ML Praktikum WS 17/18 Introduction to Deep Learning

Martin Mundt

FIAS / Goethe Uni Frankfurt

February 2018

What is deep learning?

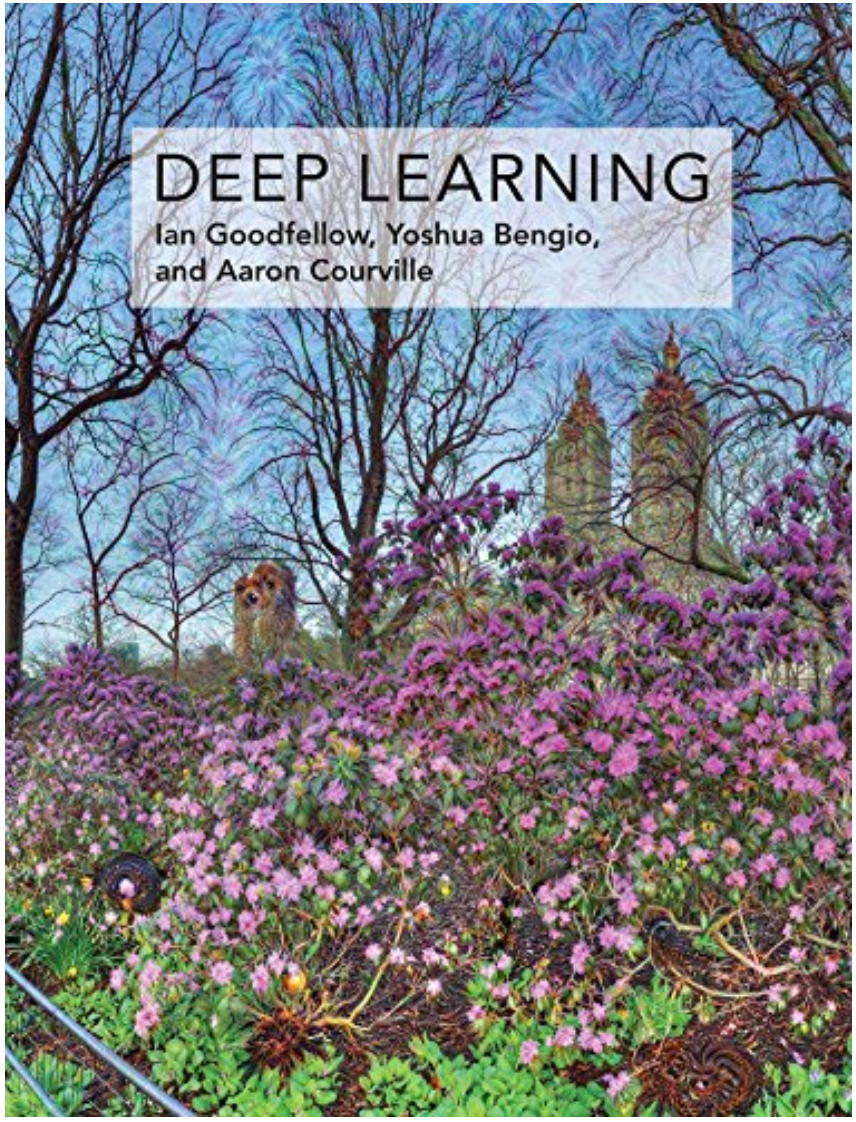
"Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction"

LeCun,Bengio,Hinton,2015

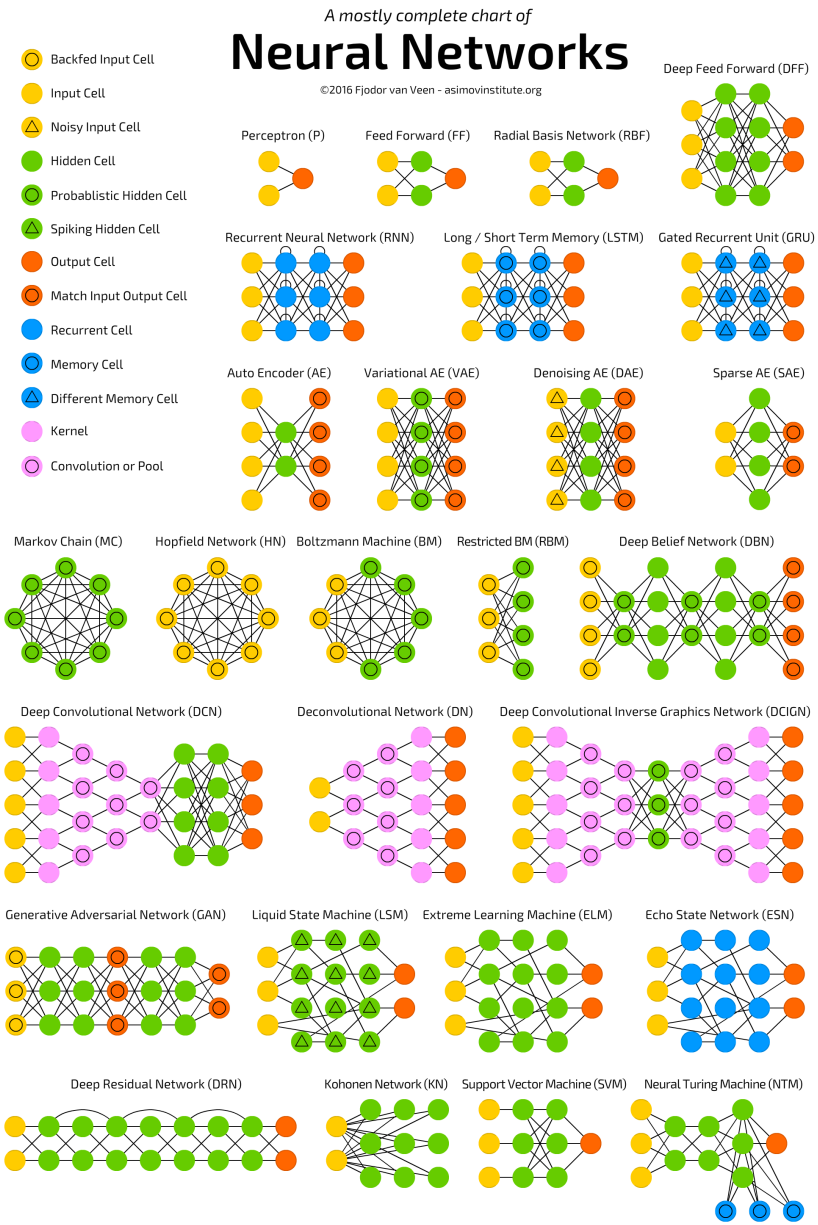
Ideas behind Deep Learning

- Inspiration from brain's hierarchy (V1,V2,.. etc.)
  $\rightarrow$ hierarchical networks with different interactions on multiple levels
- Learning features instead of engineering
- Derive intermediate representations, with increasing level of abstraction
- Generalization ability
- Latent/hidden variables
- Depending on exact algorithm either super- or unsupervised

Deep Learning book: Goodfellow, Bengio, Courville; MIT Press 2016



The Neural Network Zoo



Basic concepts of a convolution neural network (CNN)

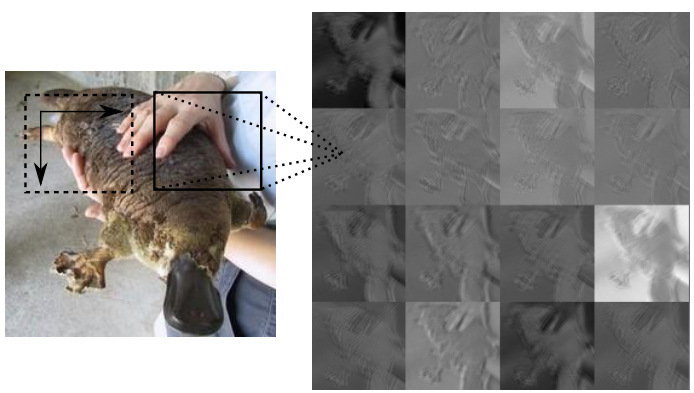
- Basically a neural network using many identical copies of a neuron
- Sparse connectivity pattern & replication re-uses parametrization
- Large amount of neurons (e.g. 650 k) $\rightarrow$ large models $\rightarrow$ "small" amount of parameters (e.g. 60 M)
- Composable network with each layer detecting more abstract "higher-level" features
- Typically trained with modified versions of (stochastic) gradient descent
- Convolution can work on high-dimensional data
- Very good at specific tasks based on: **images**, videos, audio...

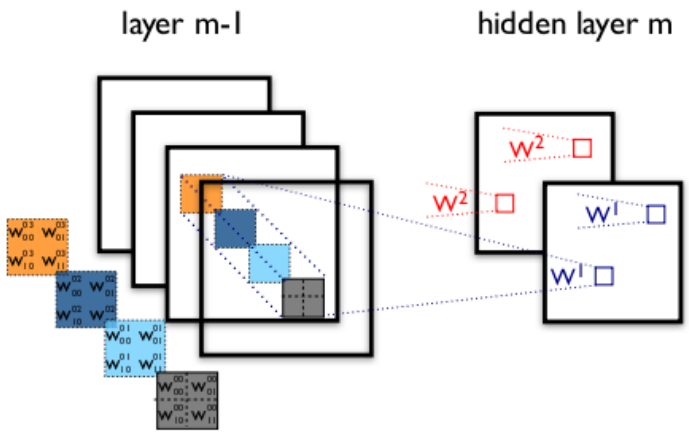
Convolutions

- Identical copies of neurons implies the same weights in more than 1 position $\rightarrow$ weight-sharing

$$x_{ijf_{\ell+1}}^{\ell+1} = \sum_{f_\ell}^{F_\ell} \sum_{a=0}^{N_A^{\ell+1}-1} \sum_{b=0}^{N_B^{\ell+1}-1} W_{abf_\ell f_{\ell+1}}^{\ell+1} y_{(i+a)(j+b)f_\ell}^{\ell} - b_{ijf_{\ell+1}}^{\ell+1}$$

$$\underline{\underline{x}}^{\ell+1} = \underline{\underline{W}}^{\ell+1} \star \underline{\underline{y}}^{\ell} - \underline{\underline{b}}^{\ell+1}$$





[http://deeplearning.net/tutorial/lenet.html]

Activations (Transfer)

$$\underline{\underline{y}}^{\ell+1} = \sigma\left[\underline{\underline{x}}^{\ell+1}\right] = \sigma\left[\underline{\underline{W}}^{\ell+1} \star \underline{\underline{y}}^{\ell} - \underline{\underline{b}}^{\ell+1}\right]$$

Activation functions $\sigma(x)$:

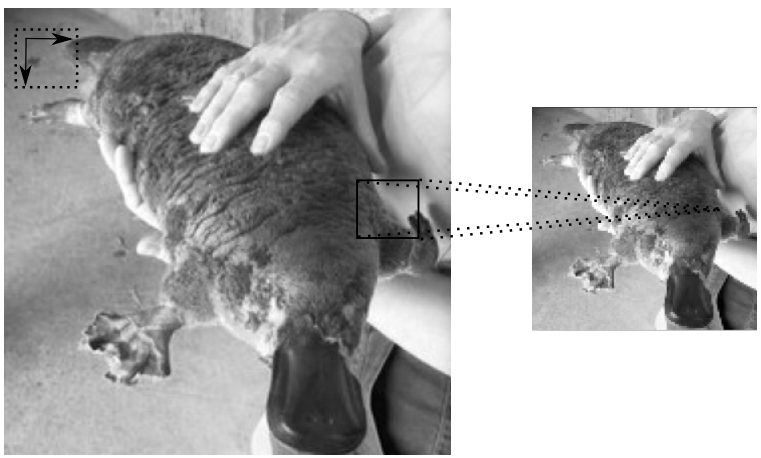
- Tanh: $f(x) = tanh(x)$ or Sigmoid: $f(x) = (1 + e^{-x})^{-1}$
- ReLU (Rectified Linear Unit): $f(x) = max(0, x)$
- Rectified sigmoidal functions etc.

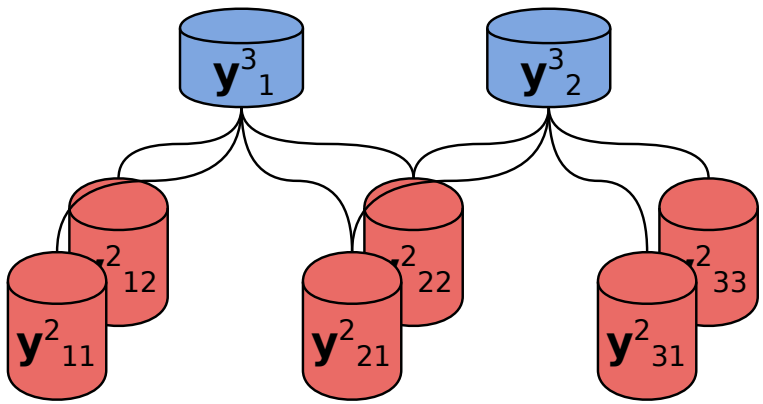
Vanishing gradient problem?

Pooling

- Dimensionality reduction
- Introduction of invariance (translation, rotation)
- Usually no learning involved

$$y^{\ell+1}(i'j') = \max_{ij \in K(i'j')} y^{\ell}(ij)$$





Classification

- Multi-layer perceptron (MLP) (expressed via convolutions)
- Feature maps reduced in size due to previous transformations
- Map onto classes $\rightarrow$ generate probability distribution (e.g. softmax)
- "Grandmother cells" in the final layer

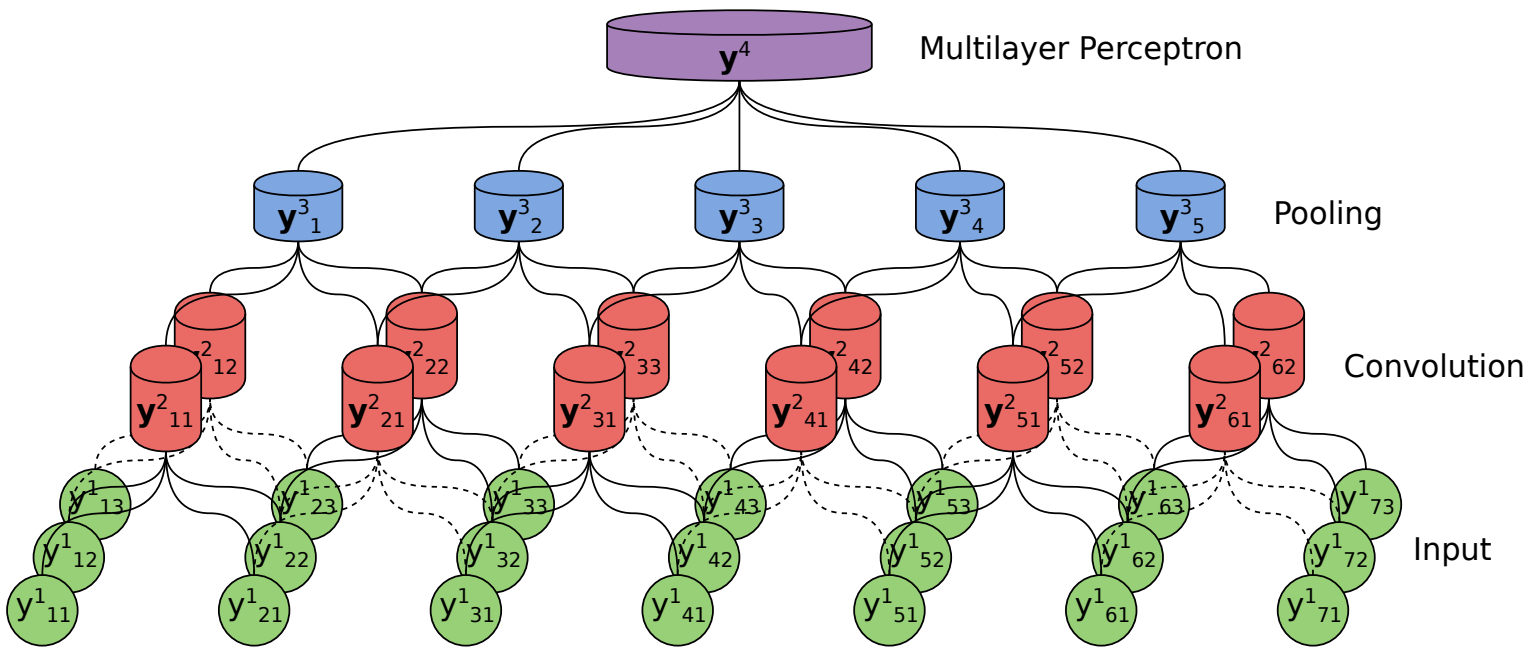


Image segmentation

- Other tasks than classification are possible by extending the "encoder"/ learned feature representation with other functions
- For image segmentation we can "flip" the CNN and add this so called "Deconvolution NN" on top.

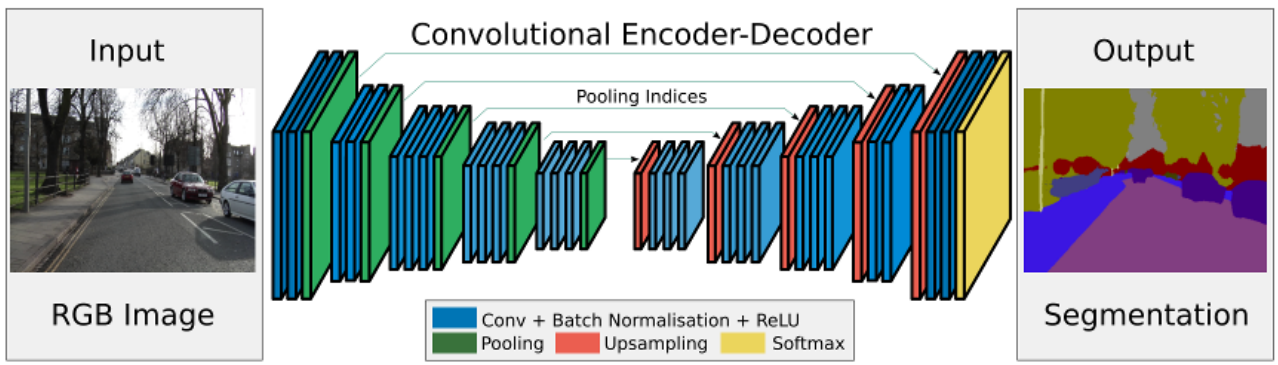
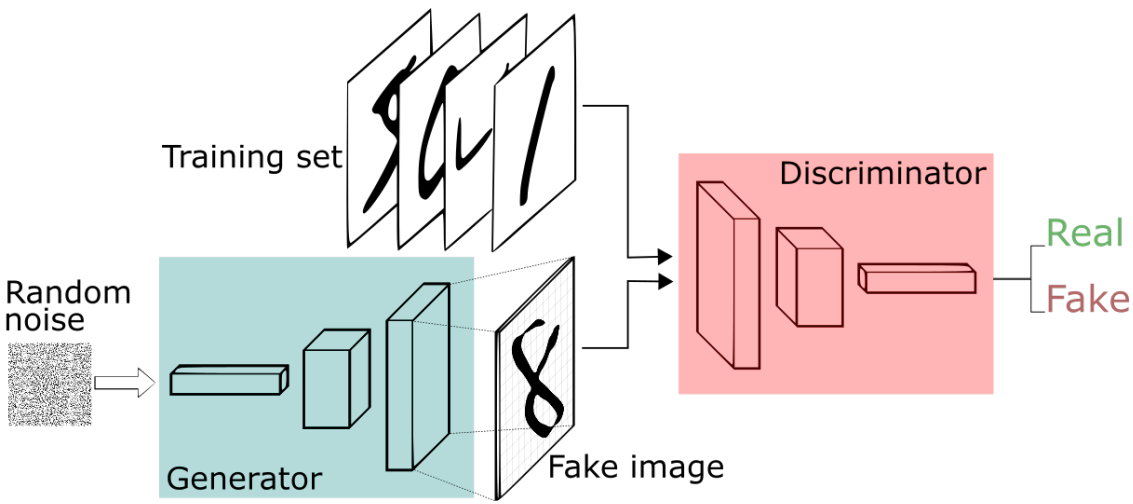


Fig. 2. An illustration of the SegNet architecture. There are no fully connected layers and hence it is only convolutional. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). It then performs convolution with a trainable filter bank to densify the feature map. The final decoder output feature maps are fed to a soft-max classifier for pixel-wise classification.

"SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation"
Badrinarayanan, Kendall, Cipolla 2015

Generation using CNN based Generative Adversarial Networks (GAN)

- Various methods to use deconvolutional NNs to generate data
- GANs are 1 option: filters of the DeconvNet are trained to generate output that resembles some training data to fool a different "discriminator" (C)NN.

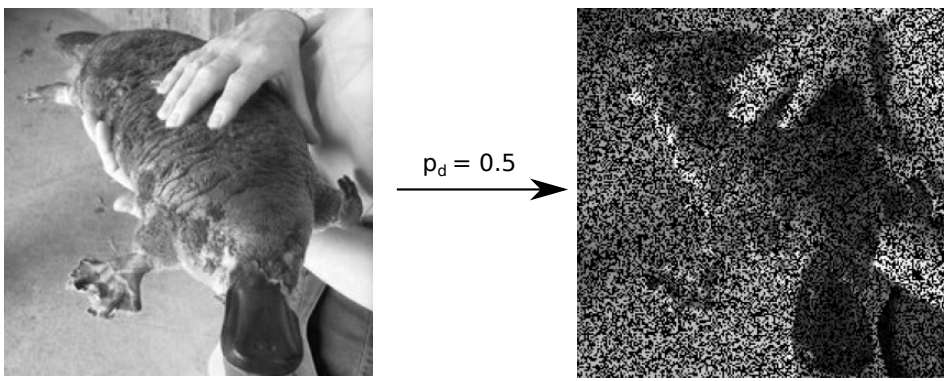


https://deeplearning4j.org/generative-adversarial-network

Dropout (an example for regularization)

- Introduction of noise $\rightarrow$ reduces complex co-adaptation of neurons

$$y_{ij}^{\ell} = \alpha(p_d) y_{ij}^{\ell} \quad \texttt{with} \quad \alpha(p_d) = \begin{cases} 0 & \text{if } u \in U(0,1) < p_d \\ 1 & \text{if } u \in U(0,1) \geq p_d \end{cases}$$



G.E. Hinton: "The brain has about 10^{14} *synapses and we only live for about* 10^9 *seconds. So we have a lot more parameters than data. This motivates the idea that we must do a lot of unsupervised learning since the perceptual input (including proprioception) is the only place we can get* 10^5 *dimensions of constraint per second.*

Weight initialization

- Especially for very deep NNs initializing weights can be tricky & is an active research topic (shallow networks are more "forgiving").
- There is rules of thumb for how weights can be initialized e.g. "Xavier initialization" or "Kaiming initialization".
- Such initialization techniques draw from distributions that are scaled with different size properties of the NN architecture design

For m inputs, n outputs and the slope of the activation function a (e.g. $a = 0$ for ReLUs):

$$\texttt{Glorot2010} \qquad W_{i,j} \sim \mathcal{U}\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right) \tag{1}$$

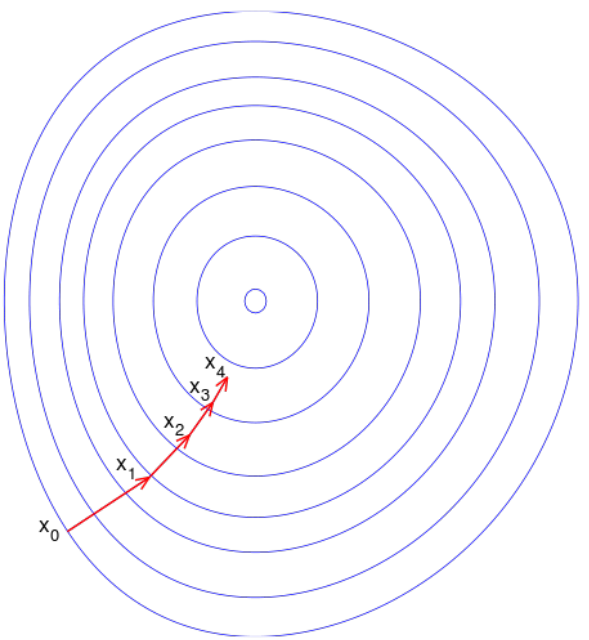
$$\texttt{Kaiming2015} \qquad W_{i,j} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{(1+a^2)\cdot m}}\right) \tag{2}$$

Gradient descent

$$C(\theta) = \frac{1}{N} \sum_{n=1}^{N} \mathfrak{L}_n(\theta)$$

GD:
while $||\nabla_\theta C(\theta)||^2 \geq \epsilon$ **and** $t \leq t_E$
$\theta_{t+1} \leftarrow \theta_t - \eta_t \nabla_\theta C(\theta)$
$t \leftarrow t + 1$
end while

SGD:
while $||\nabla_\theta \mathfrak{L}_n(\theta)||^2 \geq \epsilon$ **and** $t \leq t_E$
$\theta_{t+1} \leftarrow \theta_t - \eta_t \nabla_\theta \mathfrak{L}_n(\theta)$
$t \leftarrow t + 1$
end while



Backpropagation

- Werbos 1983, Rumelhart 1986
- Train deeper networks

Why after 20 years?

- Data
- Computational power
- Initialization
- Subtleties: ReLU, dropout ...

Backpropagation

Find an expression for the rate of change of the cost/loss function with respect to any weight in the network to minimize the cost by optimizing weights:

$$\frac{\partial \mathfrak{L}_n(W, b, x, y_{label})}{\partial W^{\ell}_{ab}} = \sum_{i=0}^{N^{\ell}_I - N^{\ell}_A} \sum_{j=0}^{N^{\ell}_J - N^{\ell}_B} \frac{\partial \mathfrak{L}_n}{\partial x^{\ell}_{ij}} \frac{\partial x^{\ell}_{ij}}{\partial W^{\ell}_{ab}}$$

$$= \sum_{i=0}^{N^{\ell}_I - N^{\ell}_A} \sum_{j=0}^{N^{\ell}_J - N^{\ell}_B} \frac{\partial \mathfrak{L}_n}{\partial x^{\ell}_{ij}} y^{\ell-1}_{(i+a)(j+b)}$$

the sums correspond to weight sharing and the latter derivative is known from forward propagation

Backpropagation

$$\frac{\partial \mathfrak{L}_{\mathfrak{n}}}{\partial x_{ij}^{\ell}} = \frac{\partial \mathfrak{L}_{n}}{\partial y_{ij}^{\ell}} \frac{\partial y_{ij}^{\ell}}{\partial x_{ij}^{\ell}} = \frac{\partial \mathfrak{L}_{n}}{\partial y_{ij}^{\ell}} \frac{\partial}{\partial x_{ij}^{\ell}} \left[\sigma \left(x_{ij}^{\ell} \right) \right]$$
$$= \frac{\partial \mathfrak{L}_{n}}{\partial y_{ij}^{\ell}} \sigma' \left(x_{ij}^{\ell} \right)$$

- $\frac{\partial \mathfrak{L}}{\partial x}$ has large value $\rightarrow$ cost can be lowered in $\sigma(x + \Delta x)$
- $\frac{\partial \mathfrak{L}}{\partial x} \approx 0$ insignificant improvement
- Heuristic definition of error as system must be near minimum if the derivative has a small value $\frac{\partial \mathfrak{L}}{\partial x} \equiv \delta_{ij}^{\ell}$

Now calculate $\frac{\partial \mathfrak{L}}{\partial x_{ij}^{\ell-1}} \equiv \delta_{ij}^{\ell-1}$ etc.

Backpropagation

4 equations for CNN backpropagation:

$$\underline{\underline{\delta}}^{L} = \nabla_x \mathfrak{L}(W, b, x, y_{label}) \circ \sigma'\left(\underline{\underline{x}}^{L}\right)$$

$$\frac{\partial \mathfrak{L}_n(f_W, xy)}{\partial \underline{\underline{x}}^{\ell}} = \underline{\underline{\delta}}^{\ell} = \left(\underline{\underline{W}}^{\ell+1,\top} \star \underline{\underline{\delta}}^{\ell+1}\right) \circ \sigma'\left(\underline{\underline{x}}^{\ell}\right)$$

$$\frac{\partial \mathfrak{L}_n(f_W, xy)}{\partial \underline{\underline{W}}^{\ell}} = \underline{\underline{\delta}}^{\ell} \star \underline{\underline{y}}^{\ell-1}$$

$$\frac{\partial \mathfrak{L}_n(f_W, xy)}{\partial \underline{\underline{b}}^{\ell}} = \underline{\underline{\delta}}^{\ell}$$

Backpropagation Algorithm

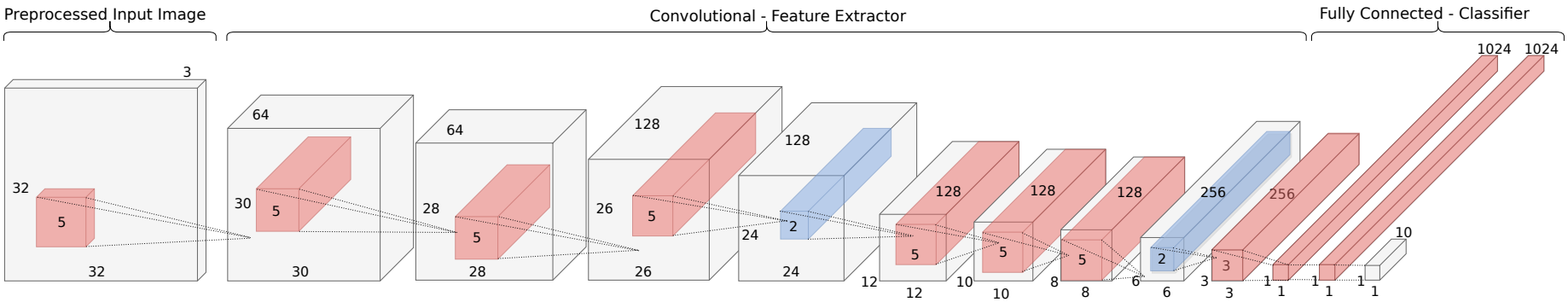
Backprop with mini-batch SGD:

1. **Input:** for n training examples set $\underline{\underline{y}}_{n}^{0}$
2. **Forward:** $\underline{\underline{y}}_{n}^{\ell+1} \leftarrow \sigma\left[\underline{\underline{W}}^{\ell+1} \star \underline{\underline{y}}_{n}^{\ell} - \underline{\underline{b}}^{\ell+1}\right]$
3. **Calculate error:** $\underline{\underline{\delta}}_{n}^{L} \leftarrow \nabla_{x}\mathfrak{L}_{n}(W, b, x, y_{label}) \circ \sigma'\left(\underline{\underline{x}}_{n}^{L}\right)$
4. **Backprop error:** $\underline{\underline{\delta}}_{n}^{\ell} \leftarrow \left(\underline{\underline{W}}^{\ell+1,\top} \star \underline{\underline{\delta}}_{n}^{\ell+1}\right) \circ \sigma'\left(\underline{\underline{x}}_{n}^{\ell}\right)$
5. **Update:** $\underline{\underline{W}}^{\ell} \leftarrow \underline{\underline{W}}^{\ell} - \frac{\eta}{N}\sum_{n} \underline{\underline{\delta}}_{n}^{\ell} \star \underline{\underline{y}}_{n}^{\ell-1}$
   $\left(\underline{\underline{b}}^{\ell} \leftarrow \underline{\underline{b}}^{\ell} - \frac{\eta}{N}\sum_{n} \underline{\underline{\delta}}_{n}^{\ell}\right)$

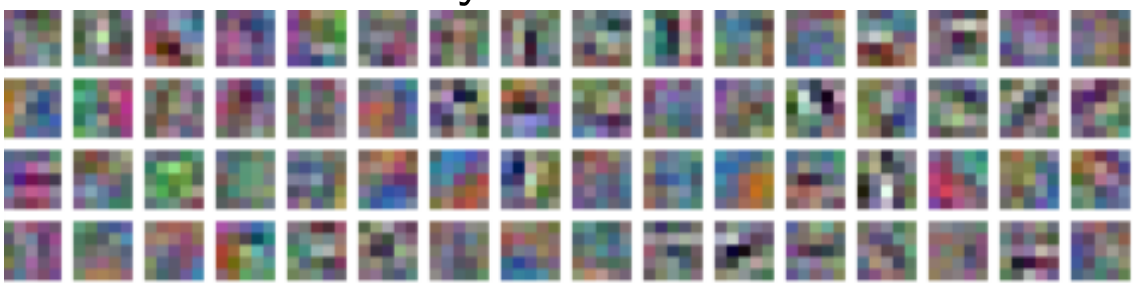
CIFAR-10

- labeled subset of 80 million tiny images dataset
- 50k training, 10k test 32x32 color images
- 10 classes (mutually exclusive) with 6000 images per class
- 6 randomly-selected batches of 10k images
- airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck

a DCNN example



Layer 1 filters

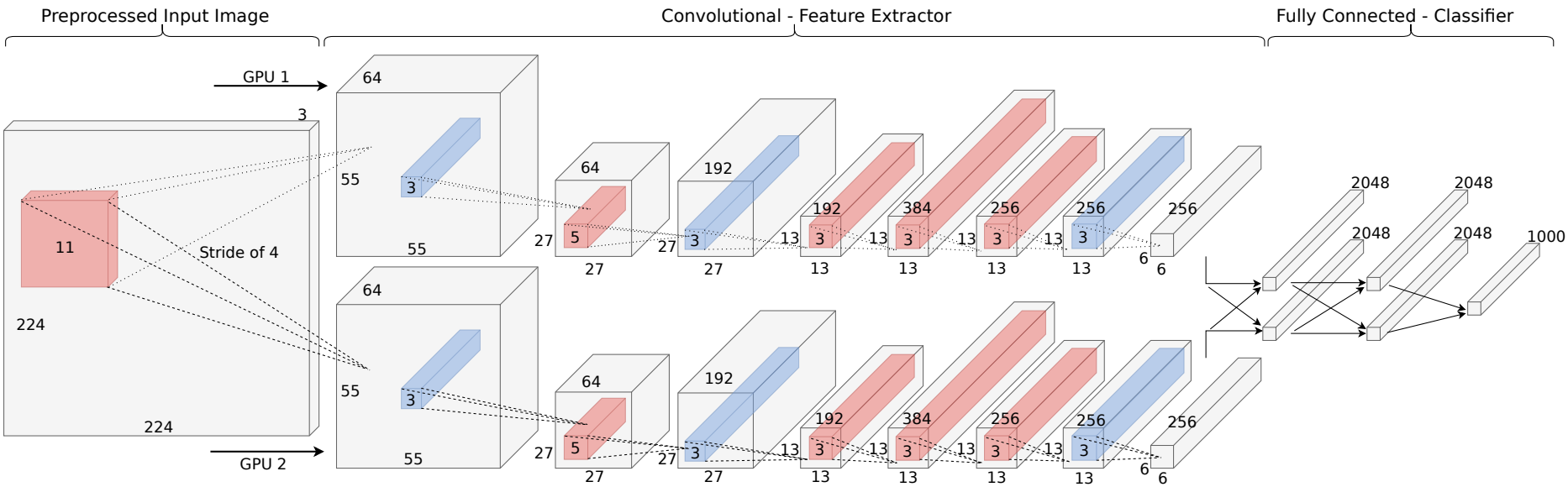


- ≈ 1-2 hours depending on GPU

ImageNet ILSVRC-12

- held in conjunction with PASCAL VOC, results "open" or "closed"
- subset of ImageNet dataset (10M images, 10k classes)
- 1000 classes, 1.2M training images and 150k testing images
- up to 5 classes per image
- classification, detection (& fine-grained classification)

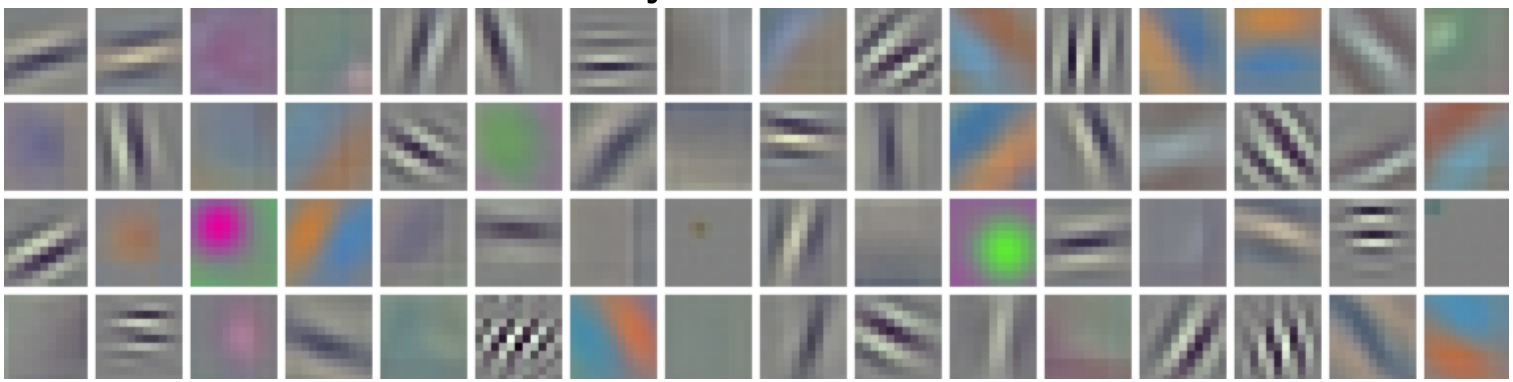
AlexNet-OWT [Krizshevsky 2014]



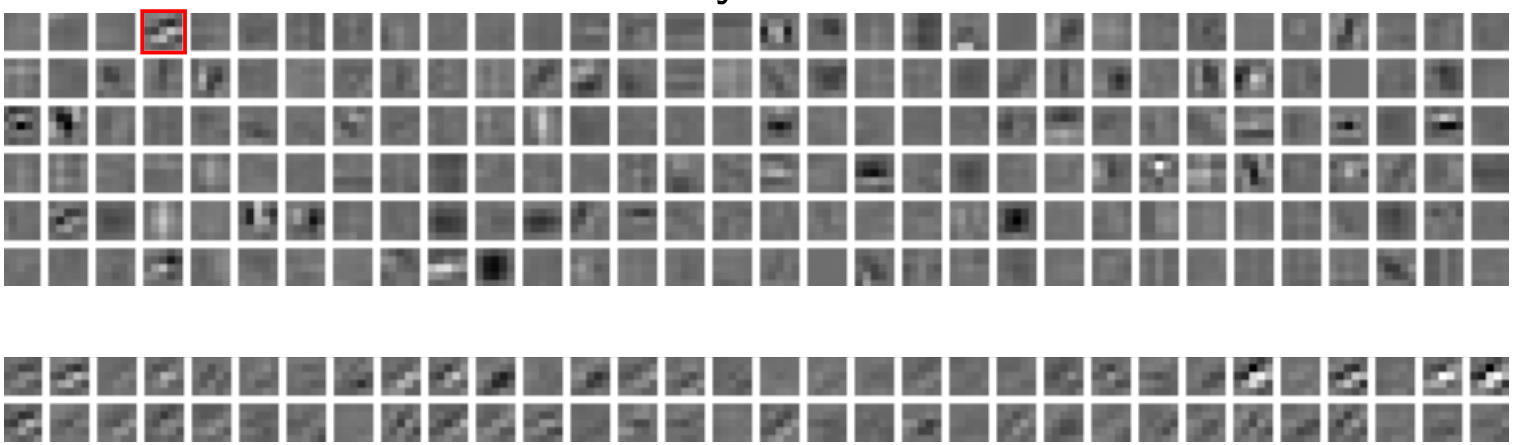
- Trained on 2 GPUs
- 43.4% top-1 error
- $\approx$ a day including cross-validation and testing
- data-parallelism in convolutional layers
- model-parallelism in fully-connected layers

AlexNet-OWT

Layer 1 filters



Some layer 2 filters



Residual Neural Networks

Re-use information & combat vanishing gradients

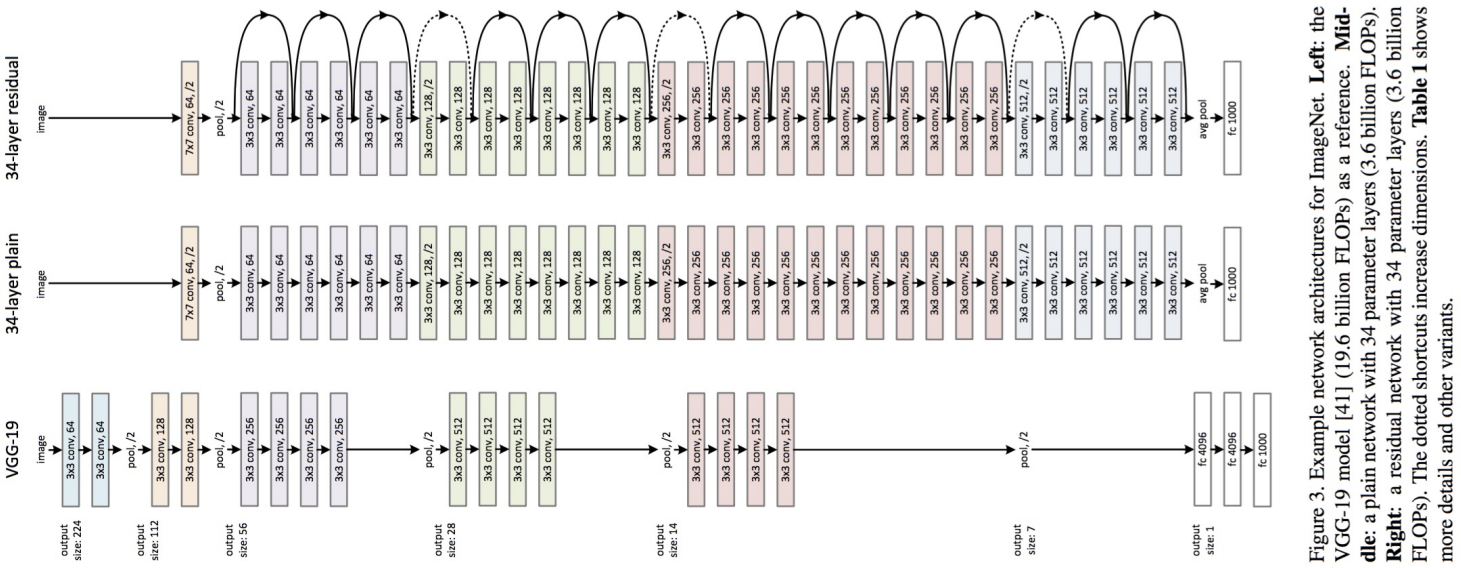


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

"Deep Residual Learning for Image Recognition"; He, Zhang, Ren, Sun; 2015

DenseNet

Extension to ResNets

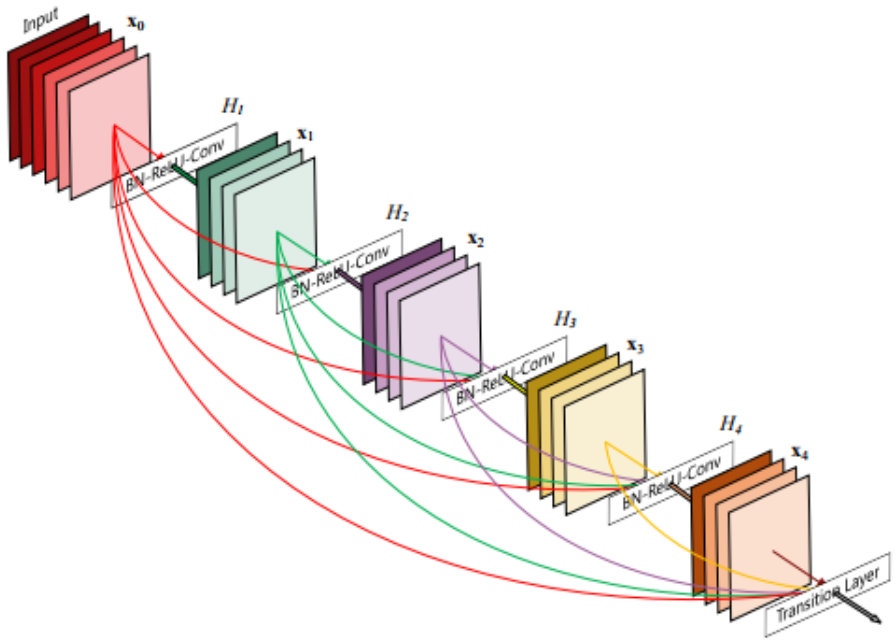


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

"Densely Connected Convolutional Networks"; Huang, Liu, van der Maaten, Weinberger; CVPR best paper 2017

Sequence models

Example: Natural Language Processing (NLP)

examples loosely as presented in https://www.coursera.org/learn/nlp-sequence-models

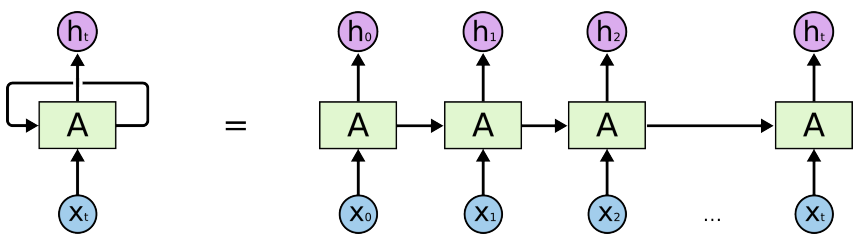
"Harry Potter discovered a beautiful spell."

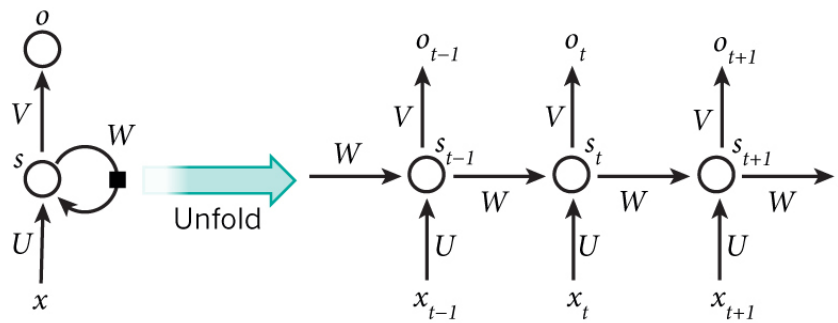
$x^{<1>} = \text{"}\texttt{Harry}\text{"}, \quad x^{<2>} = \text{"}\textit{Potter}\text{"} \ldots x^{<6>} = \text{"}\textit{spell}\text{"}$

- Many potential tasks that have different mappings:
  - many-to-one: e.g. sentiment analysis
  - one-to-many: e.g. music generation
  - many-to-many e.g. entity finding, translation

(Simple) RNNs

- Activations of previous time-step influence next time-step
- Weights operating on $x^{<t>}$ shared across t
- In addition now also weights operating between $A^{<t>}$ (s in right figure)

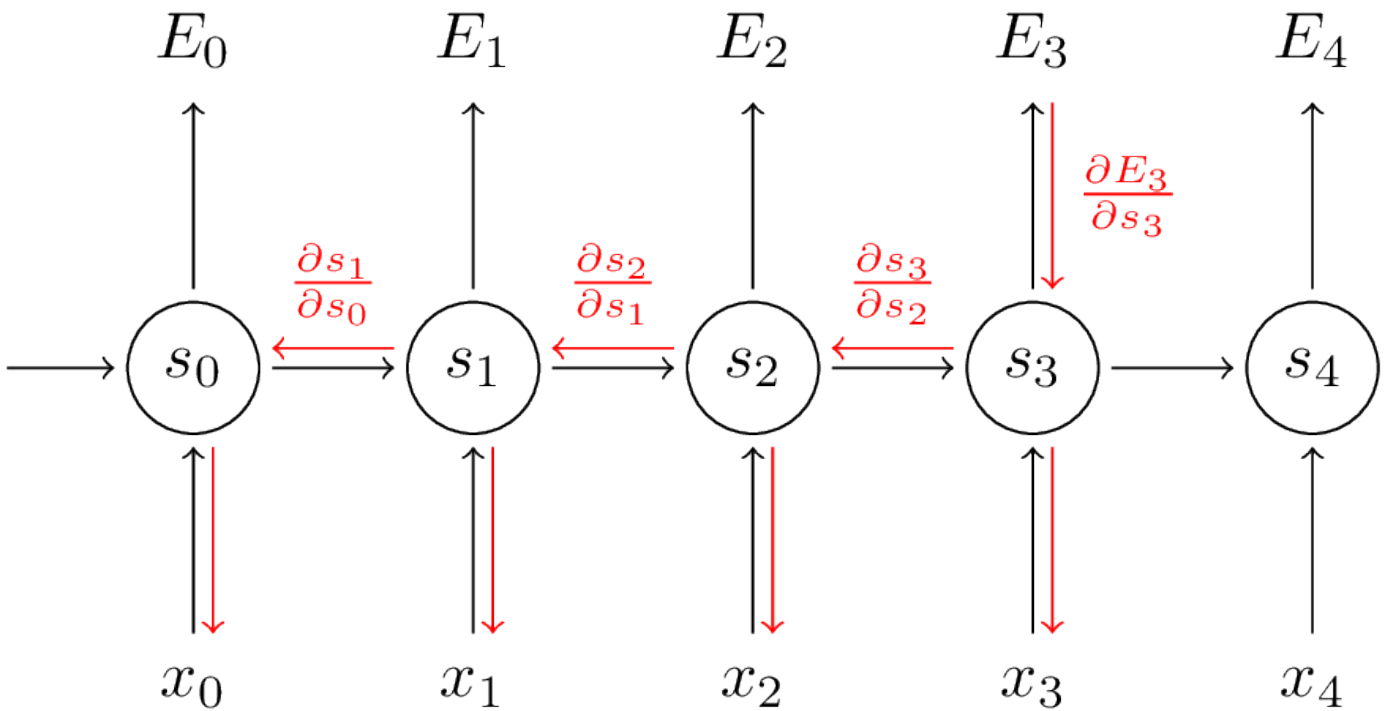




(left) http://colah.github.io/posts/2015-08-Understanding-LSTMs/
(right) http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Backpropagation through time

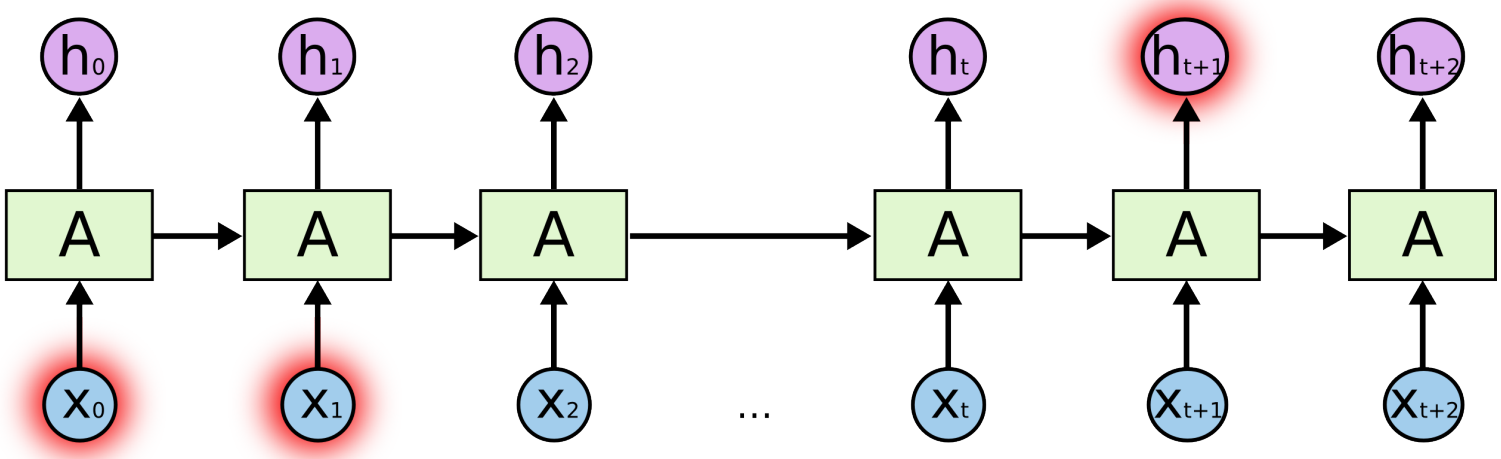
- Very similar to standard backpropagation in feedforward NNs.
- Key difference: Sum gradients for W at each time step.



http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/

Vanishing gradients & long-term dependencies

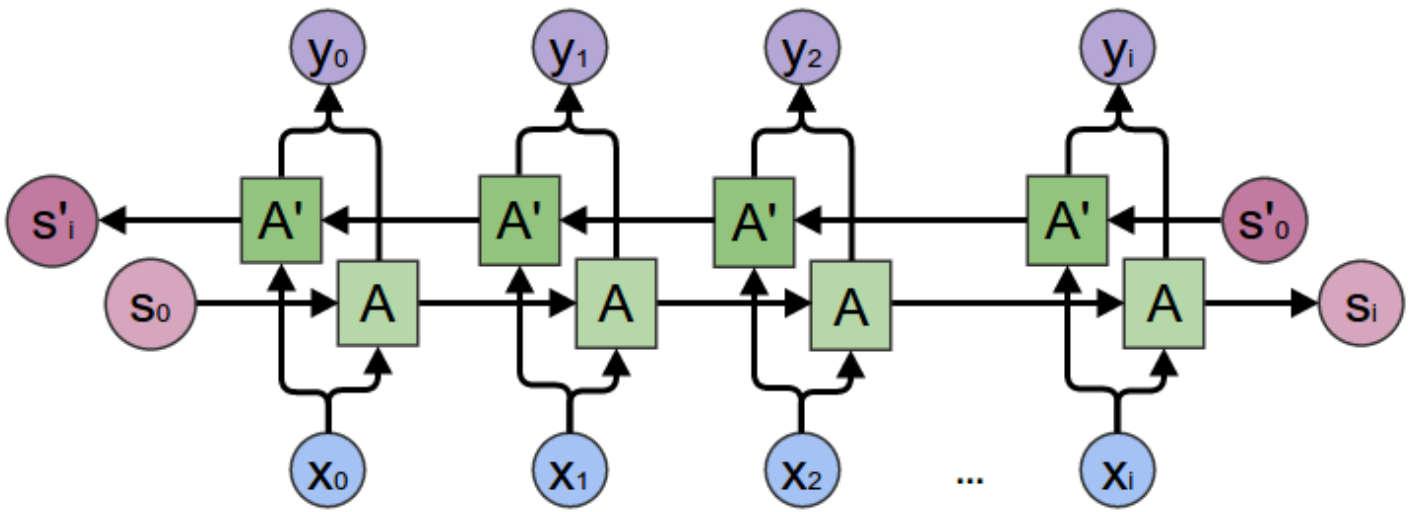
"The **cat** just had plenty of delicious food and **is** now full".
"The **cats** just had plenty of delicious food and **are** now full."



$\Rightarrow$ Gated Recurrent Units (GRU) and Long short-term memory (LSTM)

Vanishing gradients & long-term dependencies

He said: "**Teddy** Roosevelt was a great president"
He said: "**Teddy** bears are on sale!"



http://colah.github.io/posts/2015-09-NN-Types-FP/