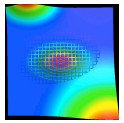
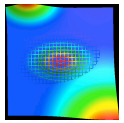


Nagy adathalmazok

2018-2019 őszi félév

2018.11.07

Neurális hálózatok



“shallow” vs. “deep”

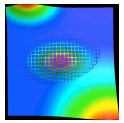
Hipotézis: deep, hierarchikus modellek exponenciálisan hatékonyabbak mint a shallow modellek (Bengio et al. 2009, Le Roux and Bengio, 2010, Delalleau and Bengio, 2011 etc.)

Delalleau and Bengio, 2011: deep sum-product network exponenciálisan kisebb hálózati elemet igényel

Mit nevezünk egy neurális háló mélységének?

Feed forward hálózatok esetében a bemeneti és a kimeneti réteg közti nem lineáris rétegek száma

Nem minde típusú hálózatra igaz ez a megállapítás!

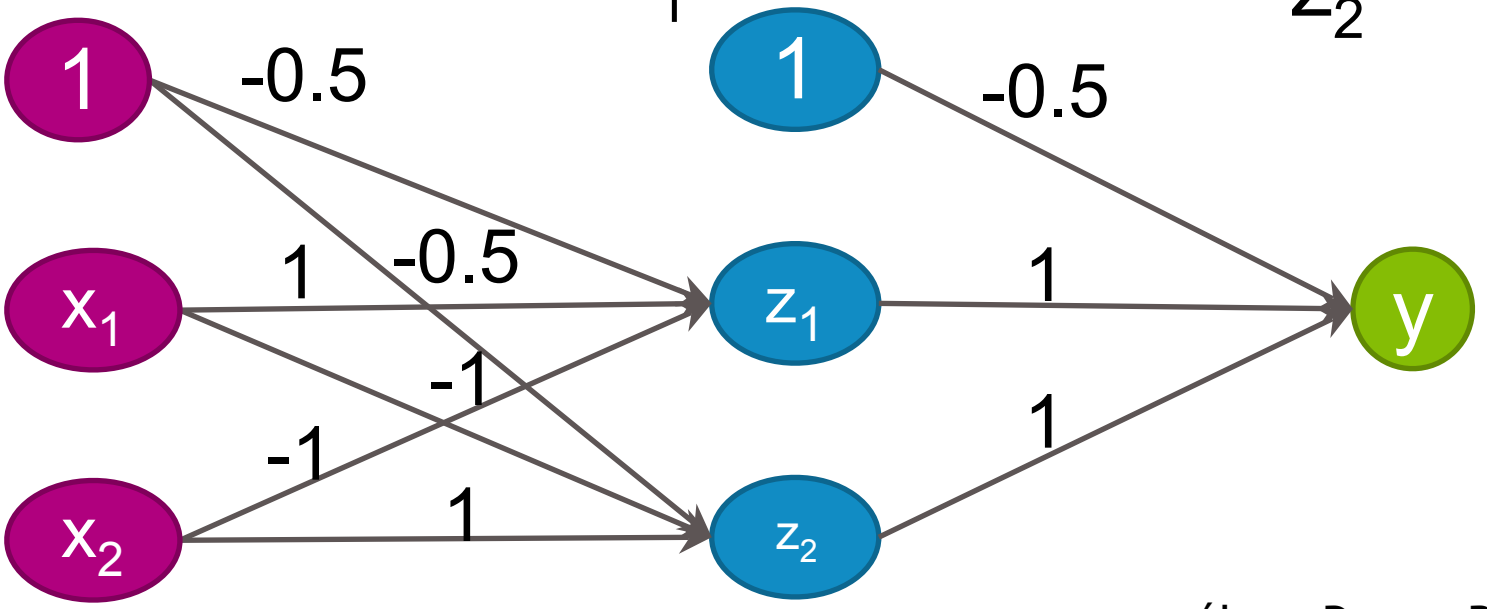


Neurális hálózatok

Két fontos jellemző

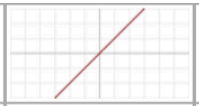
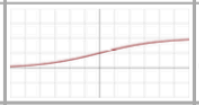
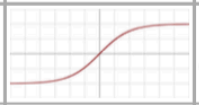
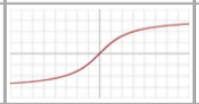
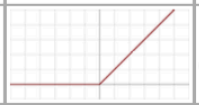
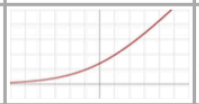
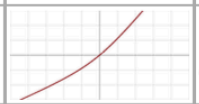
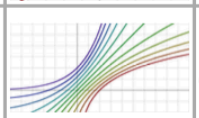
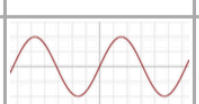
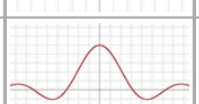
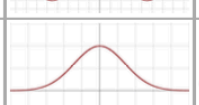
- Huzalozás: maga a gráf (irányított)

$$\text{XOR} = \underbrace{x_1 \text{ AND NOT } x_2}_{z_1} \text{ OR } \underbrace{\text{NOT } x_1 \text{ AND } x_2}_{z_2}$$



ábra: Danny Bickson

Aktiváló függvények

Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
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		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
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		$f(\alpha, x) = \begin{cases} -\frac{\log_e(1-\alpha(x+\alpha))}{\alpha} & \text{for} \quad \alpha < 0 \\ x & \text{for} \quad \alpha = 0 \\ \frac{e^{\alpha x}-1}{\alpha} + \alpha & \text{for} \quad \alpha > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \frac{1}{1-\alpha(\alpha+x)} & \text{for} \quad \alpha < 0 \\ e^{\alpha x} & \text{for} \quad \alpha \geq 0 \end{cases}$
Sinusoid		$f(x) = \sin(x)$	$f'(x) = \cos(x)$
Sinc		$f(x) = \begin{cases} 1 & \text{for} \quad x = 0 \\ \frac{\sin(x)}{x} & \text{for} \quad x \neq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x = 0 \\ \frac{\cos(x)}{x} - \frac{\sin(x)}{x^2} & \text{for} \quad x \neq 0 \end{cases}$
Gaussian		$f(x) = e^{-x^2}$	$f'(x) = -2xe^{-x^2}$

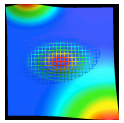
Ez határozza meg a neuron kimenetét
Lehet:

- lineáris/
  nem lineáris
- Korlátos/
  nem korlátos

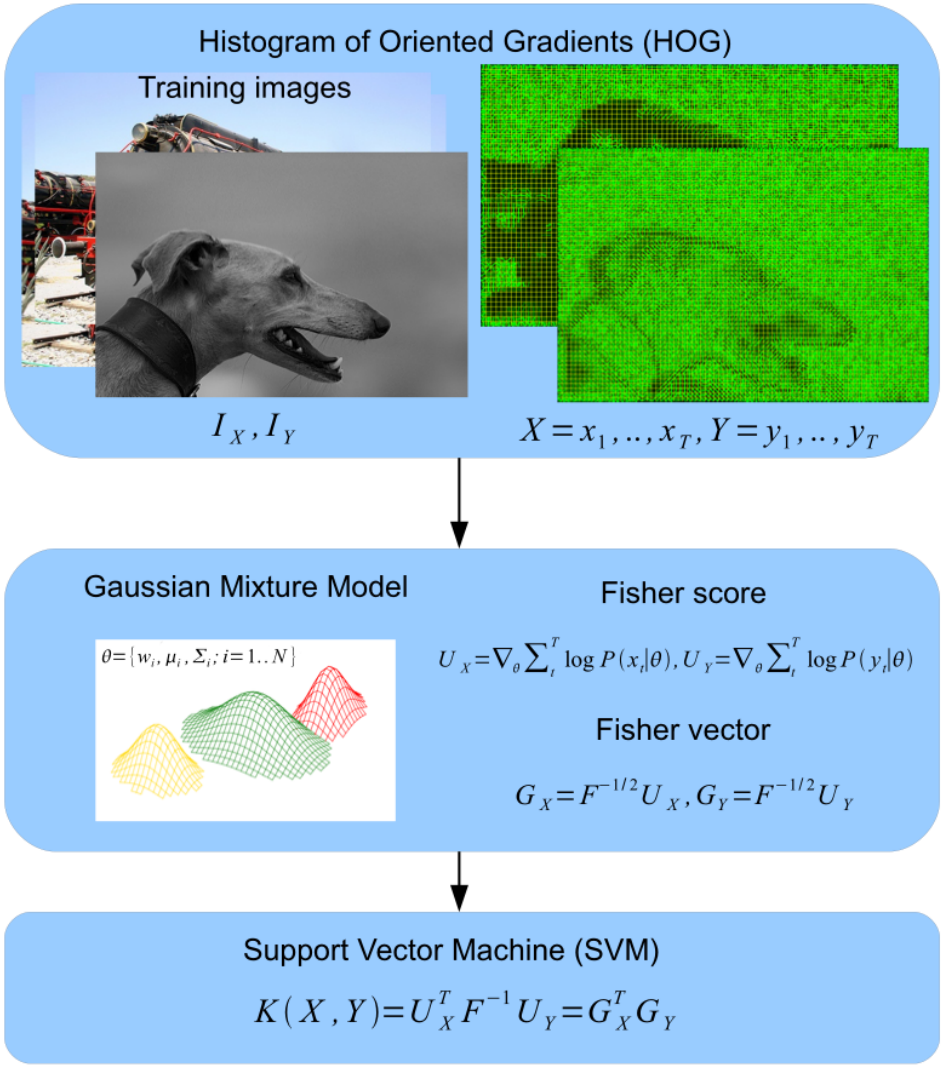
értékkészletű

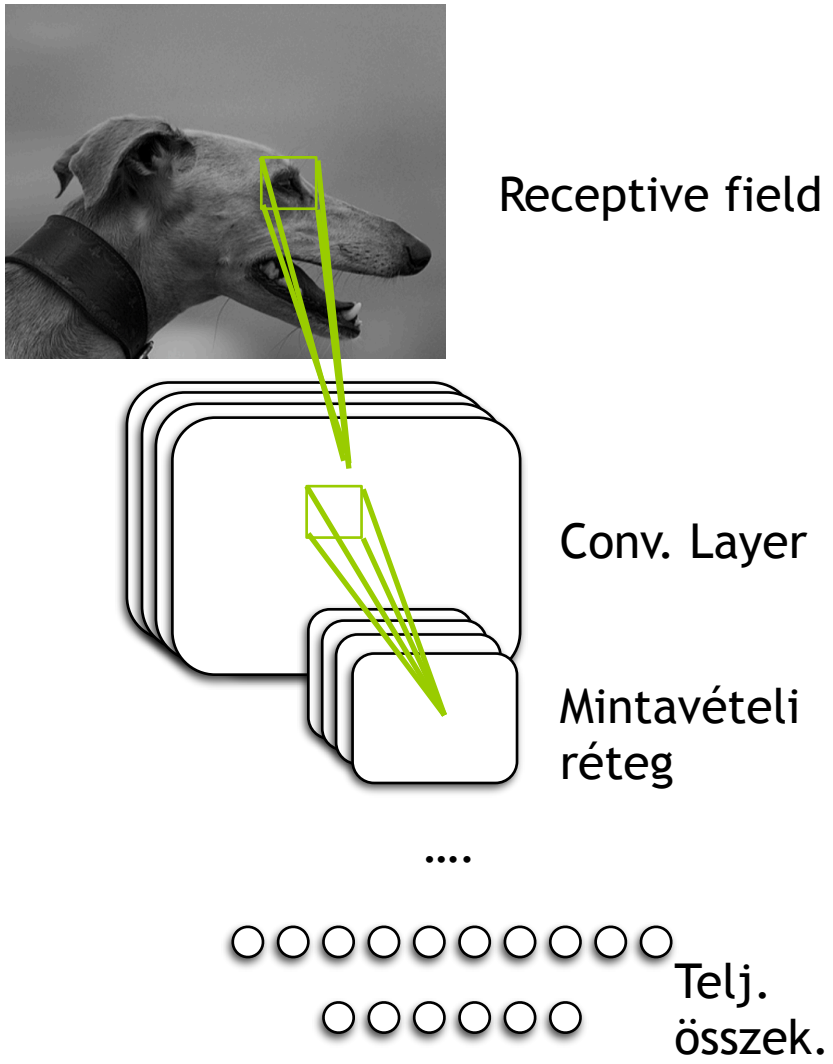
- Általában monoton, de nem szükségszerű

forrás: wiki

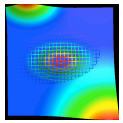


Convolutional Neural Network (CNN)



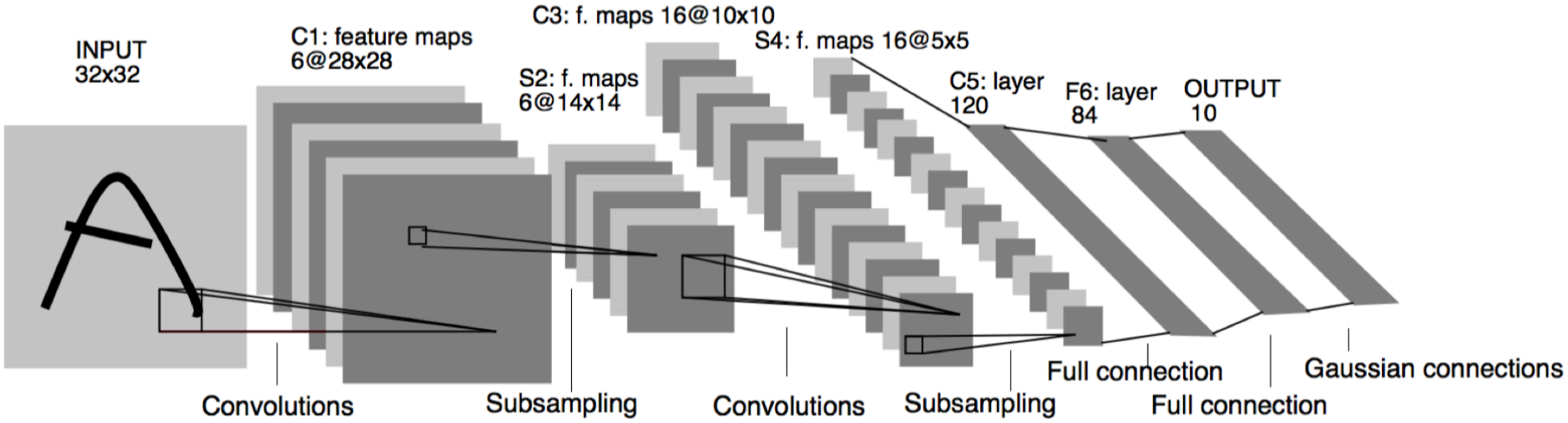


Tradicionális képosztályozás vs. CNN



LeNet-5

Kézírásfelismerés (LeCun et al. 1998)



- Tanult alacsony szintű feature kinyerés
- Nem csak tanult de sokféle is (LeNet 5: 6 +16+120)
- A mintavételi réteg egyfajta térbeli modellt eredményez szemben a BOW-
- Miután nem teljesen összekapcsolt, kevesebb paraméter

Tanítás: back propagation mini-batch alapon gradiens-el, inicializáció normál eloszlásból
A definició szerint a CNN mély struktúra

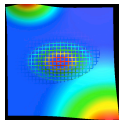
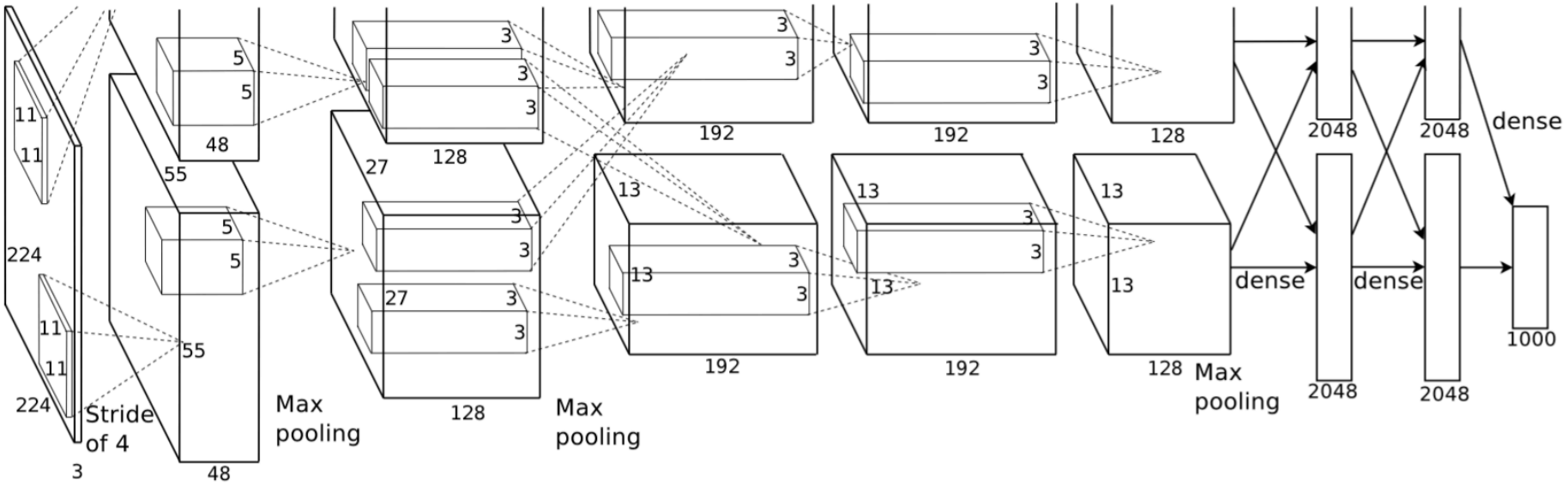


Image classification CNN-el

Krizhevsky et al. 2012 , ők nyerték az ImageNet-et 2012-ben



Fejlődés a LeNet-hez képest:

- ReLU és nem sigmoid és Local response normalization (1.2%..1.4% javulás)
- A mintavételi réteg átfedő (0.3..-0.4% javulás)
- Tradicionális képi trükkök: horizontális tükrözés, véletlen mintavétel, PCA az RGB csatornák felett és zajokhozzáadása (1% javulás)
- Dropout

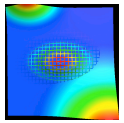
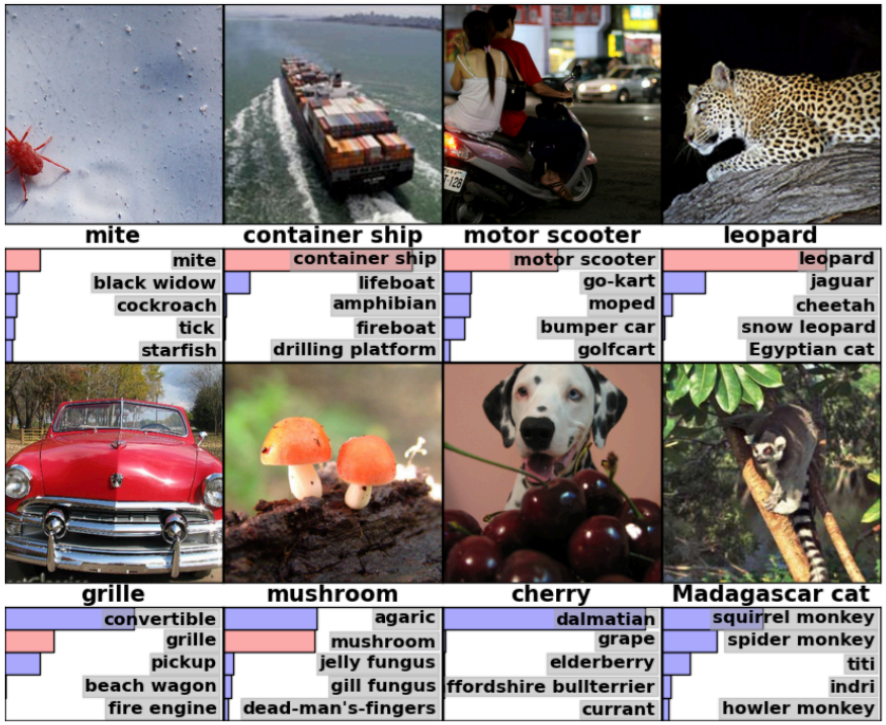


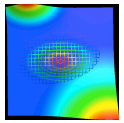
Image classification CNN-el

Krizhevsky et al. 2012
ImageNet: 150k teszt kép és 1.2 millió tanuló kép, 1000 cimke.
top-1 és top-5 hiba arány

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%**

* - Külső adat
4096 dimenziós repr.
Tanulás: 5-6 nap két Nvidia GTX 580 3(

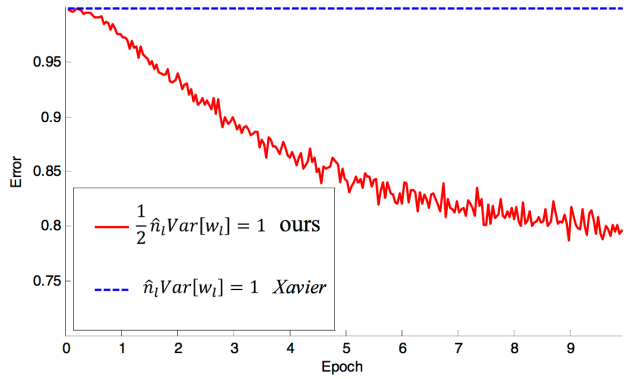




Jelenleg ismert (egyik)legjobb eredmény

He et al. 2015 (MS research): Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

Parametric ReLU + nem Gaussian inicializálás + extrém méret: A:19 réteg, B/C: 22 ré



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**

Tanulás : 8xK40 Nvidia GPU 3..4 hét (!)

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

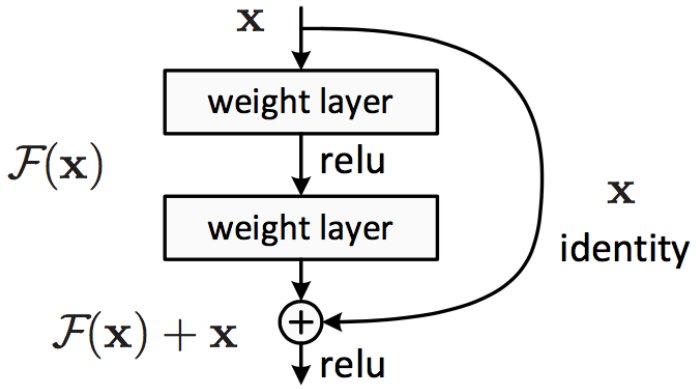
GT: torch
1: stage
2: spotlight
3: torch
4: microphone
5: feather boa

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

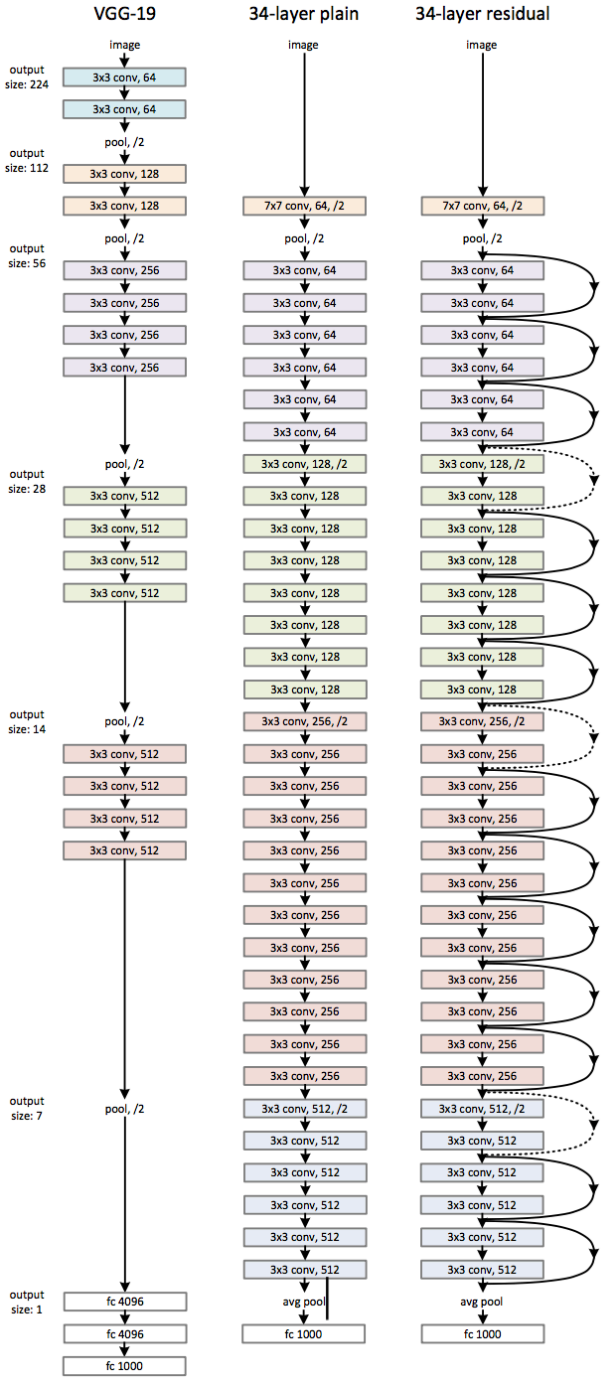
GT: go-kart
1: go-kart
2: crash helmet
3: racer
4: sports car
5: motor scooter

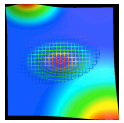
Talán legjobb

[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**





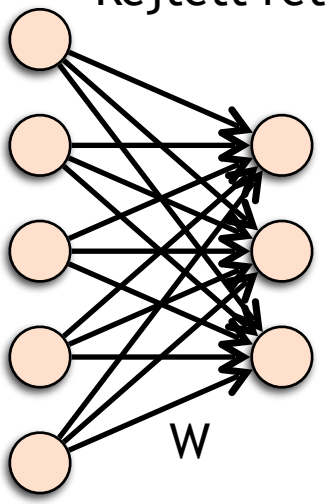
Restricted Boltzmann Machines (RBM)

1983: Boltzmann machine (Hinton and Sejnowski) -> nagyon általános modell (bármely két neuron között lehet él, csak önmagukkal nem)

1986: Harmonium (Smolensky) : gyak. RBM
2002: RBM név és Contrastive Divergence

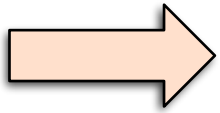
Miért restricted?
Ha Boltzmann/Gibbs a közös eloszlás -> klikkek felett faktorizálódik -> ha nem engedünk meg éleket a rétegek elemei közt a maximális klikkek két eleműek lesznek

Rejtett réteg (H)



Látható réteg, maga az adat (V)

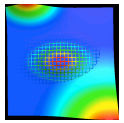
Nem hiába a név



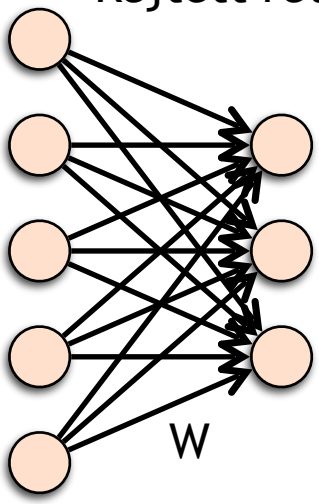
$$p(\mathbf{v},\mathbf{h}) = \frac{1}{Z}\, e^{-E(\mathbf{v},\mathbf{h})}$$

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$$

$$p(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$$



Restricted Boltzmann Machines (RBM)



Az energia függvény (ezt szeretnénk minimalizálni):

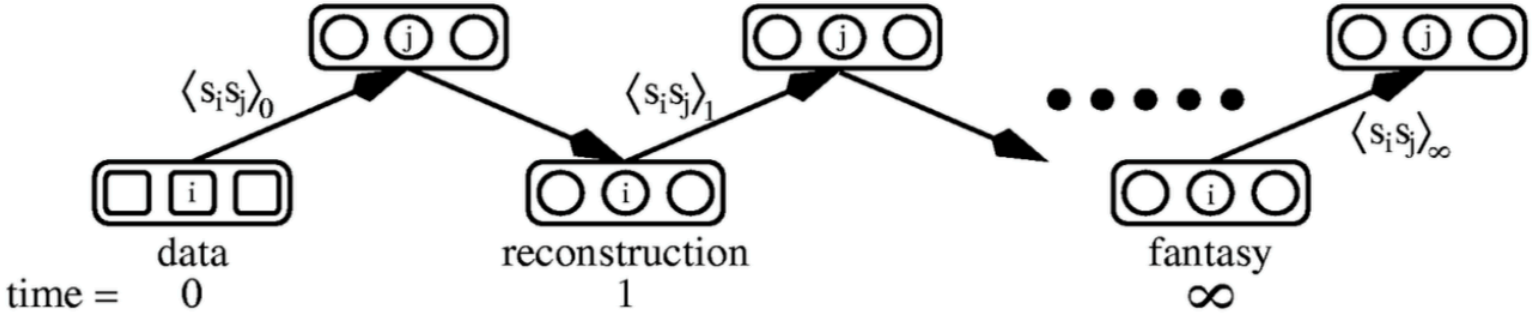
$$E(\mathbf{v},\mathbf{h}) = -\sum_{i\in\text{visible}} a_i v_i - \sum_{j\in\text{hidden}} b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

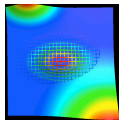
h általában szigmoid.

Gradiens:

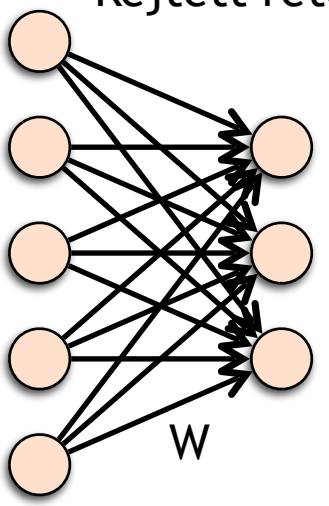
Sajnos nem triviális -> Gibbs mintavétel

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}$$





Restricted Boltzmann Machines (RBM)



$$p(h_j = 1 \mid \mathbf{v}) = \sigma(b_j + \sum_i v_i w_{ij})$$

$$p(v_i = 1 \mid \mathbf{h}) = \sigma(a_i + \sum_j h_j w_{ij})$$

A likelihood maximalizálása helyett a Kullback-Leibler divergenciát minimalizálják (a lánc stacionárius eloszlása és az eredeti adat között)

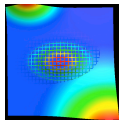
$$P^0 \parallel P_\theta^\infty = \sum_{\mathbf{d}} P^0(\mathbf{d}) \log P^0(\mathbf{d}) - \sum_{\mathbf{d}} P^0(\mathbf{d}) \log P_\theta^\infty(\mathbf{d})$$

$$= -H(P^0) - \langle \log P_\theta^\infty \rangle_{P^0},$$

Constractive divergence (Hinton 2002):
Gyönyörű trükk: Minimalizáljuk a a divergenciák különbségét

$$P^0 \parallel P_\theta^\infty \qquad P_\theta^1 \parallel P_\theta^\infty$$

-> a problémás rész kiesik, marad egy tag, de az elhanyagolható a gyakorlatban.

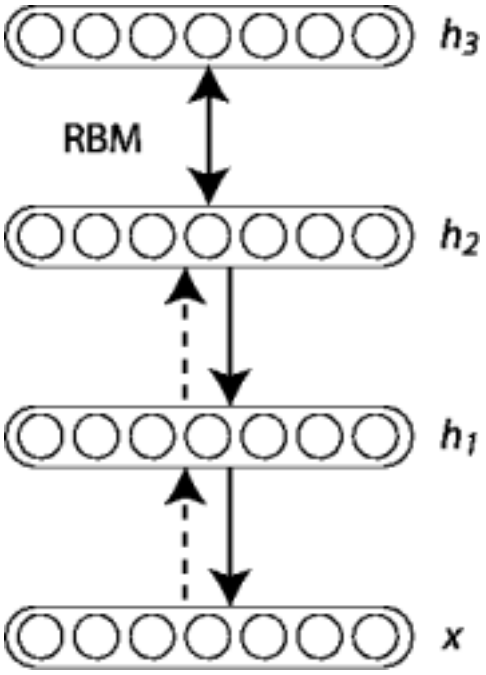


Deep Belief Network (DBN)

Boosting szerű modell

Egymásra pakolt RBM-ek -> sajnos a CD itt nem működne

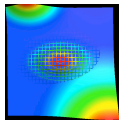
De mi lenne ha minden réteg más és más részét értené meg az problémának.



Legyen l a rejtett rétegek száma:

$$P(x, h^1, \ldots, h^\ell) = \left(\prod_{k=0}^{\ell-2} P(h^k | h^{k+1}) \right) P(h^{\ell-1}, h^\ell)$$

Greedy layer learning (Hinton et al. 2006):
bebizonyították, hogy a rétegeket lehet egymás után tanítani, nem kell back propagáció



Recurrent Neural Networks (RNN)

Dinamikus (diszkrét) rendszerek szimulációja (Rumelhart *et al*. 1986)

Három elem:

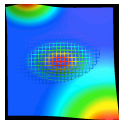
x_t : bemeneti réteg a t-dik időpontban
y_t : kimeneti réteg a t-dik időpontban
h_t : a rejtett réteg a t-dik időpontban

Ahogy sejtjük, elég egyszerű a modell leírása:

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

Memóriaként működik, az előző bemenetek számítanak.

Szavak egy mondatban?



Recurrent Neural Networks (RNN)

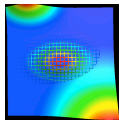
Legyen D egy sorozat:

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

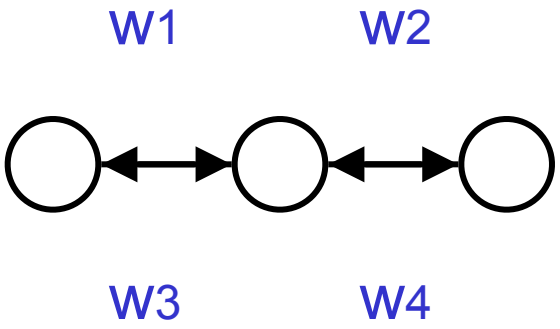
RNN esetében a paraméterek becslése (Θ) a következő költségfüggvény minimalizálási feladata:

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

ahol $\mathbf{h}_t^{(n)} = f_h(\mathbf{x}_t^{(n)}, \mathbf{h}_{t-1}^{(n)})$ és $\mathbf{h}_0^{(n)} = \mathbf{0}$. d(y,f(h)) valamilyen divergencia (pl. négyzetes hiba).



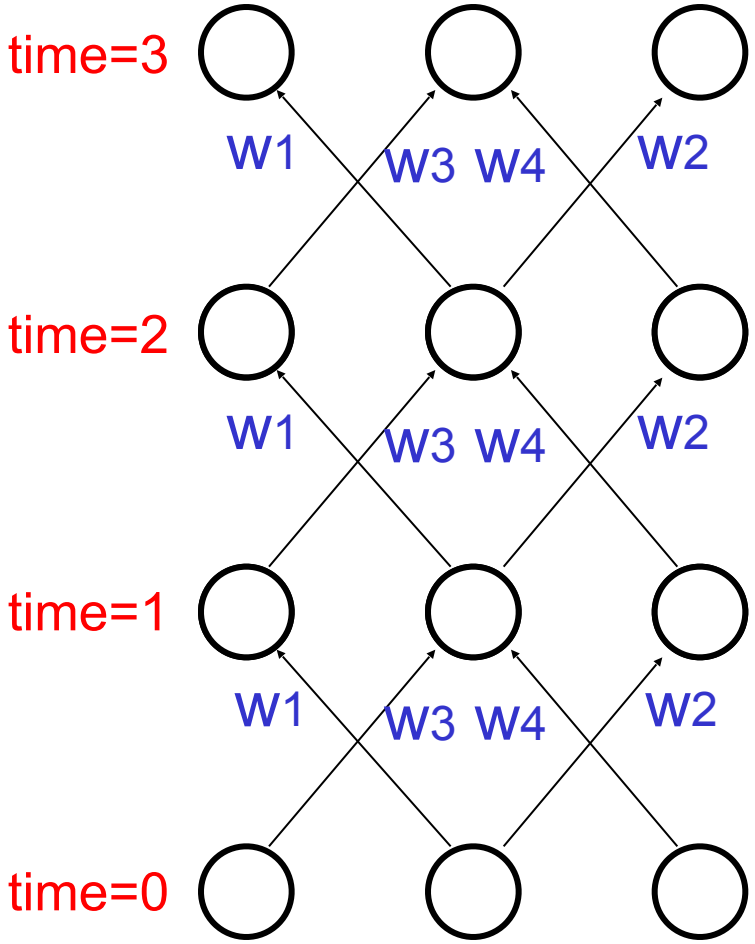
Recurrent Neural Networks (RNN)



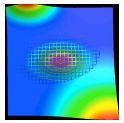
Az RNN kibontása

Ez már önmagában “deep”-szerű ☺

Megkötés, hogy minden szinten a súlyok azonosak (azaz időben nem változnak)



(ábra: Geoffrey Hinton)



Recurrent Neural Networks (RNN)

Egy egyszerűbb példa:

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

ahol W,U és V a paraméter mátrixok és Φ korlátos nem lineáris függvények. pl. szigmoid.

Ebben az esetben a paraméterek becsülhetőek SGD-vel back prop.-al a kibontott hálózaton (Rumelhart *et al*. 1986). A trükk, hogy miután felbontottuk a hálózatot átlagolással azonosan tartjuk a súlyokat.

Jó jó , de hogy lesz ebből “deep” RNN?

Long short term memory (LSTM)

[Hochreiter & Schmidhuber 1997]

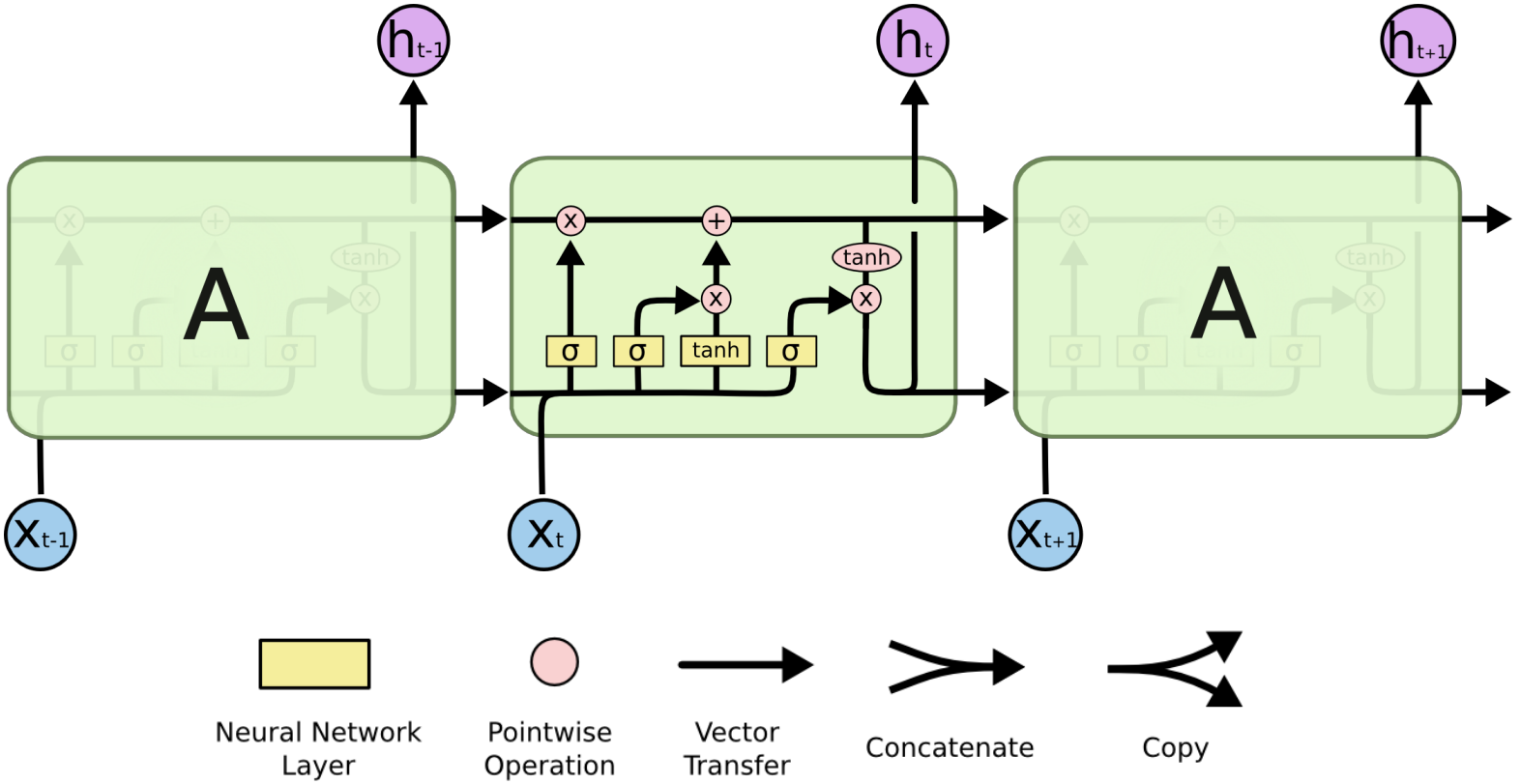
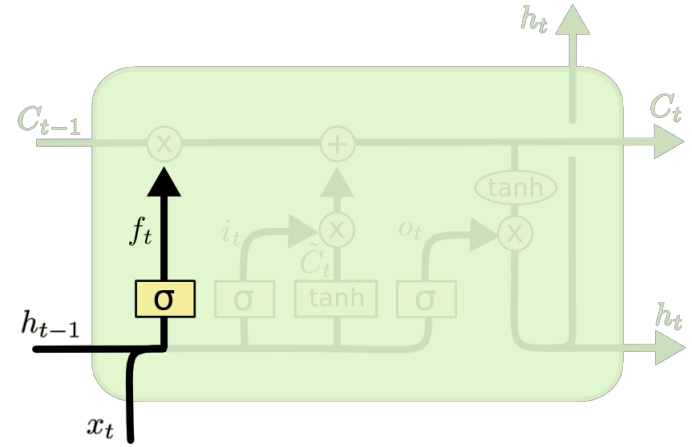


Fig: Christopher Olah

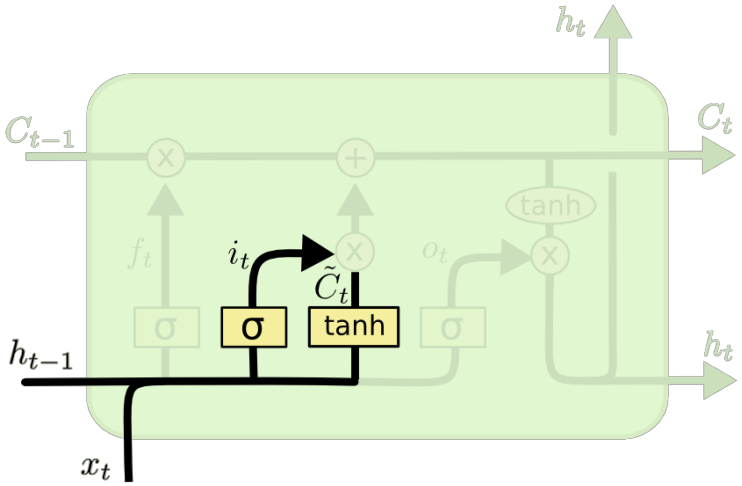
Long short term memory (LSTM)

[Hochreiter & Schmidhuber 1997]



Forget gate: megtartjuk nem tartjuk (meg)

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$



Input gate: szelekció és új állapot

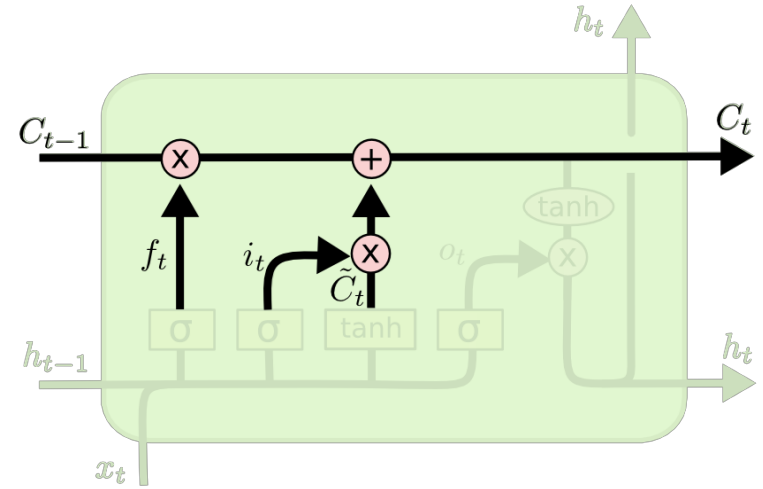
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Fig: Christopher Olah

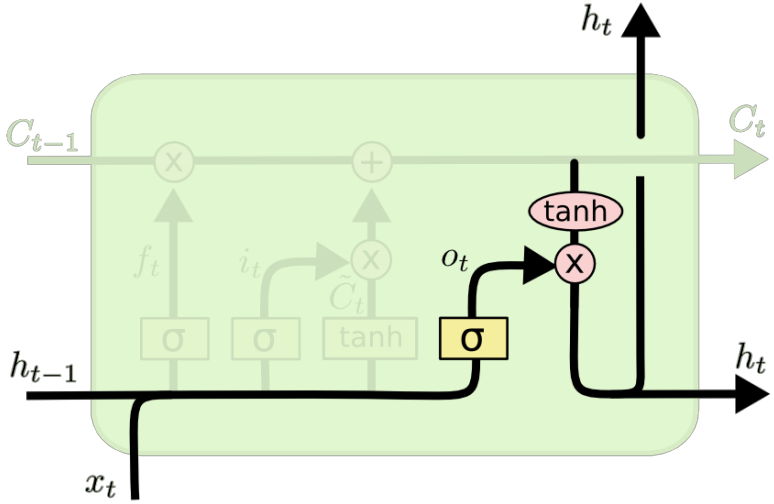
Long short term memory (LSTM)

[Hochreiter & Schmidhuber 1997]



Maga az új állapot

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



Kimenet: új állapot és kimenet

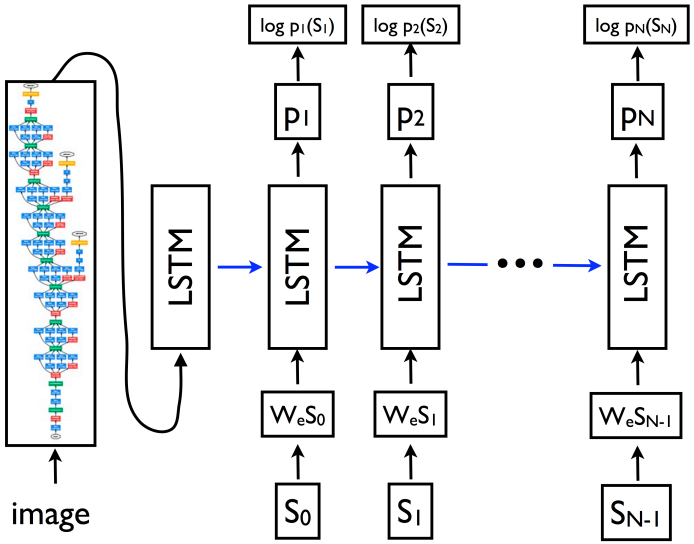
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Fig: Christopher Olah

Kép annotáció (Vinyals et al, 2016 szept.)

CNN + WE + LSTM + attention: image caption



Better Image Model

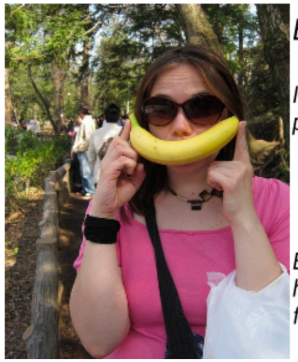
InitialModel: A close up of a plate of food on a table

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

Colors

InitialModel: A train that is sitting on the tracks.

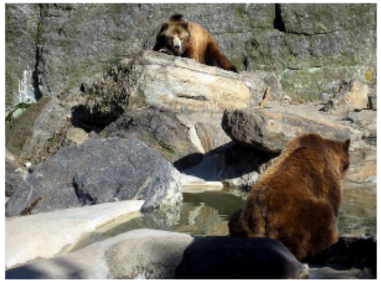
BestModel: A blue and yellow train traveling down train tracks.



Better Image Model

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

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

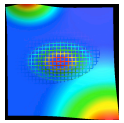


Counting

InitialModel: A brown bear is swimming in the water.

BestModel: Two brown bears sitting on top of rocks.

"Ensemble" és "scheduled sampling"



Recurrent Neural Networks (RNN)

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

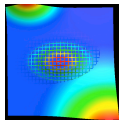
- pakoljunk egymásra konvencionális RNN-t
- ezzel többféle időosztást tudunk kezelni (súlyok!)

Pascanu et al. 2014: három irány:

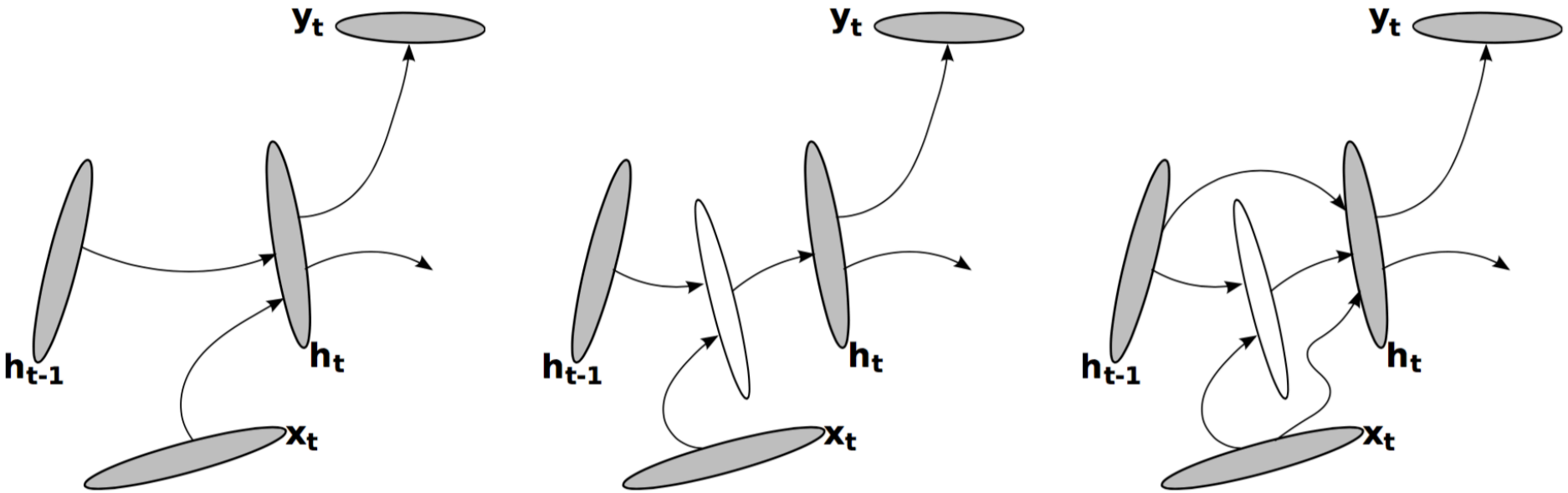
- deep Input-to-Hidden függvény
  (szomszédosság NLP, Mikolov et al. 2013)

- deep Hidden-to-Hidden függvény (DT-RNN)
  sajnos ezzel lassul az alkalmazkodás, épp ezért kerülőutak (shortcut)

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

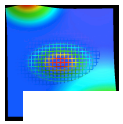


Deep Recurrent Neural Networks (dRNN, Pascanu et al. 2014)

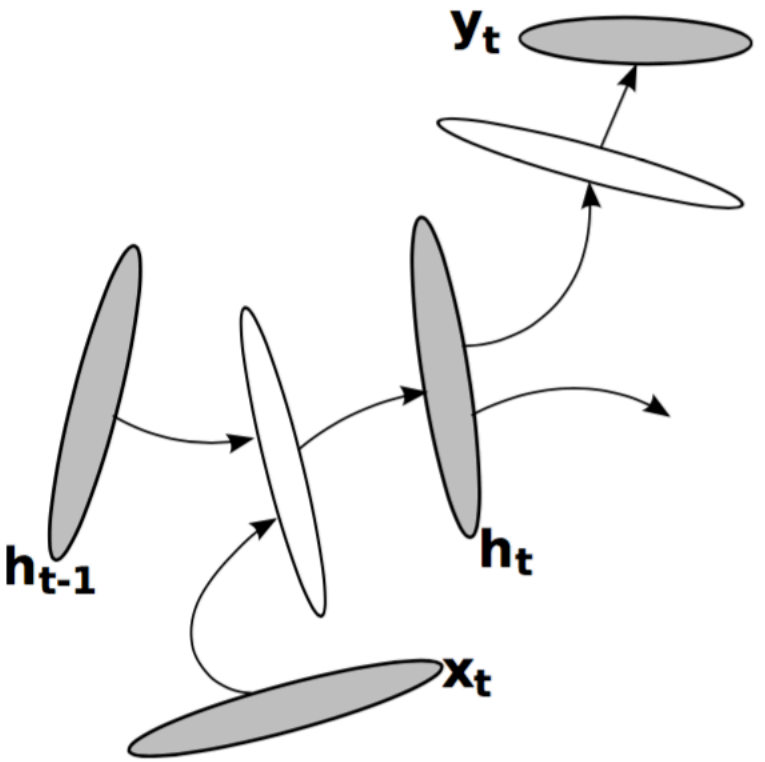


RNN DT-RNN DT-RNN kerülőuttal

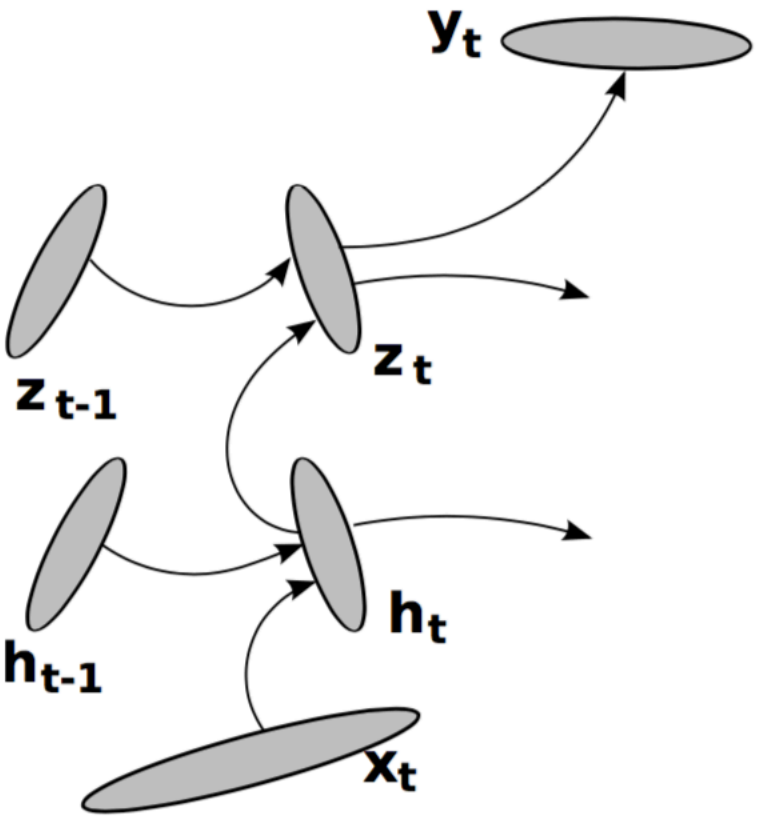
ábra: Pascanu et al. 2014



Deep Recurrent Neural Networks (dRNN, Pascanu et al. 2014)

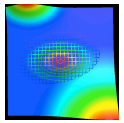


DOT-RNN



sRNN

ábra: Pascanu et al. 2014



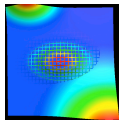
Deep Recurrent Neural Networks (dRNN, Pascanu et al. 2014)

Kiértékelés: log-prob. a következő elemek felett

1. Kísérlet: zenei adatfolyamok, Nottingham, JSB Chorales és MuseData
2. Kísérlet: karakter és szó szintű nyelvi modell, Penn Treebank Corpus

			RNN	DT(S)-RNN	DOT(S)-RNN	sRNN 2 layers
Music	Notthingam	# units	600	400,400	400,400,400	400
		# parameters	465K	585K	745K	550K
	JSB Chorales	# units	200	400,400	400,400,400	400
		# parameters	75K	585K	745K	550K
	MuseData	# units	600	400,400	400,400,400	600
		# parameters	465K	585K	745K	1185K
Language	Char-level	# units	600	400,400	400,400,600	400
		# parameters	420K	540K	790K	520K
	Word-level	# units	200	200,200	200,200,200	400
		# parameters	4.04M	6.12M	6.16M	8.48M

Megj.: SGD-vel tanítva, a tanulási növekmény manuális



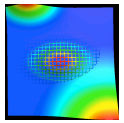
Deep Recurrent Neural Networks (dRNN, Pascanu et al. 2014)

Kiértékelés: negatív log-prob.

1. Kísérlet: zenei adatfolyam, Nottingham, JSB Chorales és MuseData
   - 200 illetve 50 lépésenként propagálás
   - dalok végén újra inicializálás

	RNN	DT(S)-RNN	DOT(S)-RNN	sRNN	DOT(S)-RNN*
Notthingam	3.225	3.206	3.215	3.258	2.95
JSB Chorales	8.338	8.278	8.437	8.367	7.92
MuseData	6.990	6.988	6.973	6.954	6.59

(*) maxout kimeneti réteg szigmoid helyett, illetve dropout (mindjárt jön)



Deep Recurrent Neural Networks (dRNN, Pascanu et al. 2014)

Kiértékelés: negatív log-prob.

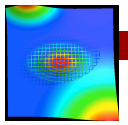
2. Kísérlet: karakter és szó szintű nyelvi modell, Penn Treebank Corpus

	RNN	DT(S)-RNN	DOT(S)-RNN	sRNN	*	⋆
Character-Level	1.414	1.409	**1.386**	1.412	1.41[1]	1.24[3]
Word-Level	117.7	112.0	**107.5**	110.0	123[2]	117[3]

ReLU szigmoid helyett a szó szintű modellnél.

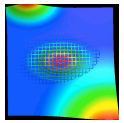
A két “baseline” modell:

- 1000 long short-term memory (Graves et al. 2013)
- shallow RNN nagz rejtett réteggel (Mikolov *et al*. 2011)



Megoldatlab problémák

- “Overfitting” (túltanulás):
  - DropOut [Hinton et al., 2012]
  - DropConnect [Wan et al., 2013]
- “Saturation”, “vanishing gradients” és ”sparsity”:
  - pReLU [Het et al., 2015]
  - Maxout [Goodfellow et al., 2013]
  - Network-in-Network [Lin et al., 2014]
  - local response/batch normalization [Ioffe et al., 2015]
  - BN Maxout NiN [Chang et al., 2015]
- Komplexitás:
  - Convolution
  - FastFood [Yang et al., 2015]
  - theoretical bounds [Lin&Tegmark, 2017, Rolnick et al., 2018]
- Struktúra:
  - spectral representation (pooling) [Rippel & Snoek, 2015]
  - Identity map és residual block [He et al., 2015]
  - Manifold tangent classifier, high-order contractive auto-encoder [Rifai et al., 2011]



Dropout (Hinton et al. 2012)

Deep hálók komplexitása rendkívül magas (VC theorem)

Következmények:

- Generalizáltságra kevés garancia vs. tapasztalat szerint működnek
- Túltanulás
- Túl sok paraméter -> lassan tanítható

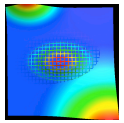
Ötlet: elkerülendő a túltanulást, a tanulás során véletlenszerűen hagyjunk ki neuronokat , képezve egy ún. “thinned” hálózatot. (elsőre még hasonló is lehetne valami pruning-hoz), hogy egymásra ne építhessenek a neuronok.

A keletkezett háló egy súlyozott, de generalizáltabb hálózat.

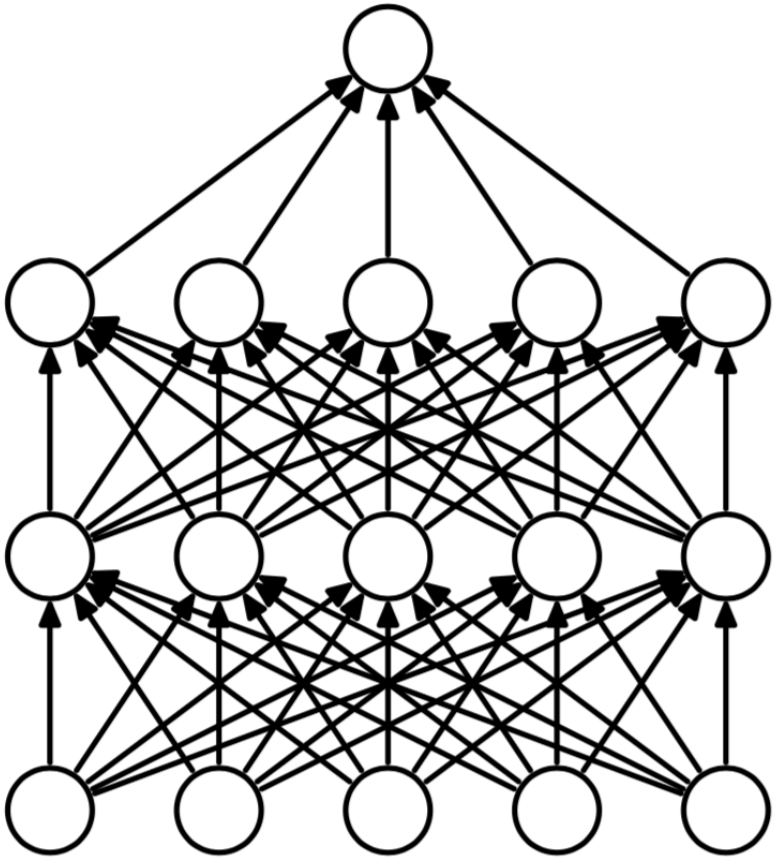
Vegülis egyfajta regularizáció.

Hipotézis: A dropout model aggregáció (avagy baggind (lesz lesz ☺))

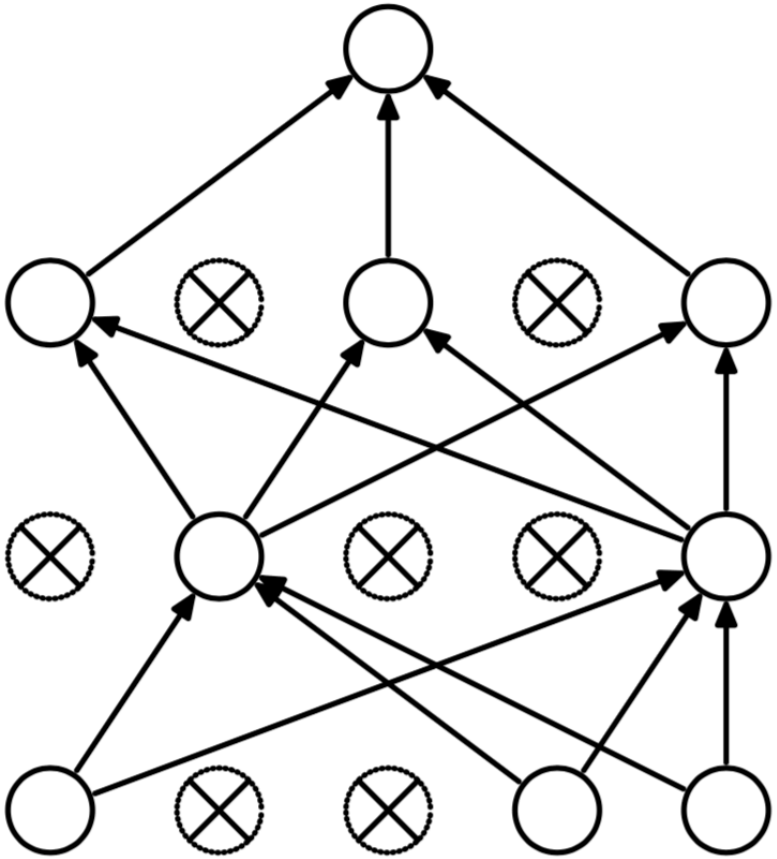
Nagy learning rate mellett működik jól (vs. manifold hipotézis?)



Dropout (Hinton et al. 2012)

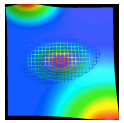


Klasszikus háló



Tanulás során

(ábra: Srivastava et al. 2014)



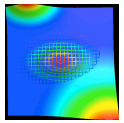
Dropout (Hinton et al. 2012)

klasszikus feed forward

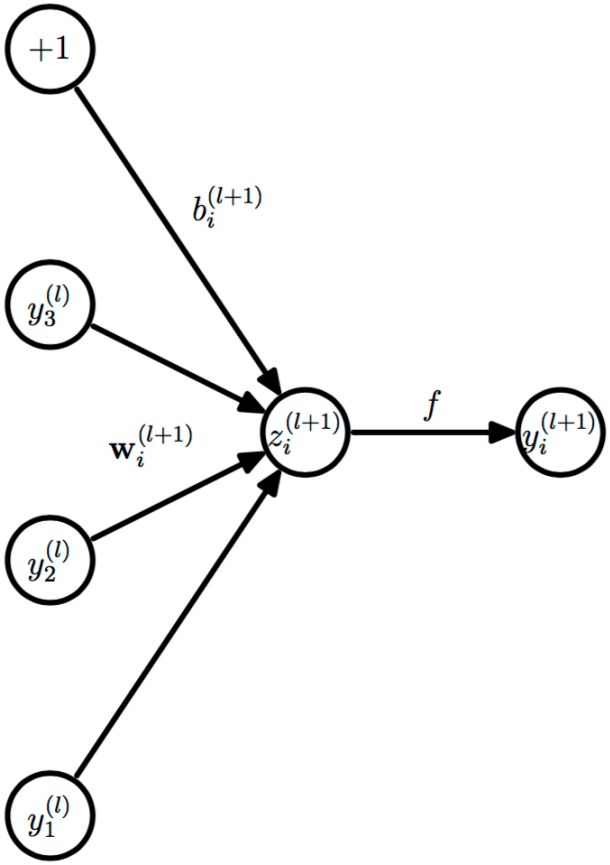
$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$

dropout

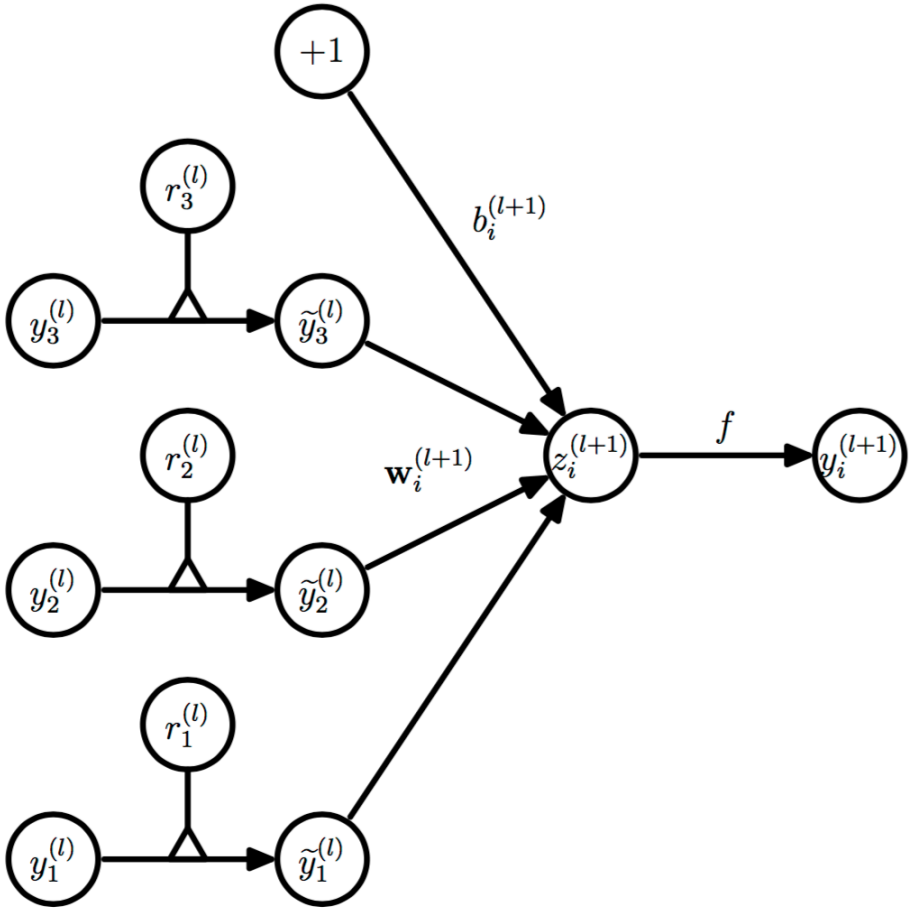
$$
\begin{aligned}
r_j^{(l)} &\sim \text{Bernoulli}(p), \\
\widetilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \widetilde{\mathbf{y}}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$



Dropout (Hinton et al. 2012)

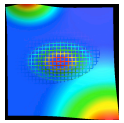


Klasszikus háló

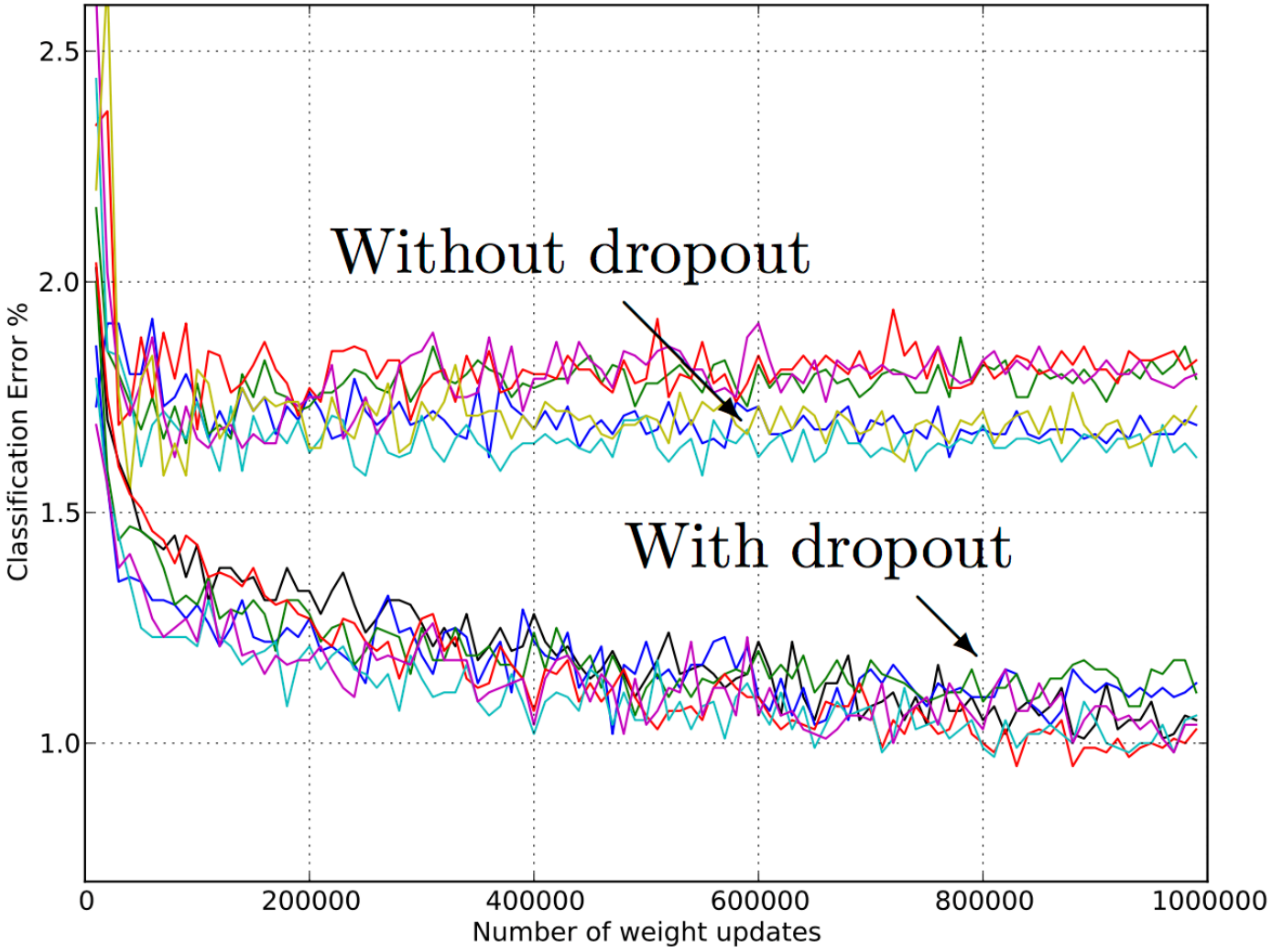


Dropout tanulás során

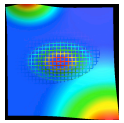
(ábra: Srivastava et al. 2014)



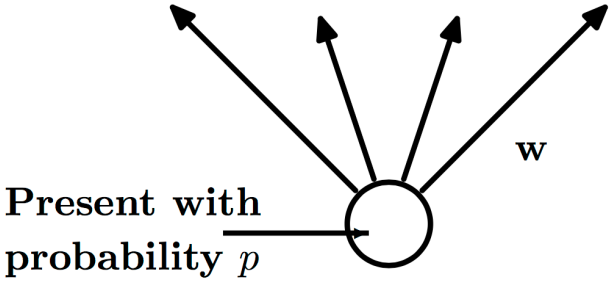
Dropout MNIST



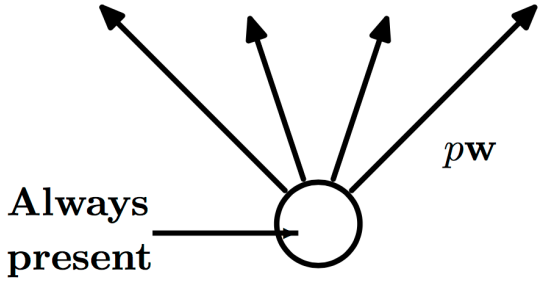
(ábra: Srivastava et al. 2014)



Dropout: kísérletek ILSVRC 2010



Tanulási fázis

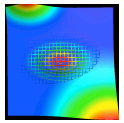


Teszt

Model	Top-1	Top-5
Sparse Coding (Lin et al., 2010)	47.1	28.2
SIFT + Fisher Vectors (Sanchez and Perronnin, 2011)	45.7	25.7
Conv Net + dropout (Krizhevsky et al., 2012)	37.5	17.0

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SVM on Fisher Vectors of Dense SIFT and Color Statistics	-	-	27.3
Avg of classifiers over FVs of SIFT, LBP, GIST and CSIFT	-	-	26.2
Conv Net + dropout (Krizhevsky et al., 2012)	40.7	18.2	-
Avg of 5 Conv Nets + dropout (Krizhevsky et al., 2012)	38.1	16.4	16.4

(ábrák, eredm. Srivastava et al. 2014)

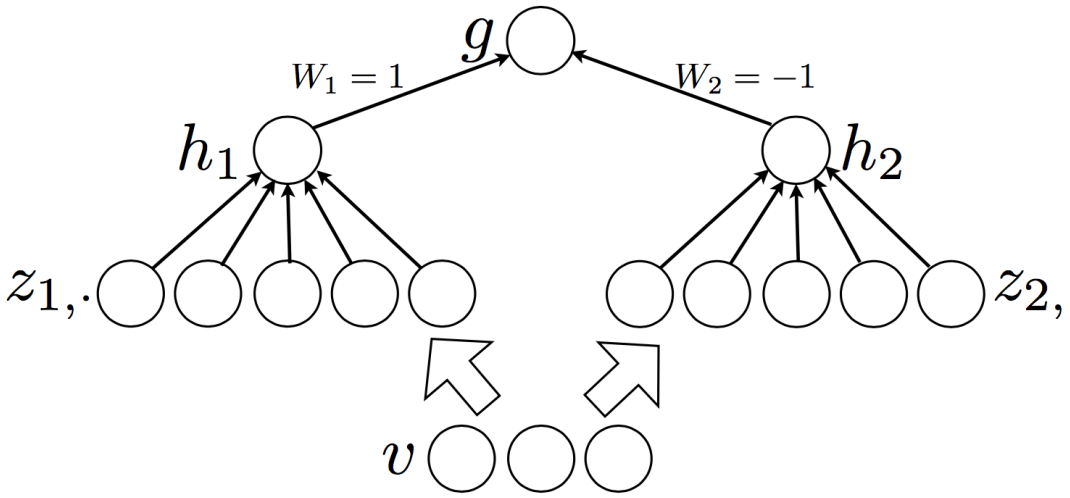


Dropout: MNIST

Method	Unit Type	Architecture	Error %
Standard Neural Net (Simard et al., 2003)	Logistic	2 layers, 800 units	1.60
SVM Gaussian kernel	NA	NA	1.40
Dropout NN	Logistic	3 layers, 1024 units	1.35
Dropout NN	ReLU	3 layers, 1024 units	1.25
Dropout NN + max-norm constraint	ReLU	3 layers, 1024 units	1.06
Dropout NN + max-norm constraint	ReLU	3 layers, 2048 units	1.04
Dropout NN + max-norm constraint	ReLU	2 layers, 4096 units	1.01
Dropout NN + max-norm constraint	ReLU	2 layers, 8192 units	0.95
Dropout NN + max-norm constraint (Goodfellow et al., 2013)	Maxout	2 layers, (5 × 240) units	0.94
DBN + finetuning (Hinton and Salakhutdinov, 2006)	Logistic	500-500-2000	1.18
DBM + finetuning (Salakhutdinov and Hinton, 2009)	Logistic	500-500-2000	0.96
DBN + dropout finetuning	Logistic	500-500-2000	0.92
DBM + dropout finetuning	Logistic	500-500-2000	**0.79**

(eredm. Srivastava et al. 2014)

Maxout (Goodfellow et al. 2013)

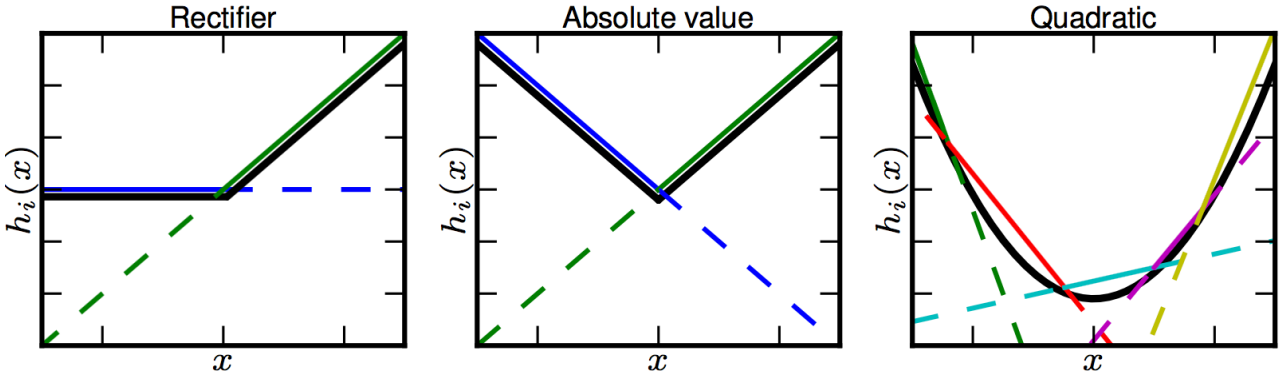


- MLP az aktivációs függvény helyett
- Piecewise linear approximation -> univerzális approximátor konvex függvényekre (ha elegendően sok affine elemünk van…)

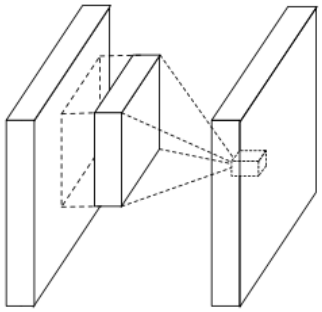
$$h_i(x) = \max_{j \in [1,k]} z_{ij}$$

ahol $x \in \mathbb{R}^d$

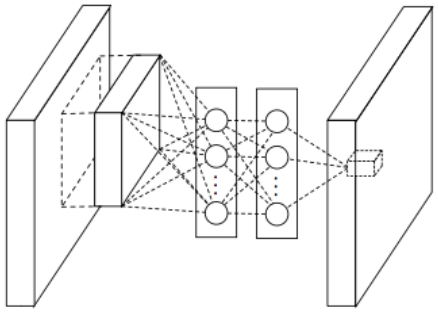
$$z_{ij} = x^T W_{\ldots ij} + b_{ij}$$



Network In Network (Lin et al., 2014)



(a) Linear convolution layer



(b) Mlpconv layer

$$f^1_{i,j,k_1} = \max(w^{1\,T}_{k_1} x_{i,j} + b_{k_1}, 0)$$
$$\vdots$$
$$f^n_{i,j,k_n} = \max(w^{n\,T}_{k_n} f^{n-1}_{i,j} + b_{k_n}, 0)$$

vs. maxout:

$$f_{i,j,k} = \max_m (w^T_{k_m} x_{i,j})$$

- Nem lineáris konvolúció
- Global average pooling (simítás)
  vs. fully connected

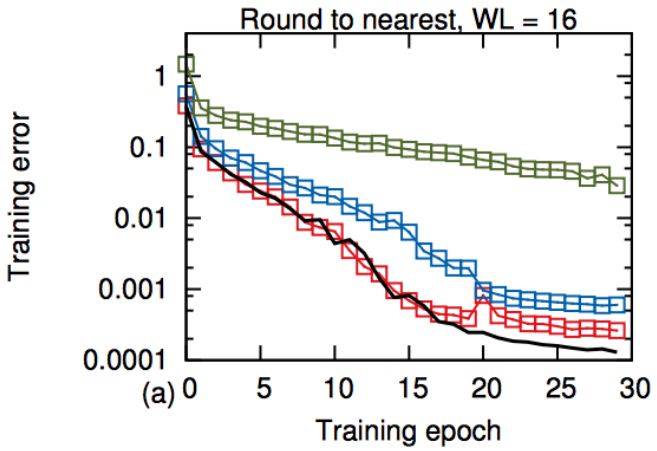
Method	Test Error
Stochastic Pooling [11]	15.13%
CNN + Spearmint [14]	14.98%
Conv. maxout + Dropout [8]	11.68%
NIN + Dropout	**10.41%**
CNN + Spearmint + Data Augmentation [14]	9.50%
Conv. maxout + Dropout + Data Augmentation [8]	9.38%
DropConnect + 12 networks + Data Augmentation [15]	9.32%
NIN + Dropout + Data Augmentation	**8.81%**

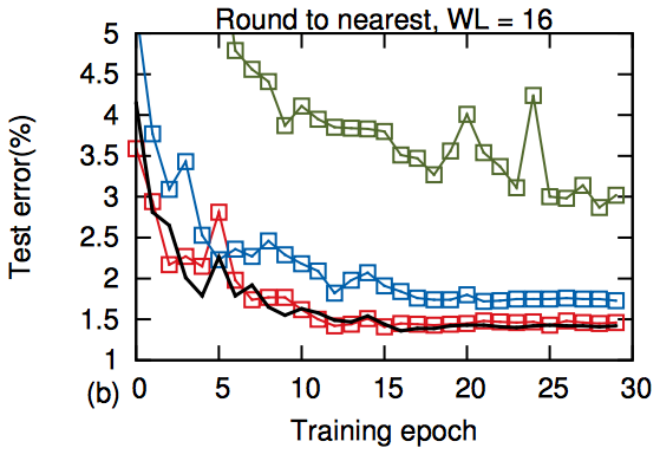
11.59% (FC) vs. 10.89% (FC+DO) vs. 10.41% (GAP) error CIFAR-10 adathalmazon

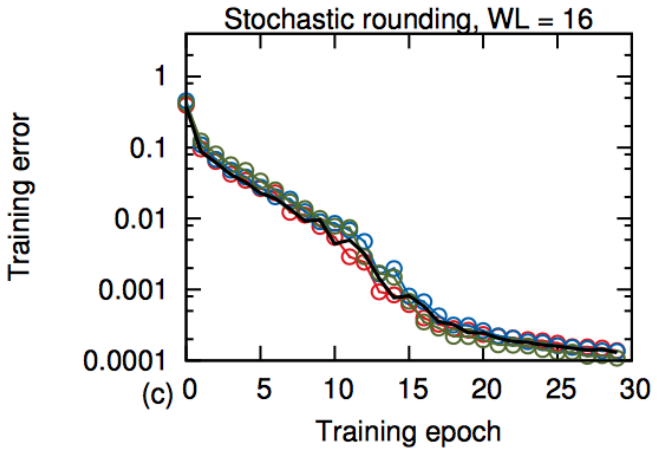
Stochasztikus kerekítés (Gupta et. al, 2015)

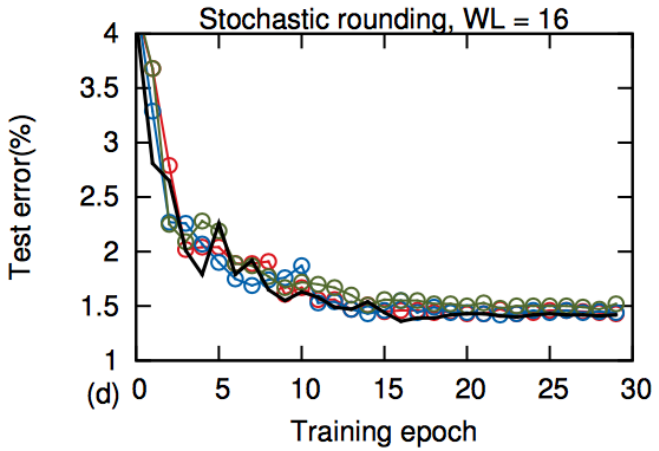
$$Convert\,(x, \langle \mathrm{IL}, \mathrm{FL} \rangle) = \begin{cases} -2^{\mathrm{IL}-1} & \text{if } x \leq -2^{\mathrm{IL}-1} \\ 2^{\mathrm{IL}-1} - 2^{-\mathrm{FL}} & \text{if } x \geq 2^{\mathrm{IL}-1} - 2^{-\mathrm{FL}} \\ Round(x, \langle \mathrm{IL}, \mathrm{FL} \rangle) & \text{otherwise} \end{cases}$$

$$Round\,(x, \langle \mathrm{IL}, \mathrm{FL} \rangle) = \begin{cases} \lfloor x \rfloor & \text{w.p. } 1 - \frac{x - \lfloor x \rfloor}{\epsilon} \\ \lfloor x \rfloor + \epsilon & \text{w.p. } \frac{x - \lfloor x \rfloor}{\epsilon} \end{cases}$$

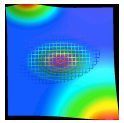






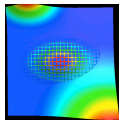


IL – Integer bits
FL – fractional bits
WL – word length



Pár implementáció

	Restrictions	Wrapper	Architectures	Notes
Theano	feed forward/ recurrent hálók	Python	Multi core/CUDA	
Chainer	feed forward/ recurrent hálók	Python	Multi core(?)/ CUDA	
GraphLab	Feed Forward: CNN, DBN	Python/C++	Multi core/CUDA/ distributed	Compact, Hadoop
TensorFlow	feed forward/ recurrent hálók	Python/C++	Multi core/CUDA/ distributed	
Caffe	Feed Forward: CNN, DBN	Python/Matlab	Multi core/CUDA	Blob
Torch	feed forward/ recurrent hálók	Lua	Multi core/CUDA	Eredetileg vizuális adatokra

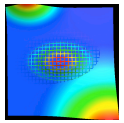


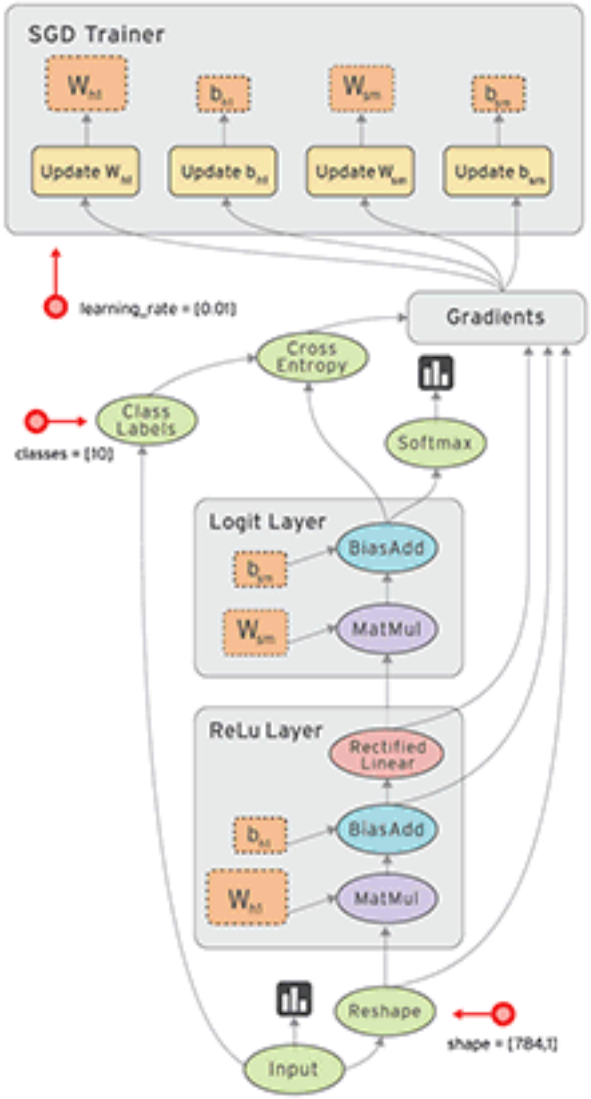
Mielőtt TensorFlow

Sok dolog kimaradt: LSTM, Deep Fried CNN stb.

Pár megjegyzés:

- Elméleti korlátok
- Generatív modell
- Igazi pruning és boosting módszerek még hiányoznak





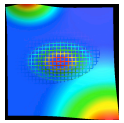
TensorFlow ™

Google-nél fejlesztették, alapvetően eloszott, többmagos vagy éppen GPU-s környezetben is használható (utóbbi csak Linux)

Python és C++

Sokféle háló: CNN, DBN, RNN, LSTM

Példa: MNIST



TensorFlow ™

Telepítés:

Lépjünk be a saját környezetünkbe.

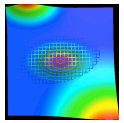
Mi: Ubuntu/Linux 64-bit, CPU only, Python 3.4

export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.11.0rc1-cp34-cp34m-linux_x86_64.whl

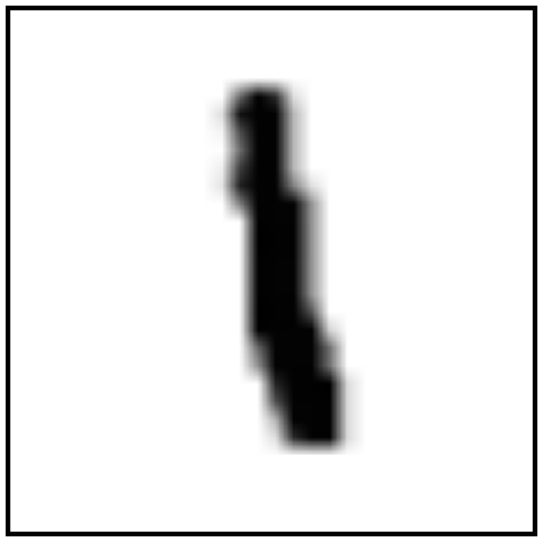
sudo pip install --upgrade $TF_BINARY_URL

Teszt:

Import tensorflow as ts



MNIST kézírásfelismerés



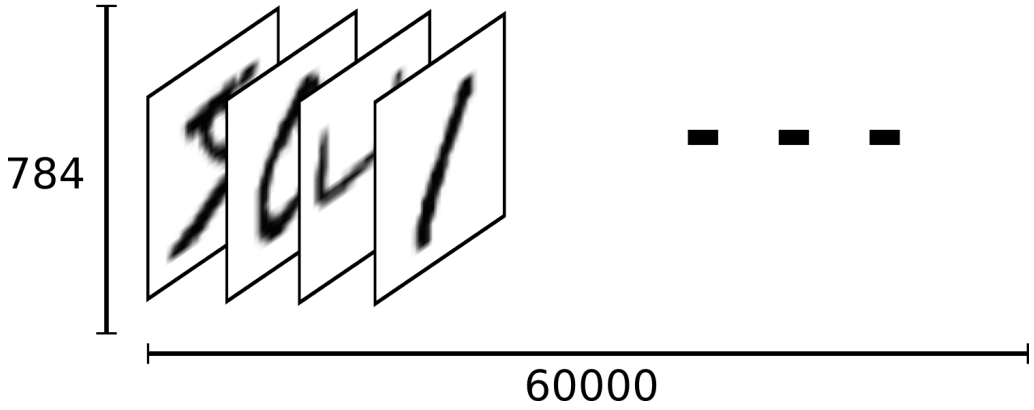
≈

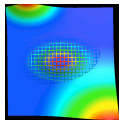
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	.6	.8	0	0	0	0	0	0
0	0	0	0	0	0	.7	1	0	0	0	0	0	0
0	0	0	0	0	0	.7	1	0	0	0	0	0	0
0	0	0	0	0	0	.5	1	.4	0	0	0	0	0
0	0	0	0	0	0	0	1	.4	0	0	0	0	0
0	0	0	0	0	0	0	1	.4	0	0	0	0	0
0	0	0	0	0	0	0	1	.7	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	.9	1	.1	0	0	0	0
0	0	0	0	0	0	0	.3	1	.1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

MNIST:

- 60000 kép
- 28x28 felbontás
- egy csatorna
- Tíz cimke: 0..9

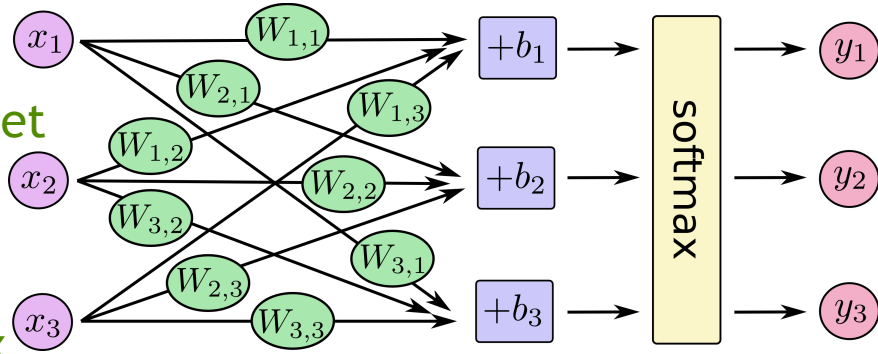
mnist.train.xs



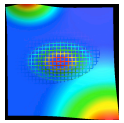


MNIST kézírásfelismerés (shallow)

```
# mint előbb a graphlab-nál, itt is a tutorialt követjük
import tensorflow as tf

# beállítjuk az alap változókat és az bemenetet
x = tf.placeholder("float", [None, 784])
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))
# aktiváló függvényünk többosztályos softmax
y = tf.nn.softmax(tf.matmul(x,W) + b)

# eredeti osztálycimkék
y_ = tf.placeholder("float", [None,10])
# célfüggvény
cross_entropy = -tf.reduce_sum(y_*tf.log(y))

# generálunk egy update lépést
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
```

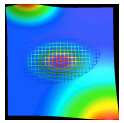


$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$



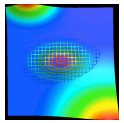
MNIST kézírásfelismerés (shallow)

```
# elindítunk egy session-t, de előbb megadjuk az inicializáció-t
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)

# 1000 lépésben tanítunk batch alapon
for i in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

# kiértékelünk
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
print sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels})
```



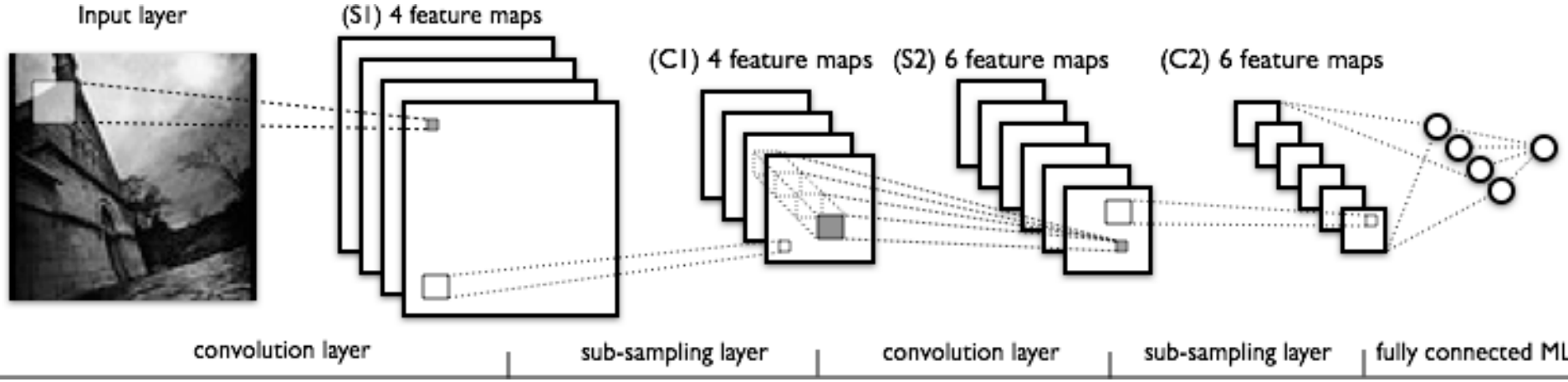
MNIST kézírásfelismerés (CNN)

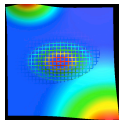
```
# mint a nevéből is adódik, jobban illeszkedik az interaktív python környezetekbe
sess = tf.InteractiveSession()

# definiálunk egyszerű objektumokat az inicializációhoz
def weight_variable(shape):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)

def bias_variable(shape):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)

# egyszerű 2d conv : lépésköz 1, W a filter formája, x maga a kép amire alkalmazzuk
def conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

# alap mintavétel, x-ből 2x2-es csoportonként lesz egy negyed annyi elemből álló kép
def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
```



MNIST kézírásfelismerés (CNN)

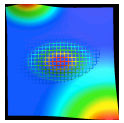
```
# első és második konvolúciós és mintavételi réteg
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
x_image = tf.reshape(x, [-1,28,28,1])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)
```





MNIST kézírásfelismerés (CNN)

```
# teljesen összekapcsolt réteg a 7x7-es képekből (64) egy 1024 dimenziós flat rejtett réteg
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])

# ReLU
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

# dropout során ez a réteg megmarad!
keep_prob = tf.placeholder("float")
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# Kimeneti réteg, megint softmax
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
y_conv=tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
```



MNIST kézírásfelismerés (CNN)

```
# ugyanaz a célfüggvény
cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))

# ADAM optimizer (Kingma et al. 2015)
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
sess.run(tf.initialize_all_variables())
for i in range(1000):
  batch = mnist.train.next_batch(50)
  if i%100 == 0:
    train_accuracy = accuracy.eval(feed_dict={
        x:batch[0], y_: batch[1], keep_prob: 1.0})
    print "step %d, training accuracy %g"%(i, train_accuracy)
  train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})

print "test accuracy %g"%accuracy.eval(feed_dict={
    x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0})
```