.

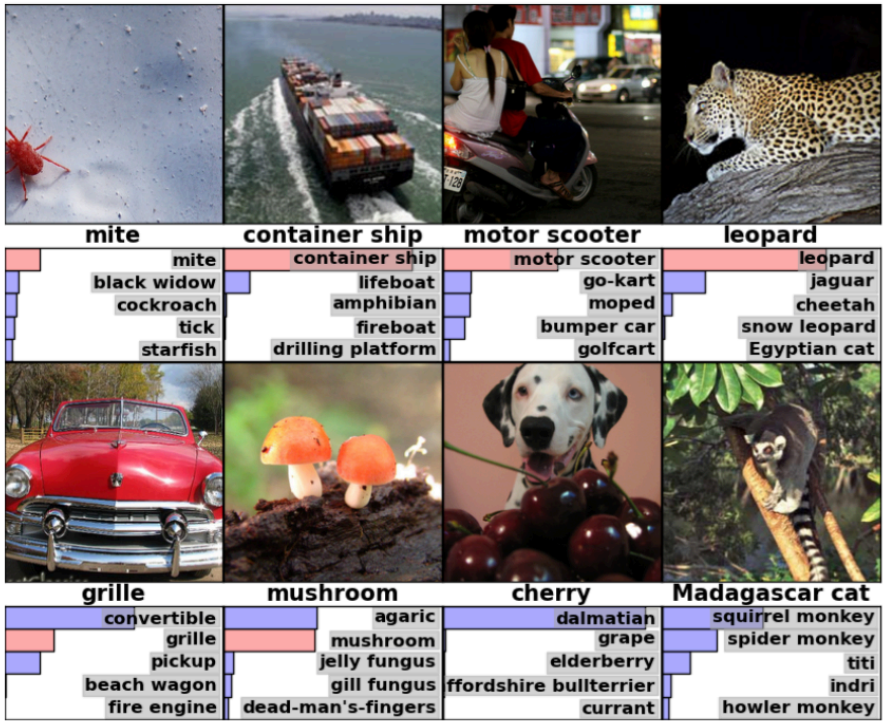
Evaluation: top-1 and top-5 error rate

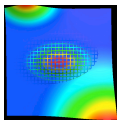
* - additional data

4096 dim. representation per image

5-6 days with 2 Nvidia GTX 580 3GB

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

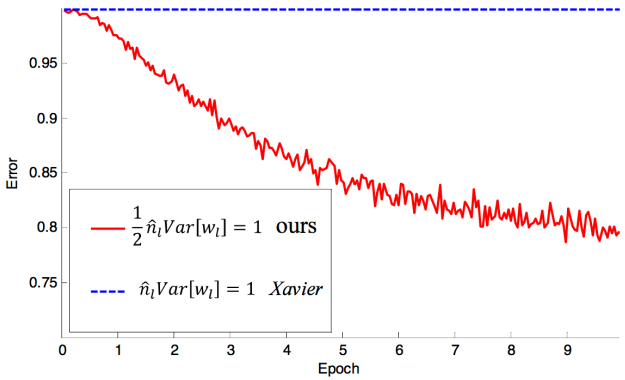




Recent results

[He et al. 2015]: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

Parametric ReLU + zero mean Gaussian init + extreme (at the time…) deep network:
A:19 layers, B: 22 layers, C: 22 layers with more filters



model	top-1	top-5
MSRA [11]	29.68	10.95
VGG-16 [25]	28.07†	9.33†
GoogLeNet [29]	-	9.15
A, ReLU	26.48	8.59
A, PReLU	25.59	8.23
B, PReLU	25.53	8.13
C, PReLU	**24.27**	**7.38**

Training of model C: 8xK40 Nvidia GPU 3..4 weeks (!)

GT: coucal
1: coucal
2: indigo bunting
3: lorikeet
4: walking stick
5: custard apple

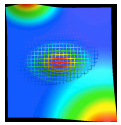
GT: komondor
1: komondor
2: patio
3: llama
4: mobile home
5: Old English sheepdog

GT: yellow lady's slipper
1: yellow lady's slipper
2: slug
3: hen-of-the-woods
4: stinkhorn
5: coral fungus

GT: torch
1: stage
2: spotlight
3: torch
4: microphone
5: feather boa

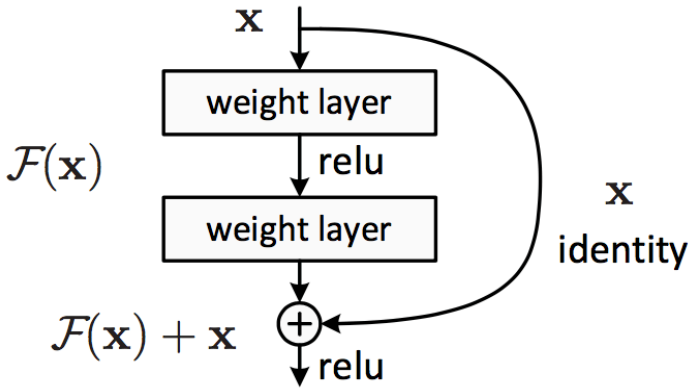
GT: banjo
1: acoustic guitar
2: shoji
3: bow tie
4: cowboy hat
5: banjo

GT: go-kart
1: go-kart
2: crash helmet
3: racer
4: sports car
5: motor scooter

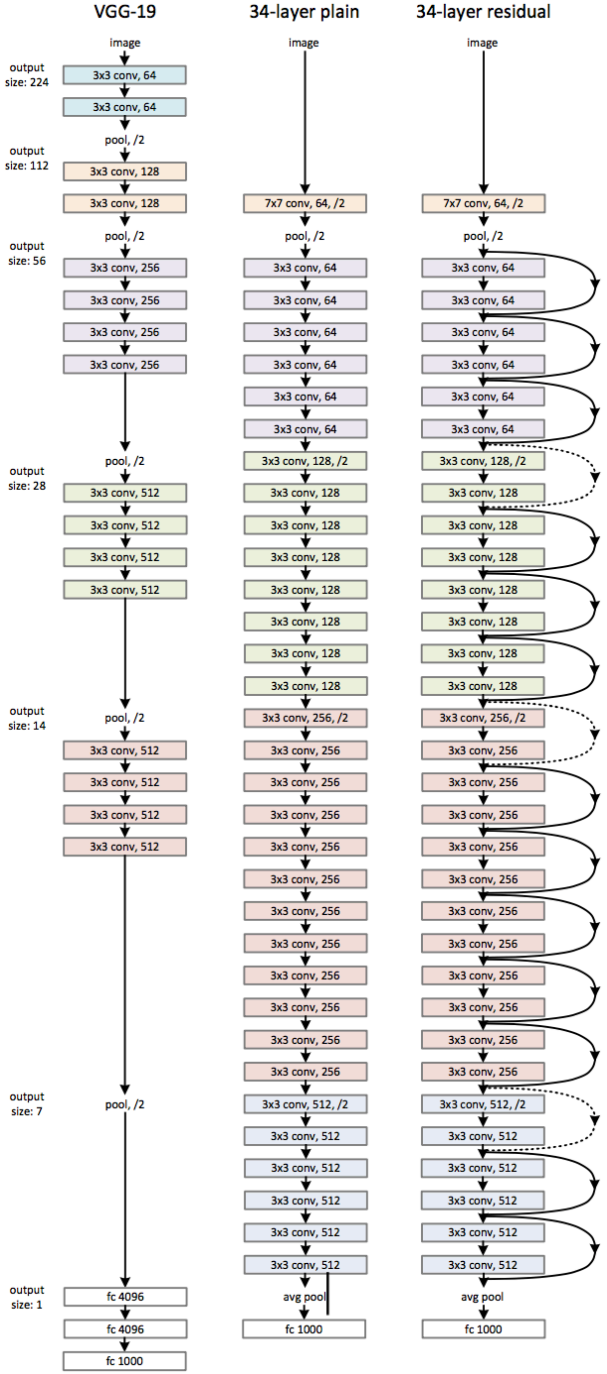


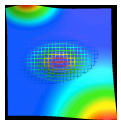
Recent results

[He et al. 2015]: ResNet:"Is learning better networks as easy as stacking more layers?"



model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**





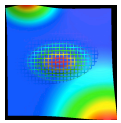
Feed-forward Neural Networks

We already know: VC-dimension (VCdim) of linear separator is d+1

Arbitrary feed-forward neural network [Cover, 1968, Baum & Haussler, 1989, Maas, 1993, Sakurai, 1993] with linear threshold, piecewise linear or sigmoidal activation functions and w parameters:

- with fixed depth VCdim = O(wlogw)
- if the depth is unbounded the VCdim is $O(w^2)$

There exists a feed-forward network with infinite VCdim: a special activation function and the network has only a single hidden layer [Sontag, 1992].

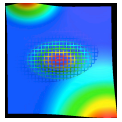


2. Recurrent Neural Networks

- The nodes are either input, output or hidden
- Connections between the nodes: directed edges
- Presumption: finite set of nodes -> finite set of layers (are there any layers?)
- There are some directed cycles -> not a directed acyclic graph anymore … ☹
  - Common: self loops only
- Posteriors are similar to FF

Milestones:

- classic "back-propagation through time" (BPTT) model [Werbos et al., 1988]
- LSTM [Hochreiter & Schmidhuber, 1997]
  - Forget gate [Gers et al., 2000]
- GRU [Cho et al., 2014]



Recurrent Neural Networks (RNN)

Simulates a discrete-time dynamical system [Rumelhart et al. 1986]

Three components:

x_t input in time t

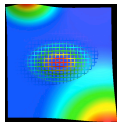
y_t output in time t

h_t hidden state in time t

The connection between the layers are straightforward:

$$\mathbf{h}_t = f_h(\mathbf{x}_t, \mathbf{h}_{t-1})$$
$$\mathbf{y}_t = f_o(\mathbf{h}_t),$$

In comparison to feed forward networks, the main difference is the connection between the current and the last hidden state (a loop in the network) -> can carry along information about the previous inputs! But for how long?



Recurrent Neural Networks (RNN)

Let be given a sequence of samples

$$D = \left\{ \left((\mathbf{x}_1^{(n)}, \mathbf{y}_1^{(n)}), \ldots, (\mathbf{x}_{T_n}^{(n)}, \mathbf{y}_{T_n}^{(n)}) \right) \right\}_{n=1}^{N}$$

Estimation of the parameters (Θ) of RNN is based on minimization of the following additive cost function:

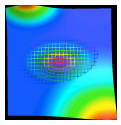
$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} d(\mathbf{y}_t^{(n)}, f_o(\mathbf{h}_t^{(n)}))$$

where

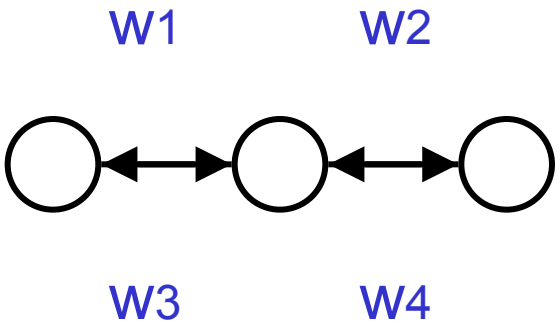
$$\mathbf{h}_t^{(n)} = f_h(\mathbf{x}_t^{(n)}, \mathbf{h}_{t-1}^{(n)})$$

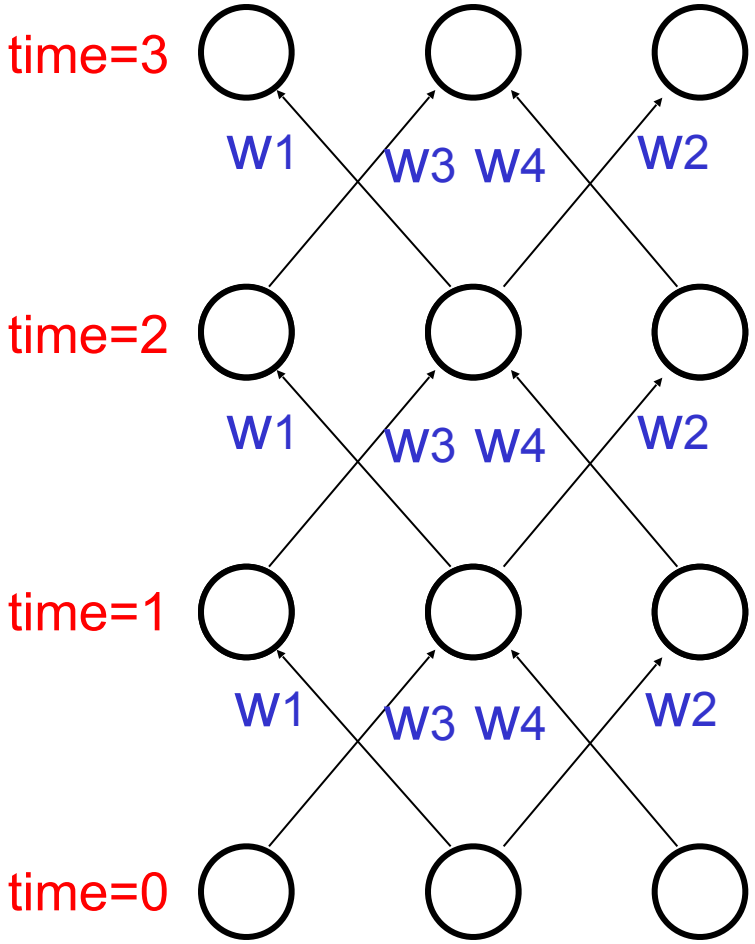
$$\mathbf{h}_0^{(n)} = \mathbf{0}$$

The d(y,f(h)) is some penalty function (divergence, distance etc.).



Recurrent Neural Networks (RNN)

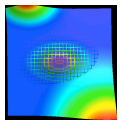




Feed forward representation of RNN

This unfolded representation is already “deepish” ☺️ but with the same weights at each layer (time)

(figures by Geoffrey Hinton)



Recurrent Neural Networks (RNN)

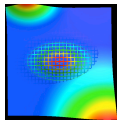
A particular example:

$$\mathbf{h}_t = f_h(\mathbf{x}_t, \mathbf{h}_{t-1}) = \phi_h\left(\mathbf{W}^\top \mathbf{h}_{t-1} + \mathbf{U}^\top \mathbf{x}_t\right)$$
$$\mathbf{y}_t = f_o(\mathbf{h}_t, \mathbf{x}_t) = \phi_o\left(\mathbf{V}^\top \mathbf{h}_t\right),$$

where W,U and V are the weight matrices and the Φ functions are some bounded non-linear functions, such as the sigmoid.

The parameters of this conventional RNN can be estimated by SGD over the cost function with back propagation through time [Rumelhart et al. 1986]. The trick is to unfold the network and after back propagation we average the weights through time to have identical functions (as we assumed initially).

The question remains, how to “deepen” RNN?



Deep Recurrent Neural Networks

Stacked RNN (sRNN [Schmidhuber 1992, El Hihi and Bengio 1996]):

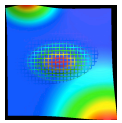
- stacking multiple recurrent hidden layers on top of each other
- modeling multiple time scales in the input sequence

[Pascanu et al. 2014]: three type of expansions:

- deep Input-to-Hidden function
  (temporal neighbours in NLP [Mikolov et al. 2013])

- deep Hidden-to-Hidden function (DT-RNN)
  with shortcuts to preserve the responsiveness of RNN

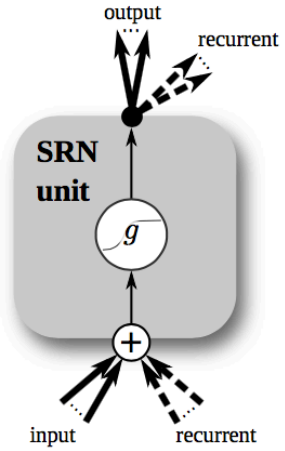
- deep Hidden-to-Output function (DO-RNN)

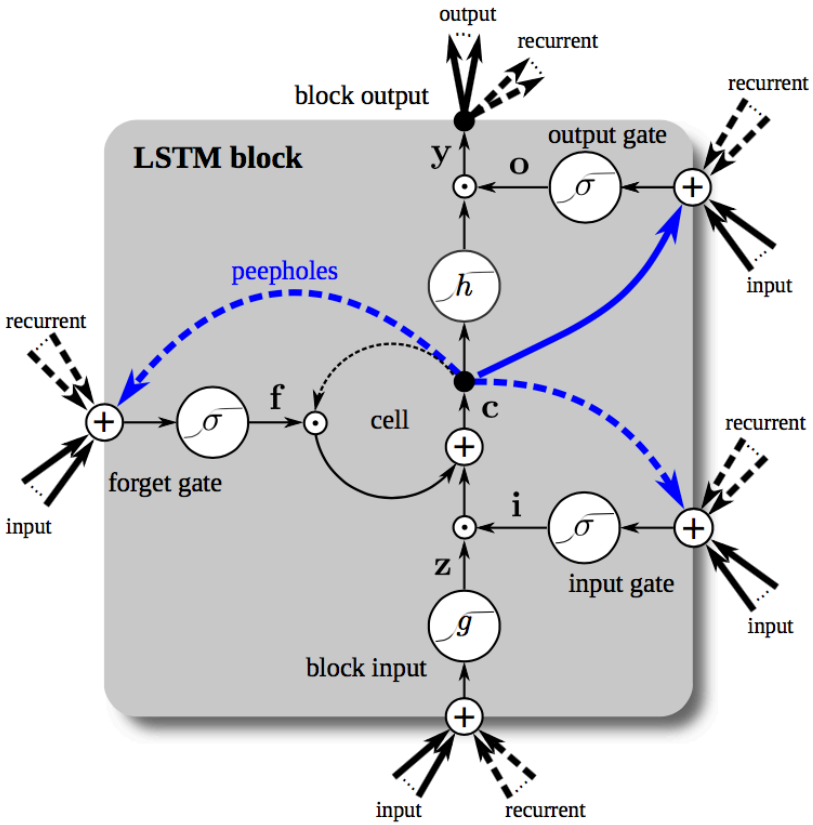


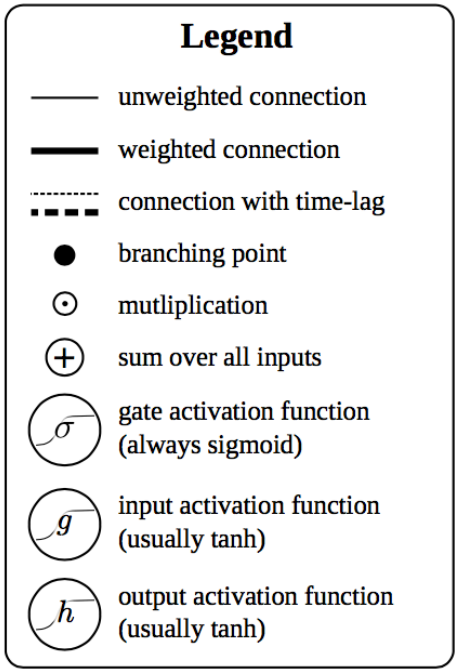
Long-short term memory in general (Greff et al., 2015)

[Hochreiter & Schmidhuber 1997] actually 2005
[Hochreiter,1991]: vanishing gradients prevents RNN to utilize long sequences

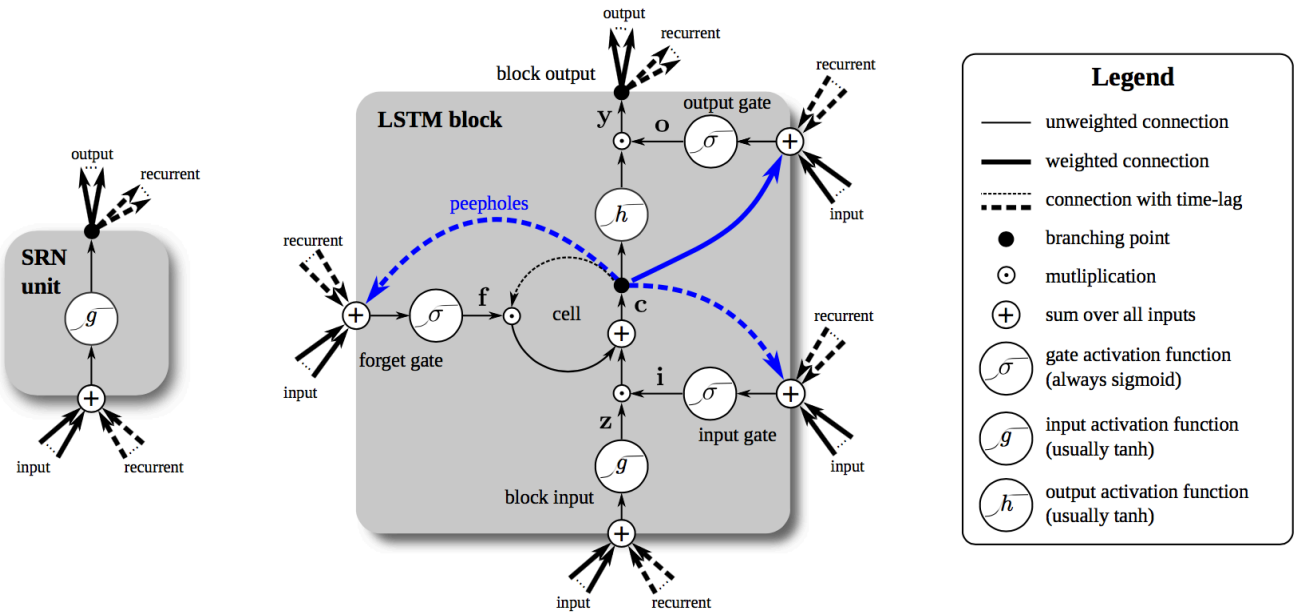
Idea: memorization inside the cell







Long-short term memory in general (Greff et al., 2015)



$$\mathbf{z}^t = g(\mathbf{W}_z\mathbf{x}^t + \mathbf{R}_z\mathbf{y}^{t-1} + \mathbf{b}_z) \qquad \textit{block input}$$

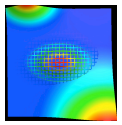
$$\mathbf{i}^t = \sigma(\mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i) \qquad \textit{input gate}$$

$$\mathbf{f}^t = \sigma(\mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f) \qquad \textit{forget gate}$$

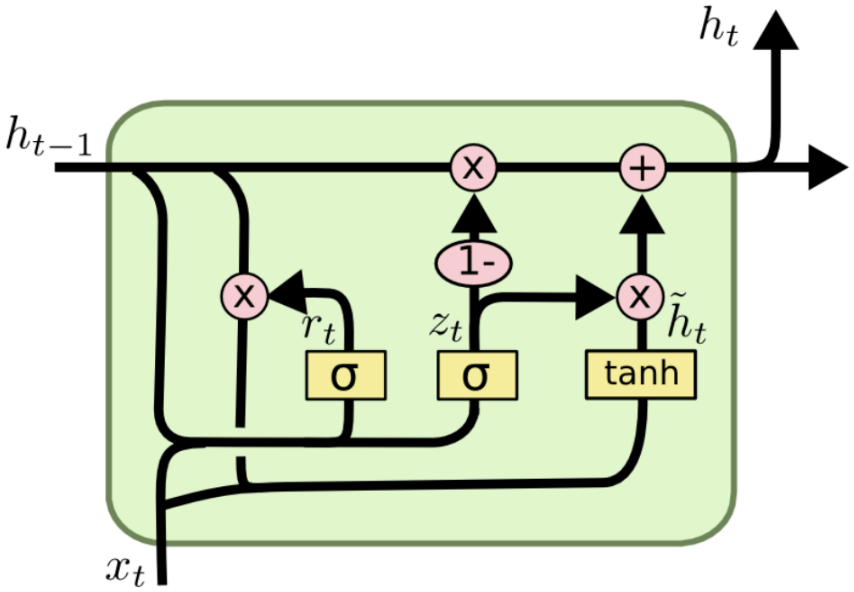
$$\mathbf{c}^t = \mathbf{i}^t \odot \mathbf{z}^t + \mathbf{f}^t \odot \mathbf{c}^{t-1} \qquad \textit{cell state}$$

$$\mathbf{o}^t = \sigma(\mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o) \qquad \textit{output gate}$$

$$\mathbf{y}^t = \mathbf{o}^t \odot h(\mathbf{c}^t) \qquad \textit{block output}$$



Gate recurrent unit or GRU (Cho et al., 2014)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

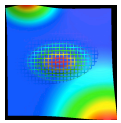
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

No forget gate!

Fig: Christopher Olah



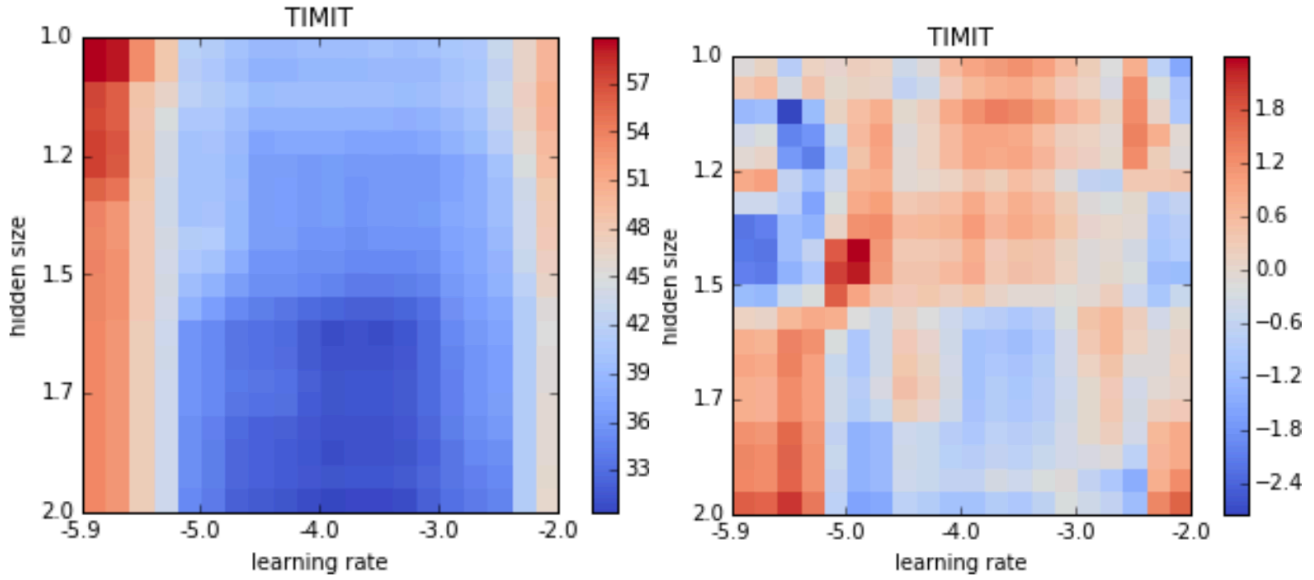
Notes on LSTM

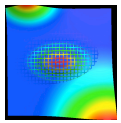
Hyperparameters:

- Random search… [Anderson, 1953, Solis & Wets, 1981]

- The size of the hidden layer is independent of the learning rate [Greff et al., 2017]:

  They can be determined independently. First, determine the learning rate over a small network, then the number of hidden units

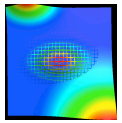




Notes on LSTM

Performance of various versions:

- They are actually very similar [Greff et al., 2017, Chung et al., 2014]
  - GRU is similar in performance but simpler than regular LSTM
  - Forget gate was introduced in 2000 [Gers et al., 2000]
  - Recurrent connections between all gates -> overfit
- Bidirectional LSTMs are better
- Full gradient was introduced only in 2005
- Forget gate and output activation are crucial
- For text, image caption we need attention, what could it be?
- VC dimension of RNN: similar to feed-forward with parameters multiplied typically by the maximal length of sequences [Koiran & Sontag, 1998]



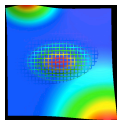
3. Generative models

- The nodes are either input or hidden (no output!)
- Connections between the nodes: not necessary directed edges
- Presumption: finite set of nodes -> finite set of layers (are there any layers?)
- There are some directed/undirected cycles -> not a directed acyclic graph ☹
- Posteriors are similar to FF, but no restrictions (full graph? ☺)

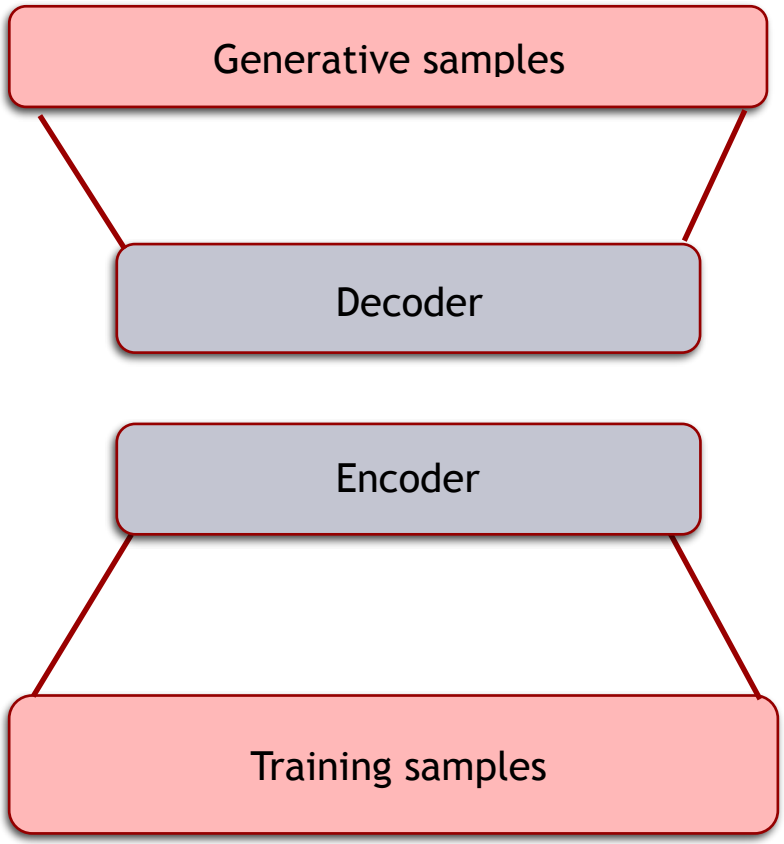
Important models:

- Boltzmann Machine [Hinton et al., 1983]
- Restricted Boltzmann Machine, Harmonium [Smolensky et al., 1986]
- Deep Belief Networks [Hinton et al., 2006]
- Variational Autoencoders [Dayan et al., 1995, Kingma et al., 2013]
- Generative Adversarial Networks [Goodfellow et al., 2014]

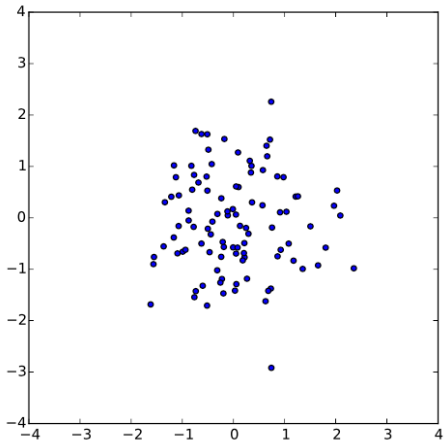
VCdim for regression: fat-shattering, upper bounded by a bit larger networks
VCdim [Alon et al., 1997, Anthony & Bartlett, 1999]

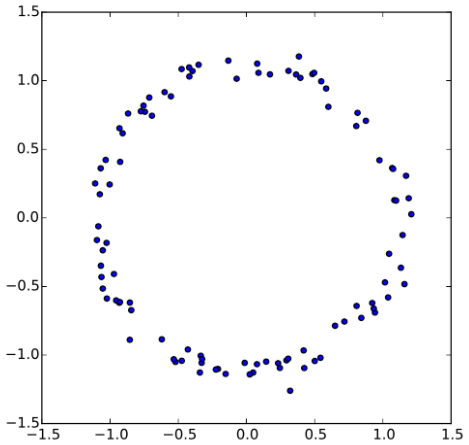


Variational Autoencoders [Kingma et al., 2013]



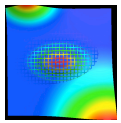
Theorem: any distribution in finite R^d can be formulized via d normal distributions.
E.g. Left is a normal distribution in 2D, right? (fig. Doersch)





Our goal is to generate samples based on an arbitrary distribution (e.g. images).

What is the p(x)? Parametric and …



Variational Autoencoders

Latent models: hidden random variables -> in our case hidden normal distributions. How many?

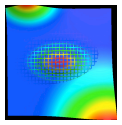
$$P(X) = \int P(X|z;\theta)P(z)dz$$

Our goal is to maximize the log-likelihood over the training samples.

Our assumption is:

$$P(X|z;\theta) = \mathcal{N}(X|f(z;\theta), \sigma^2 * I)$$

This is a very well known distribution ☺️!



Variational Autoencoders

Isotropic normal distributions! Gaussian Mixtures.

What are they representing?

E. g. MNIST:

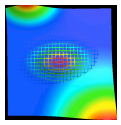
1. Digit (GMM!)
2. Angle
3. Thickness
4. Continuity

Etc.

Should we hard-wire them?

All right, but still the cardinality of z is ….

We like CDF ☺️



Variational Autoencoders

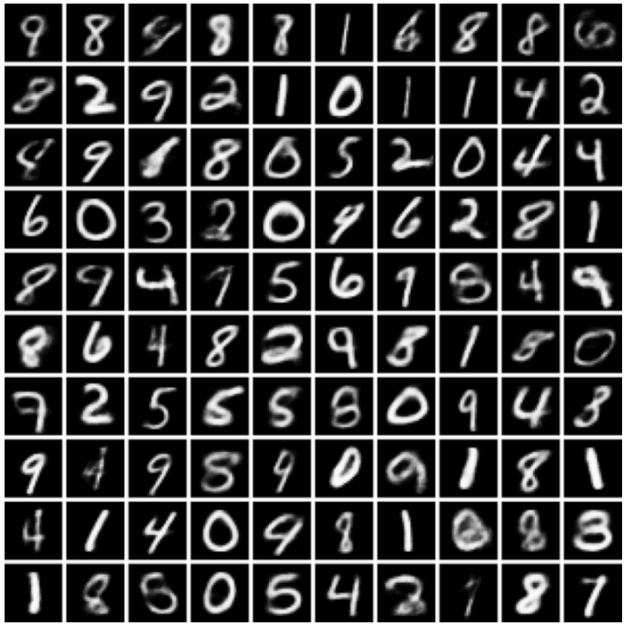
How to sample z? If we know p(z), it could be that p(x|z) is small…

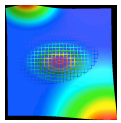
Idea: generate a new distribution Q(z) ~ p(z|x) where p(x|z) is non zero (compatible z-s to generate x). In other words, (RBM! [Hinton, 2002]) KL divergence of p(z|x) and Q(z) should be low:

$$\mathcal{D}\left[Q(z)\|P(z|X)\right] = E_{z\sim Q}\left[\log Q(z) - \log P(z|X)\right]$$

Samples generated by a VAE

Trained on MNIST





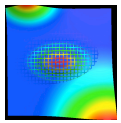
Generative Adversarial Networks

[Goodfellow et al. 2014]:

Discriminative	vs.	Generative networks
Tries to identify false samples generated by the generative model		Tries to generate false samples which tricks the discriminative model
Typically a CNN.		Typically a deconvolution.

Let be G a generative model, D a discriminator ->

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$



Generative Adversarial Networks

Let be G a generative model, D a discriminator

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

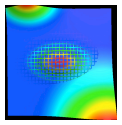
end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

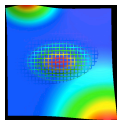
end for

It can be proved that it will converge and the generative distribution will be similar to the data distribution. Recent result: replace KL with Wasserstein divergence -> Wasserstein GAN [Arjovsky et al., 2017]



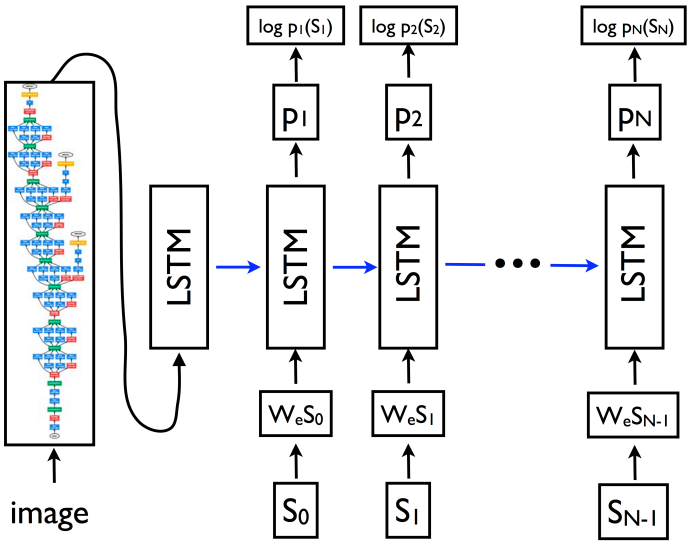
Generative Adversarial Networks

[Radford, Metz and Chintala, 2016]: bedrooms



Example 1: Image caption (Vinyals et al, 2016)

CNN + WE + LSTM: image caption



Better Image Model

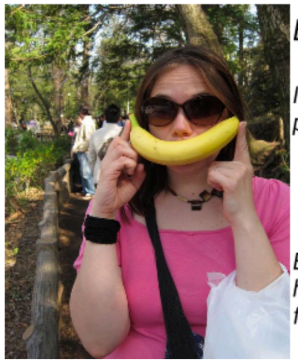
InitialModel: A close up of a plate of food on a table

BestModel: A bunch of bananas and a bottle of wine.

Colors

InitialModel: A train that is sitting on the tracks.

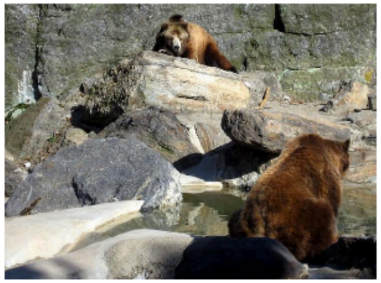
BestModel: A blue and yellow train traveling down train tracks.



Better Image Model

InitialModel: A close up of a person eating a hot dog.

BestModel: A woman holding a banana up to her face.



Counting

InitialModel: A brown bear is swimming in the water.

BestModel: Two brown bears sitting on top of rocks.

Ensemble, BeamSearch and scheduled sampling

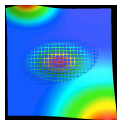
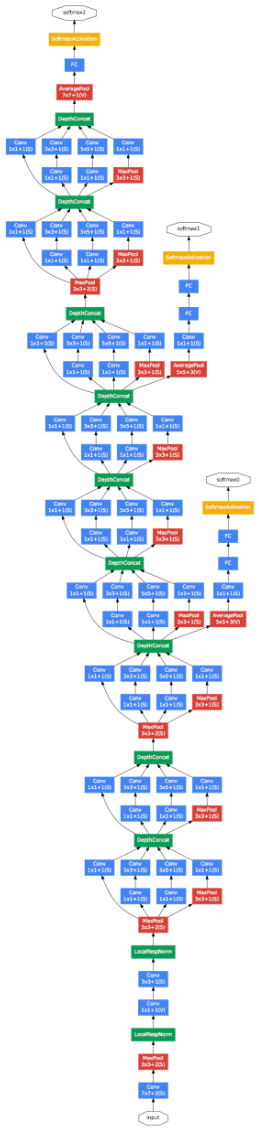
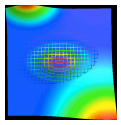


Image model (GoogLeNet + BN)

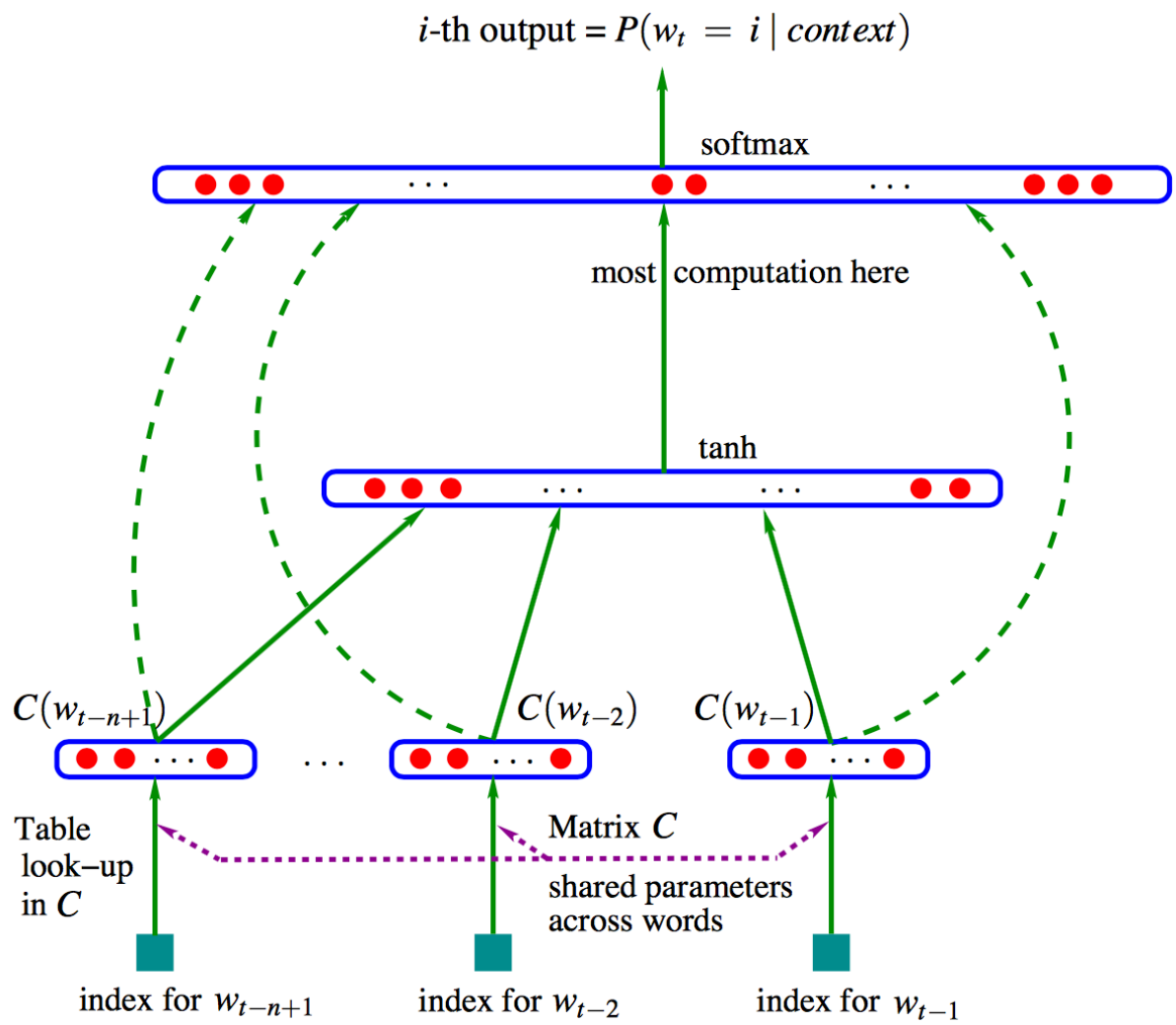
[Szegedy et al., 2014]:
Inception: replace convolutions with smaller but deeper mini networks -> dimension reduction
[Ioffe & Szegedy, 2015]: Batch normalization (we will discuss it)

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								





Word embedding [Y. Bengio et al., 2006]



An actual language model:

Predict terms from the context

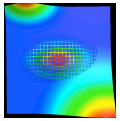
Input representation:
One-hot encoding (dim. is the size of the dictionary)

This is the original model, recent models use smoothed input word representations

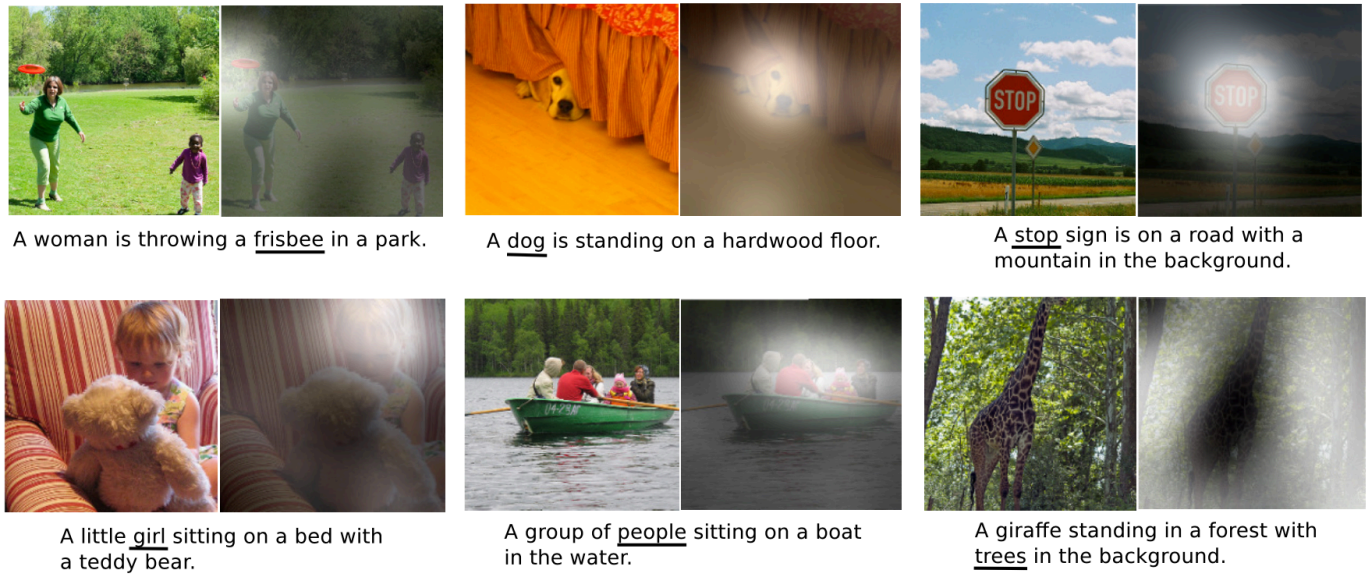
Interesting property [Mikolov et al., 2013]:
King + Woman close to Queen in L2

-> [Rothe,Ebert & Schütze, 2016]:
Orthogonal word embedding, Polarity

In IC: 512 dim emb.



Attention [Xu et al., 2015]



Trick: instead of hard wiring of input selection -> distribution

- Differentiable ☺️
- Distribution: another RNN's softmax output -> we can train it!
- Overfitting…

Soft vs. hard attention:

- Soft: linear combination of location vectors (image parts)
- Hard: one-hot coded

Shortcomings?

Are the vectors additive? (images?)
Connected or non-connected components?
In image caption, if hard coded:
|L| vs. T vs. |W|?
Distribution of importance of locations?
Object vs. concept detection…
(cat, bird vs. daylight, winter etc.)

BUT: do we even need attention? ☺️

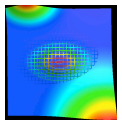
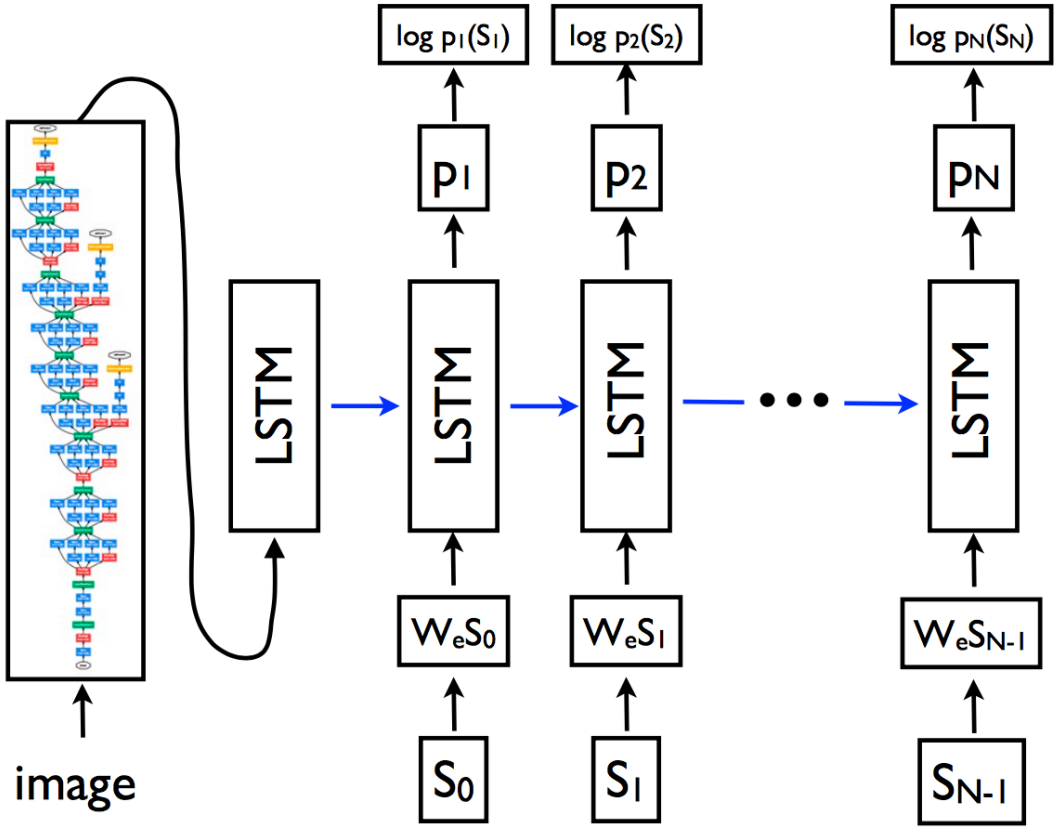


Image caption (Vinyals et al, 2016)

Putting everything together:
CNN (BN Inc.) + WE (d=512) + LSTM (#hidden=512) : image caption



Ensemble, BeamSearch and scheduled sampling
NO Attention! ☺

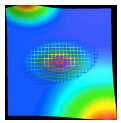
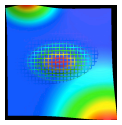


Image caption (Vinyals et al, 2016)

- Pre-trained image model: trained on ImageNet, fine-tuning (tricky) helped a bit
- Word embedding was not pre-trained
- Ensemble:
  - Multiple models with different initialization, learning parameters or even different networks
- BeamSearch:
  - Consider the k best sentences before generating the next word
  - Beam size matters, actually k=3 was the best on the MS COCO challenge
- Optimization: SGD with fixed learning rate and without momentum + Dropout
- Transfer Learning: models trained on different datasets
- Scheduled sampling: curriculum learning strategy, flip a coin to use the predicted or the true previous word

Together 20+% in performance



Example 2: Object detection

Traditional models:

- Haar wavelet [Poggio et al., 1998] and Haar-like features [Viola and Jones, 2001]
  Rigid features + SVM or AdaBoost (and also reduce the number of features)
- Deformable parts model [Felzenswalb et al. 2010]
- 100 Hz: HOG + Boosted Trees etc. [Benenson et al. 2012]

And non-rigid feature extraction (CNN) :

- R-CNN [Girshick et al., 2014]:
  Regions with CNN features
- SPP-net [He et al., 2014]:
  Spatial Pyramid Pooling in DNN
- Fast R-CNN [Girschick et al., 2015]:
  CNN feature maps
- RegionLet [Wang et al., 2015]:
  Integral image over CNN
- Faster R-CNN [Ren et al., 2015]:
  Region Proposal Network

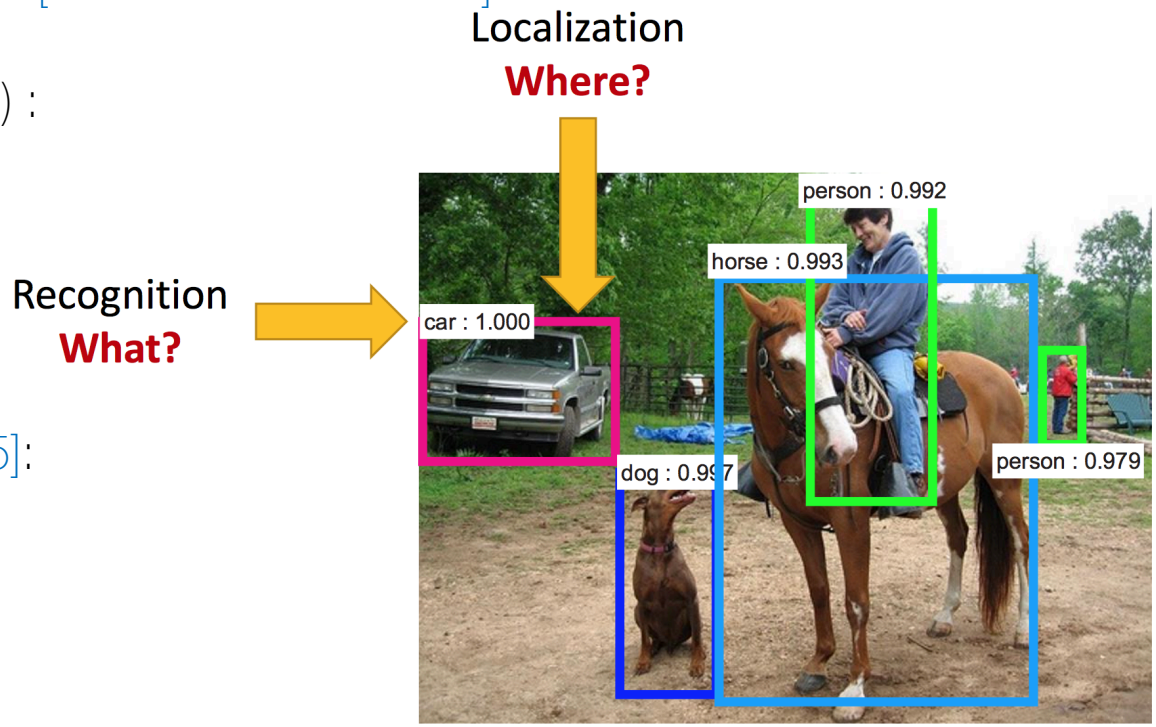
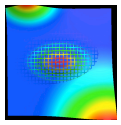
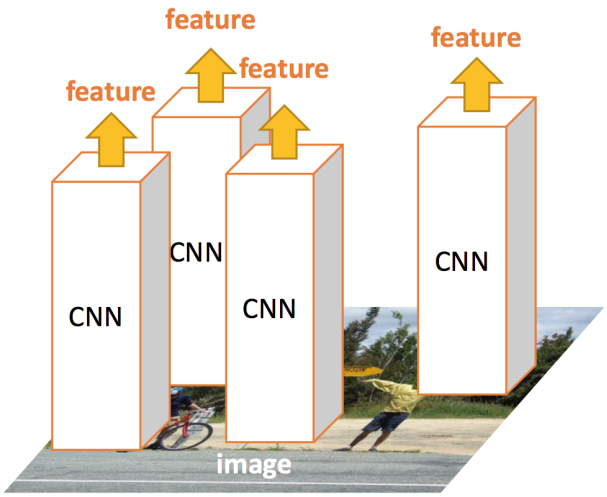
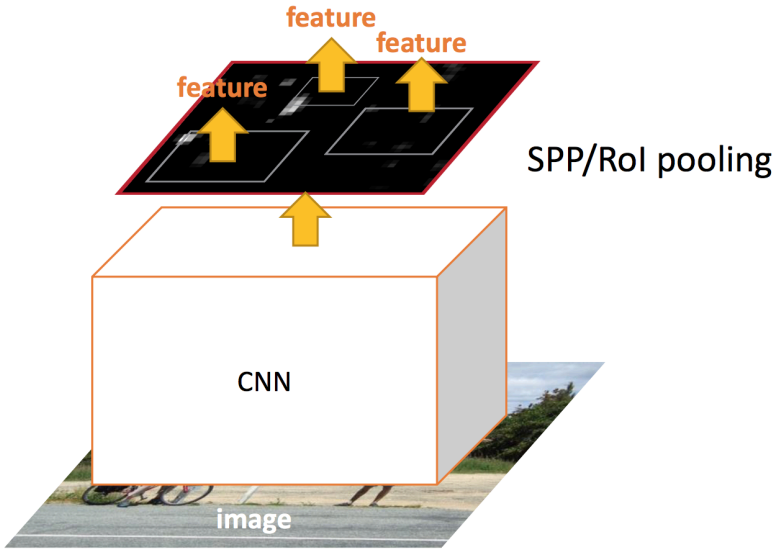


Fig. Kaiming He



R-CNN vs. Fast R-CNN

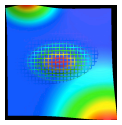




CNN-s over the candidate regions
Rigid region size
High complexity
In practice:
Separate SVM over the feats.
not CDF ☹

One CNN per image
Feature map per pixel (filters/pixel)
Arbitrary sized regions of feature maps
Much faster than R-CNN
SVM over the candidate feature maps
Still not CDF ☹

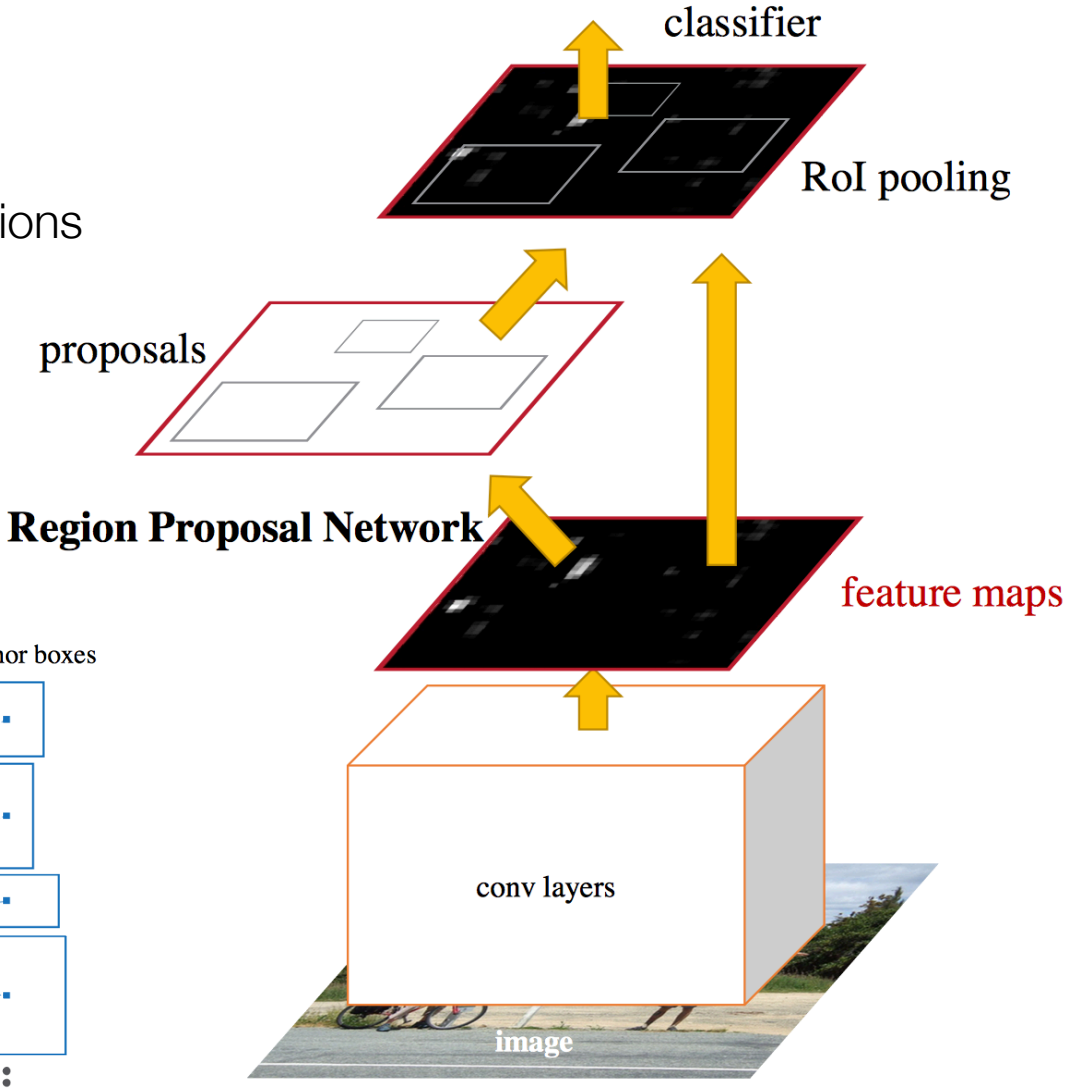
Fig. Kaiming He

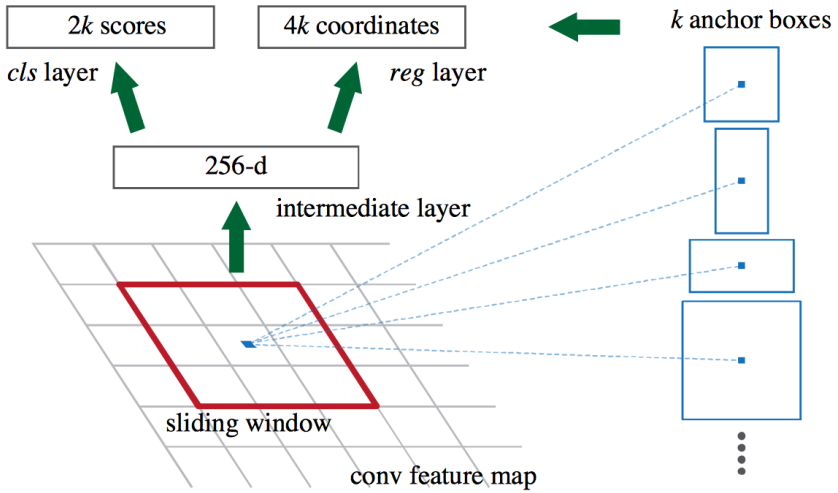


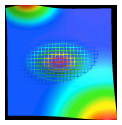
Faster R-CNN [Ren et al., 2015]

Instead of complex search:

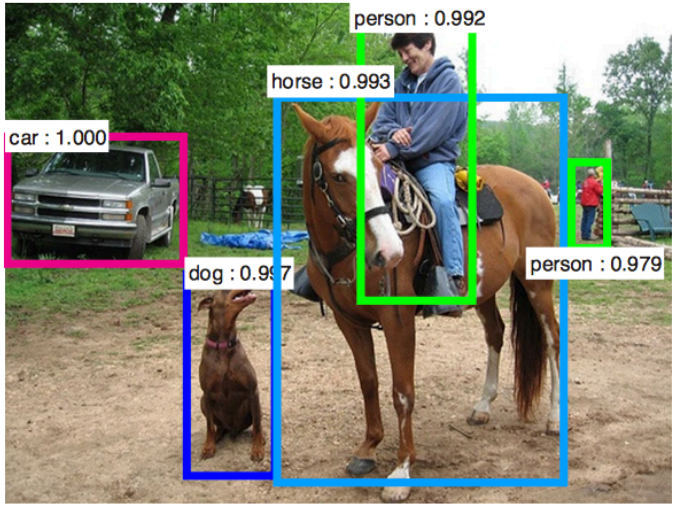
- Region proposals:
  different sized but rectangular regions
  on multiple scales
- Still rigid
- End-to-end optimization: CDF ☺️
- Very fast <200ms (on a Titan...)
- Embedded systems?

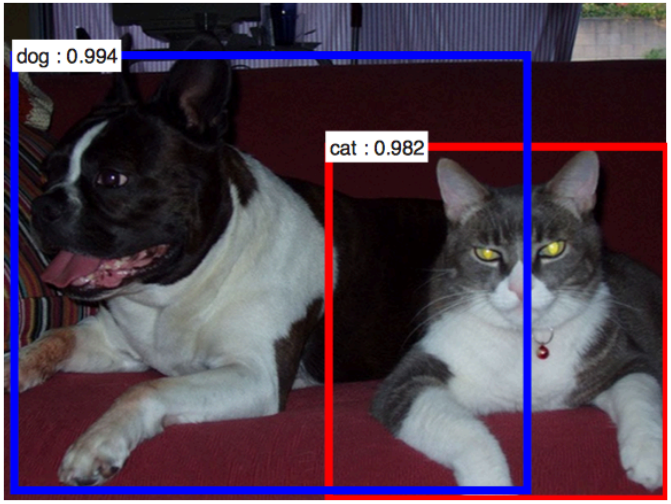


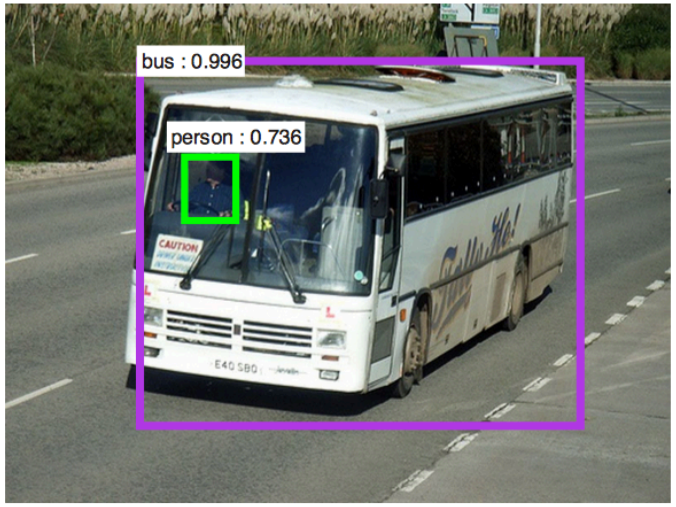


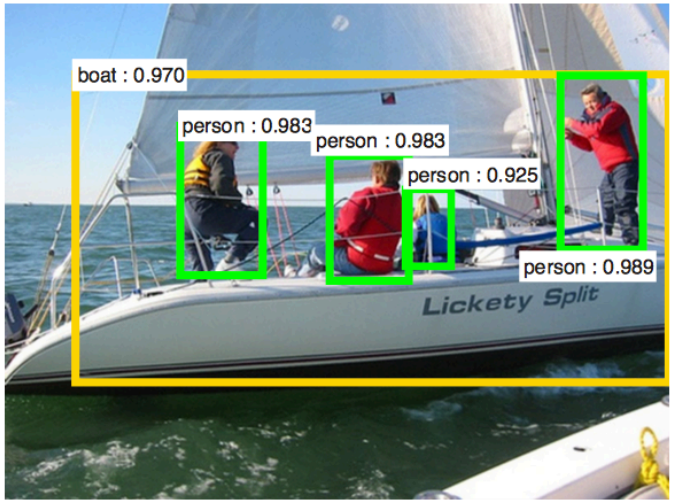


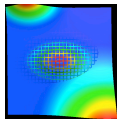
Faster R-CNN [Ren et al., 2015]







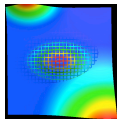




Some connected topics

Missing, but very important topics:

- Optimization: SGD, Newton, ADAM [Kingma & Ba, 2013], RMSProp [Hinton et al., 2014], Nesterov [Nesterov, 1983] etc.
- reinforcement learning [Cassandra, 1998, Sutton and Barto, 1998, Sorokin et al., 2015], AlphaGo, Virtual-Real translations
- self-organizing maps [Ritter et al., 1992, Kohonen, 2013]
- Support Vector Networks (MLP!) [Cortes & Vapnik, 1992] and kernels [Schoelkopf, Herbrich & Smola, 2001]
- Bayesian Networks, when we do not know the structure and Dynamic Bayesian Networks [Pearl, 2011]
- Manifold learning and statistical manifolds [Campbell, 1986]



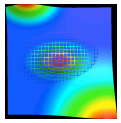
Some unsolved problems

Saturation, vanishing gradients and sparsity empirical risk minimization: optimization, converge speed etc.

- pReLU [Het et al., 2015]
- Maxout [Goodfellow et al., 2013]
- local response/batch normalization [Ioffe et al., 2015, Szegedy et al., 2016]

Generalization gap:

- DropOut [Hinton et al., 2012], DropConnect [Wan et al., 2013]
- Convolution (in comparison to MLP) [Lecun et al.,1998]
- FastFood [Yang et al., 2015]
- Memorization? [Zhang et al., 2017]
- Batch size affects generalization [Keskar et al., 2017]
- practical VCdim? Tree ensembles!



Some unsolved problems

Architecture: network structure

- Lower bounds for deepness to approximate a given function [Rolnick & Tegmark, 2017]
- Network-in-Network [Lin et al., 2014], BN Maxout NiN [Chang et al., 2015]
- spectral representation (pooling) [Rippel & Snoek, 2015]
- Identity map and residual block [He et al., 2015], highway networks [Srivastava et al., 2015]
- Manifold tangent classifier, high-order contractive auto-encoder [Rifai et al., 2011]
- Compression? [He et al., 2015, Ullrich et al., 2017]
- Do we have to learn the parameters? Fisher Information [Cencov, 1982] & Johnson-Lindenstrauss theorem [J&L,1984]: reduce VC dimension from O(wlogw) to O(w) in special cases?
- Embedded systems: We need robust and reliable models (e.g. autonomous drones and vehicles: Nvidia Tegra vs. Titan)

Thank you!