Deep Learning

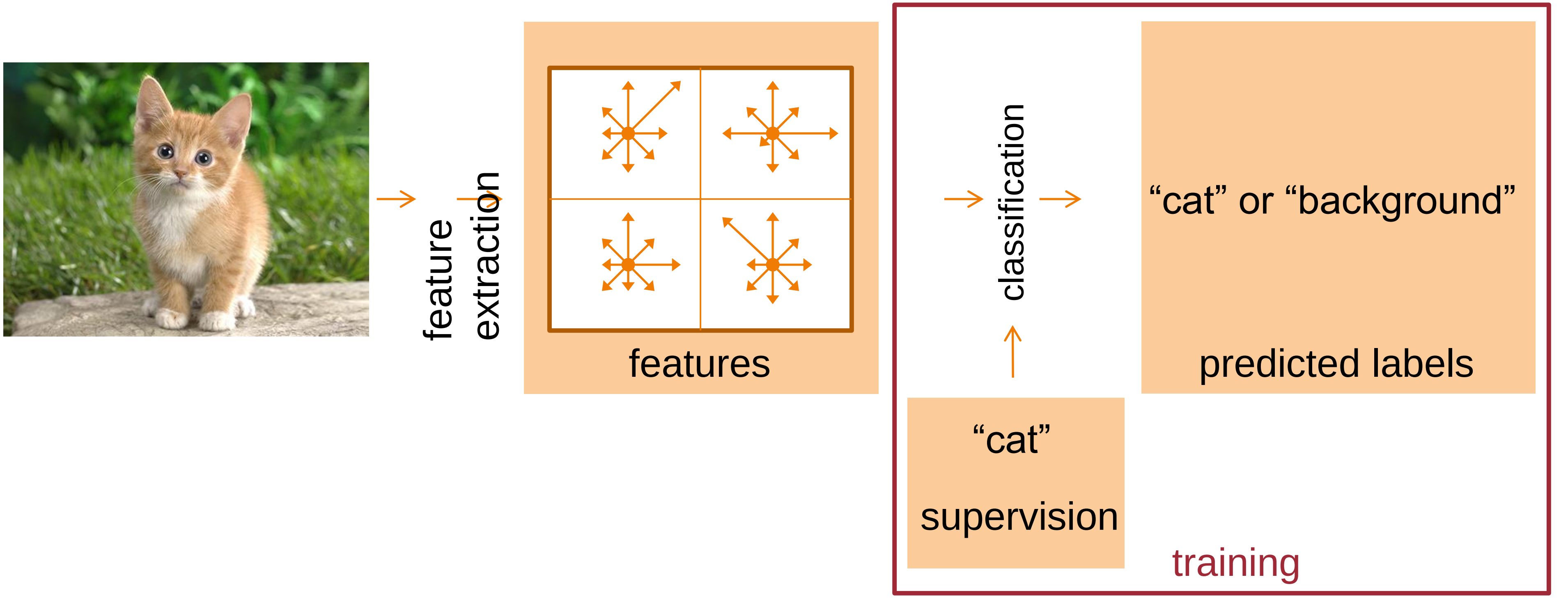
CMSC 422

MARINE CARPUAT

marine@cs.umd.edu

Based on slides by Vlad Morariu

Standard Application of Machine Learning to Computer Vision



- Features: e.g., Scale Invariant Feature Transform(SIFT)
- Classifiers: SVM, Random Forests, KNN, …
- Features are hand-crafted, not trained
  - eventually limited by feature quality

Cat image credit: https://raw.githubusercontent.com/BVLC/caffe/master/examples/images/cat.jpg

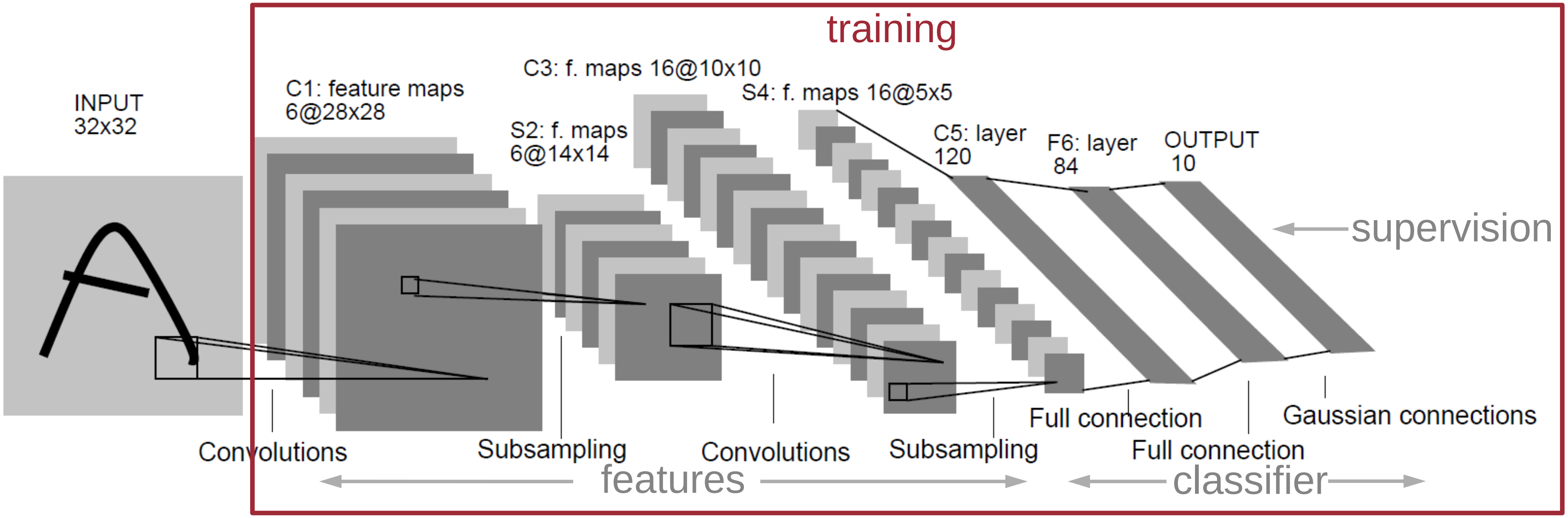
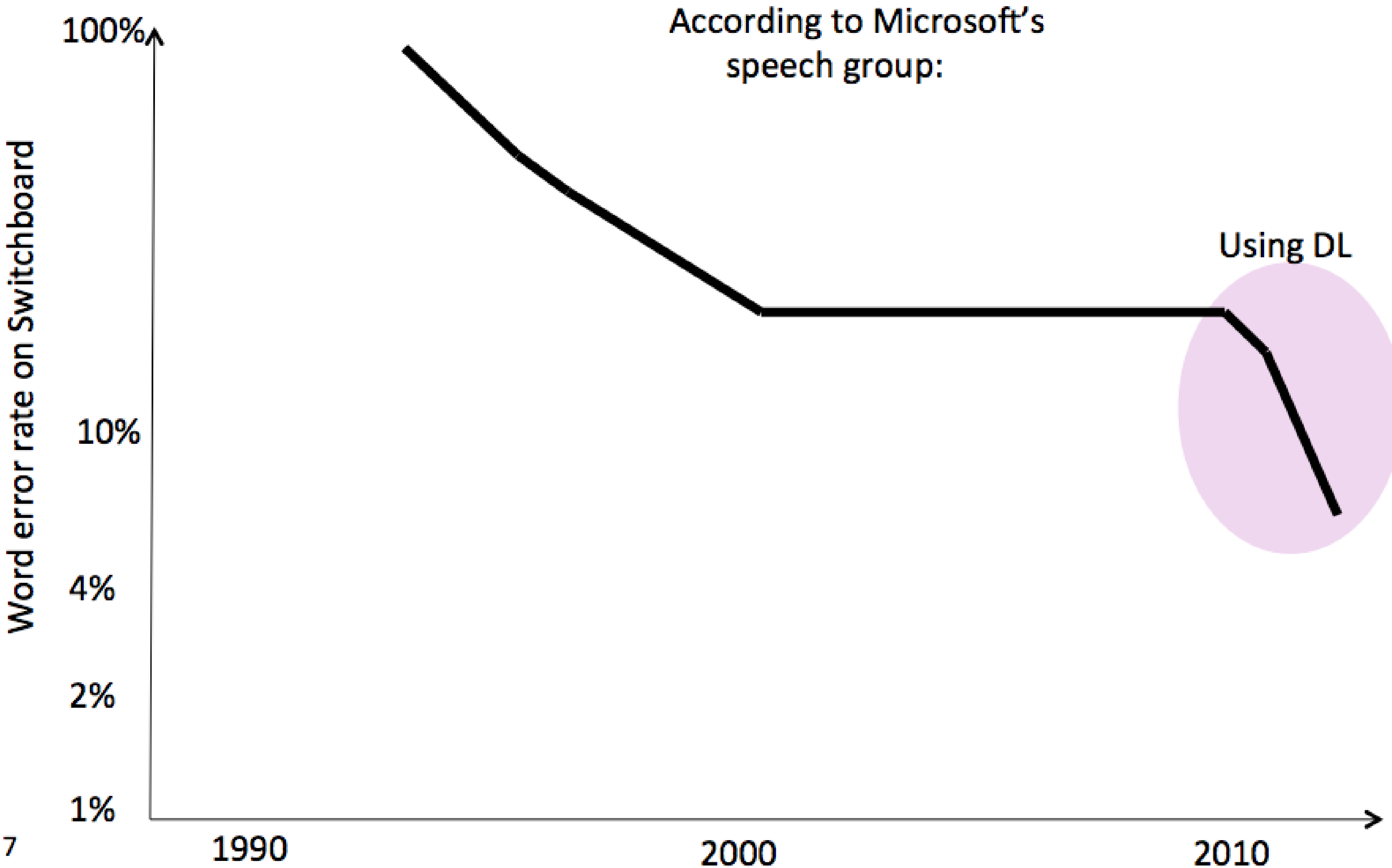


Image credit: LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. “Gradient-based learning applied to document recognition.” Proceedings of the IEEE, 1998.

- Deep learning
  - multiple layer neural networks
  - learn features and classifiers directly (“end-to-end” training)
  - breakthrough in Computer Vision, now in other AI areas

Speech Recognition



Slide credit: Bohyung Han

Image Classification Performance

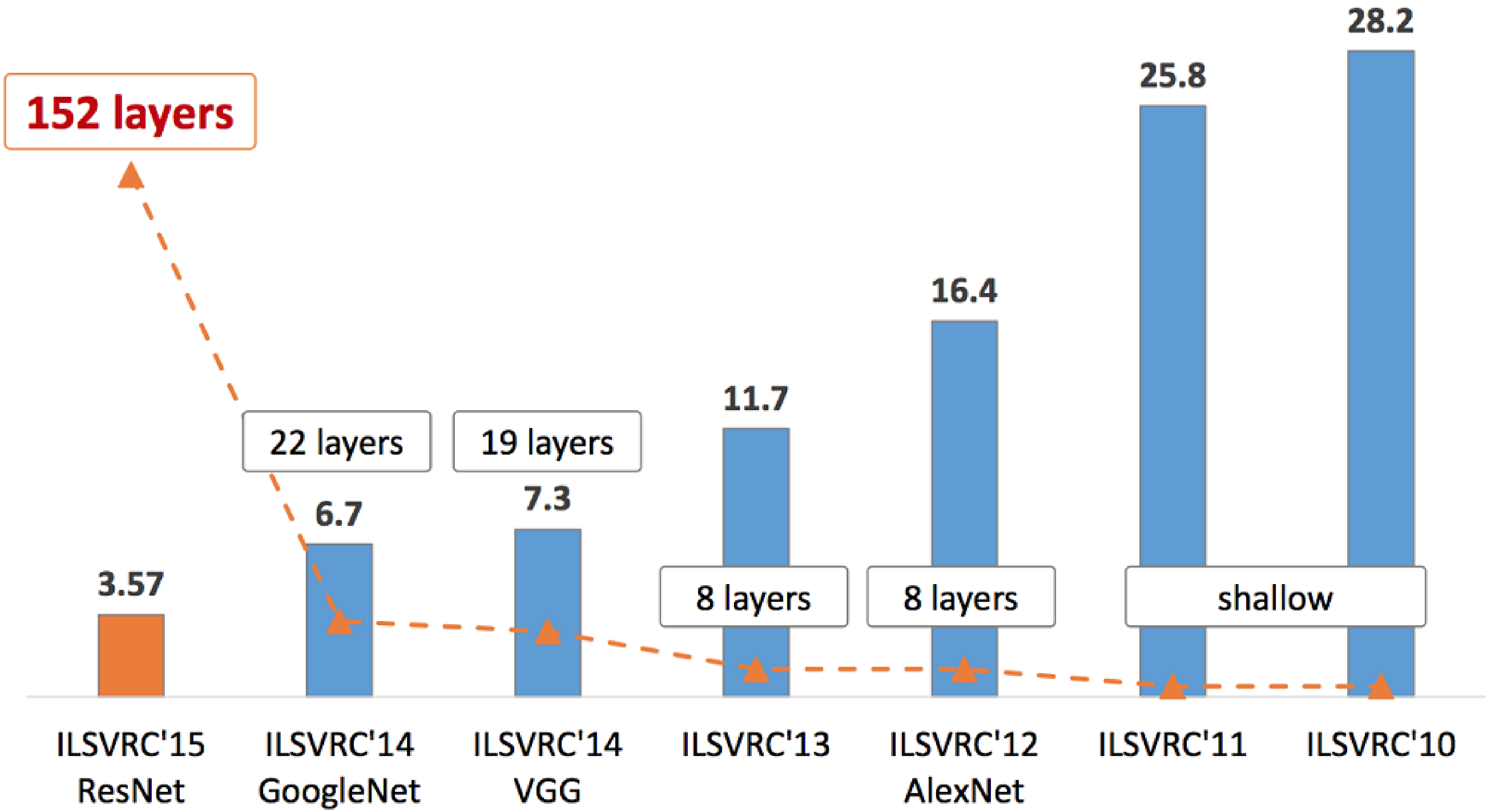


Image Classification Top-5 Errors (%)

Figure from: K. He, X. Zhang, S. Ren, J. Sun. "Deep Residual Learning for Image Recognition". arXiv 2015. (slides)

Slide credit: Bohyung Han

Today's lecture: key concepts

- Convolutional Neural Networks
- Revisiting Backpropagation and Gradient Descent for Deep Networks

Multi-Layer Perceptron (MLP)

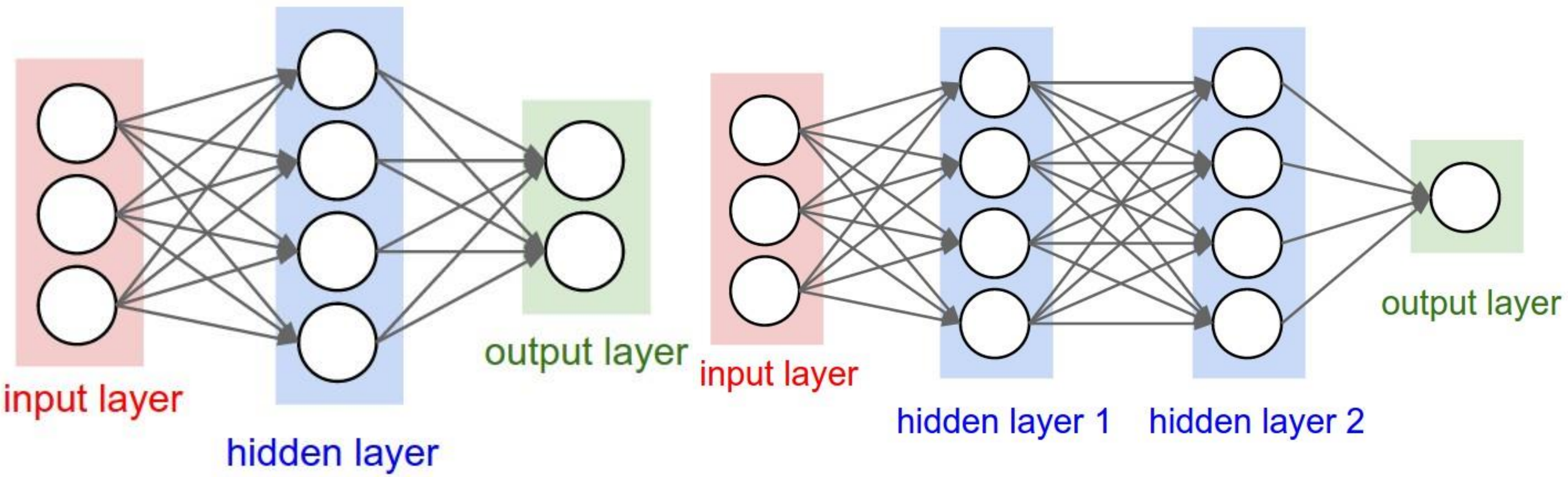


Image source: http://cs231n.github.io/neural-networks-1/

Neural Networks Applied to Vision

LeCun, Y; Boser, B; Denker, J; Henderson, D; Howard, R; Hubbard, W; Jackel, L, "Backpropagation Applied to Handwritten Zip Code Recognition," in Neural Computation, 1989

- USPS digit recognition, later check reading
- Convolution, pooling ("weight sharing"), fully connected layers

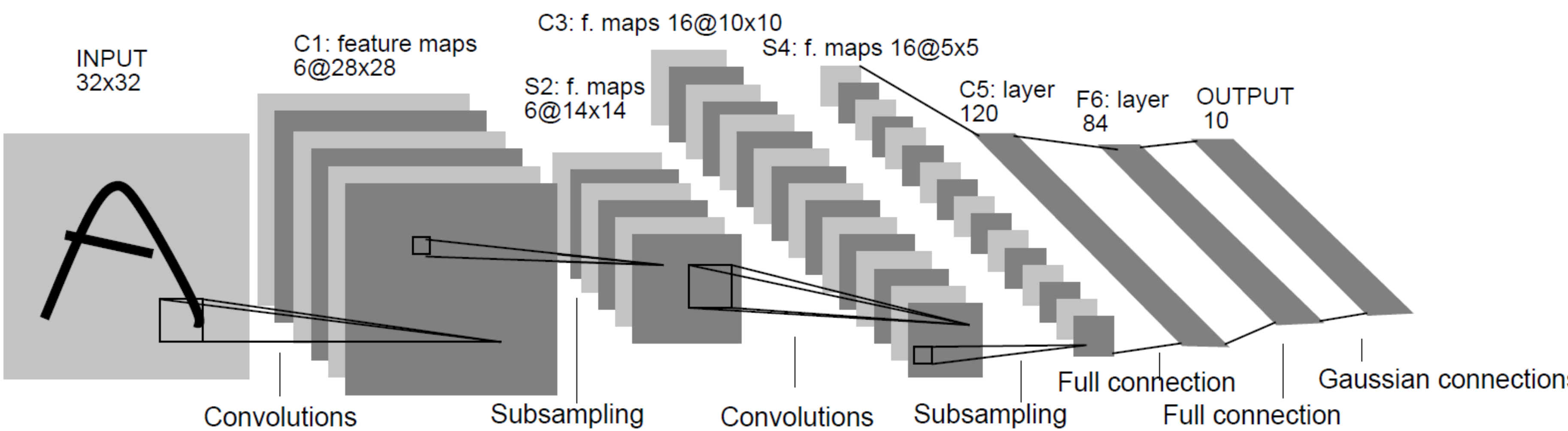


Image credit: LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 1998.

Architecture overview

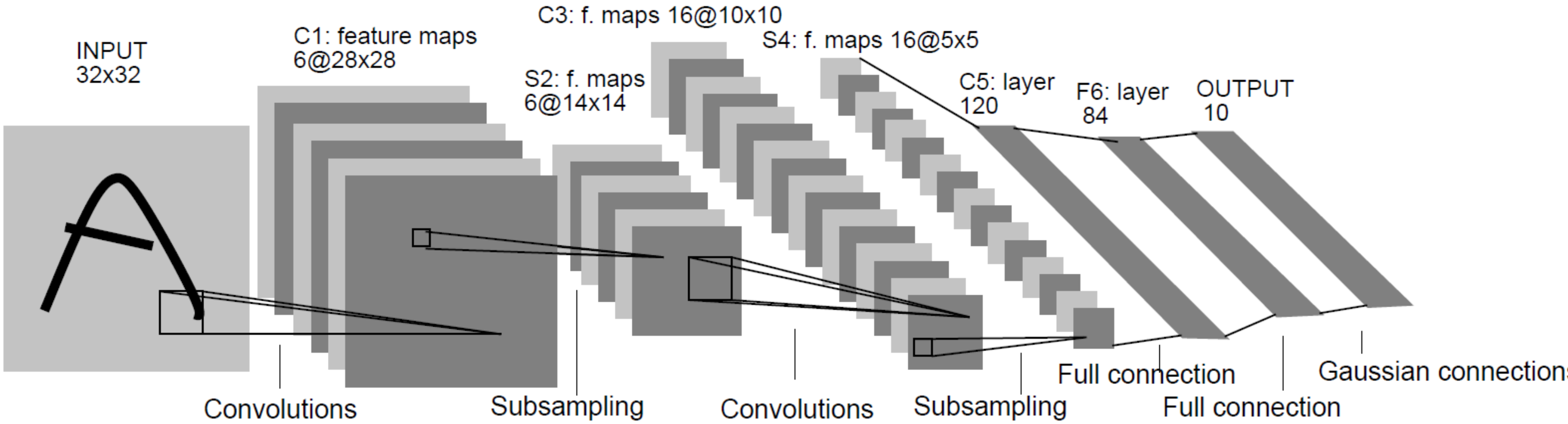


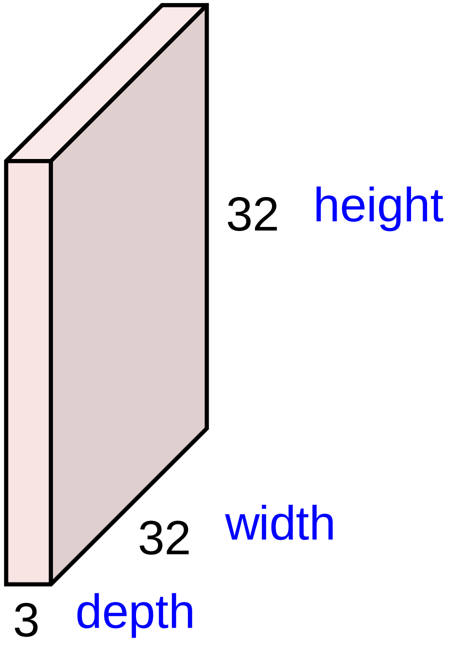
Image credit: LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. “Gradient-based learning applied to document recognition.” Proceedings of the IEEE, 1998.

Components:

- Convolution layers
- Pooling/Subsampling layers
- Fully connected layers

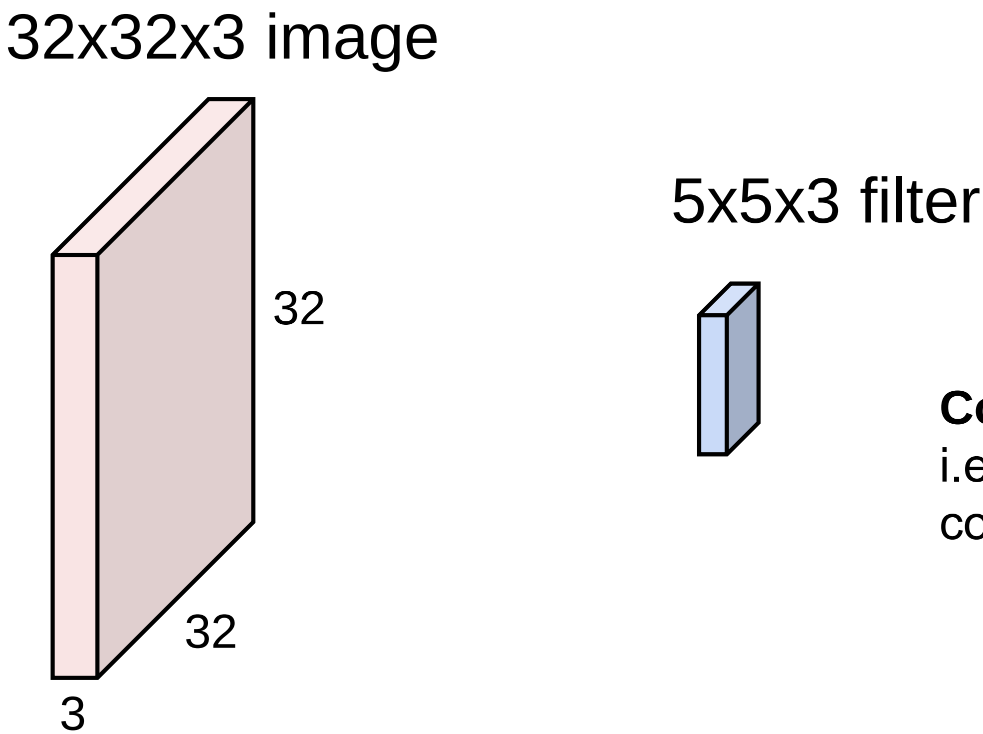
Convolutional Layer

32x32x3 image



Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson

Convolutional Layer



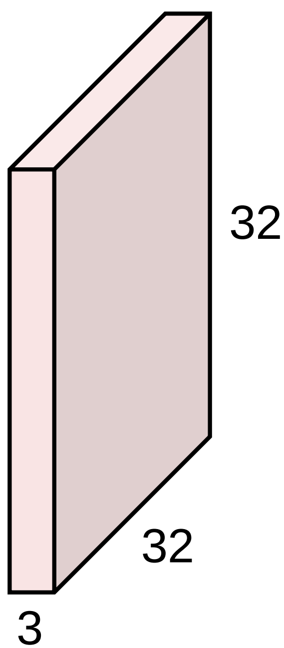
Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson

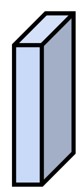
Convolutional Layer

Filters always extend the full depth of the input volume

32x32x3 image

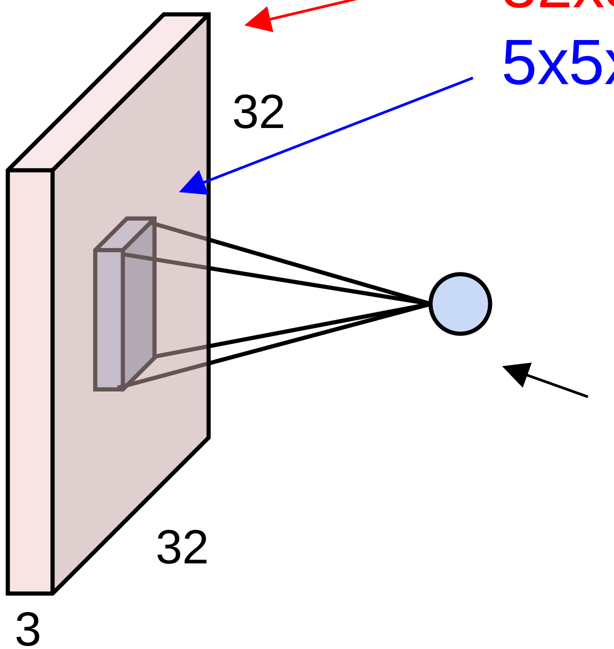


5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolutional Layer

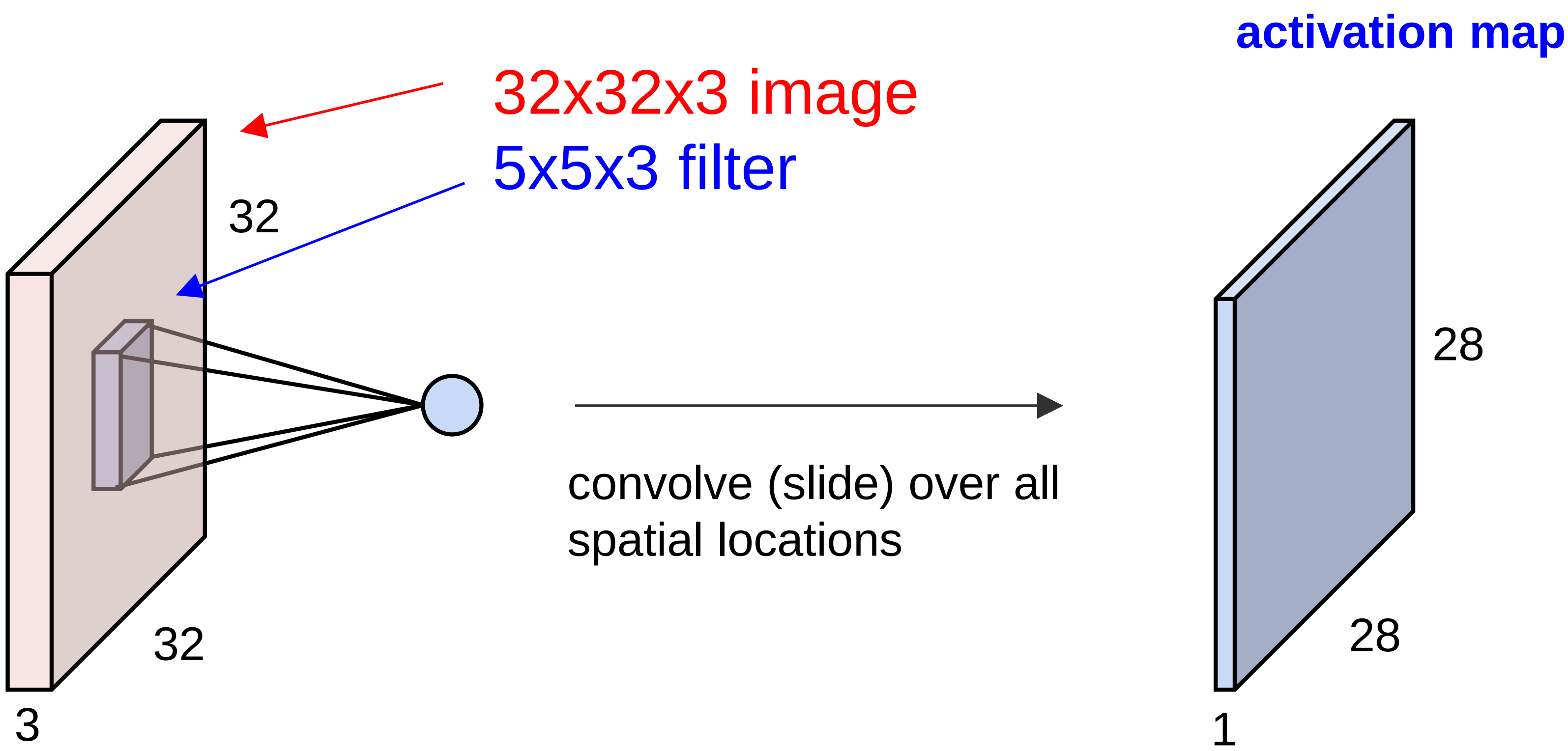


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson

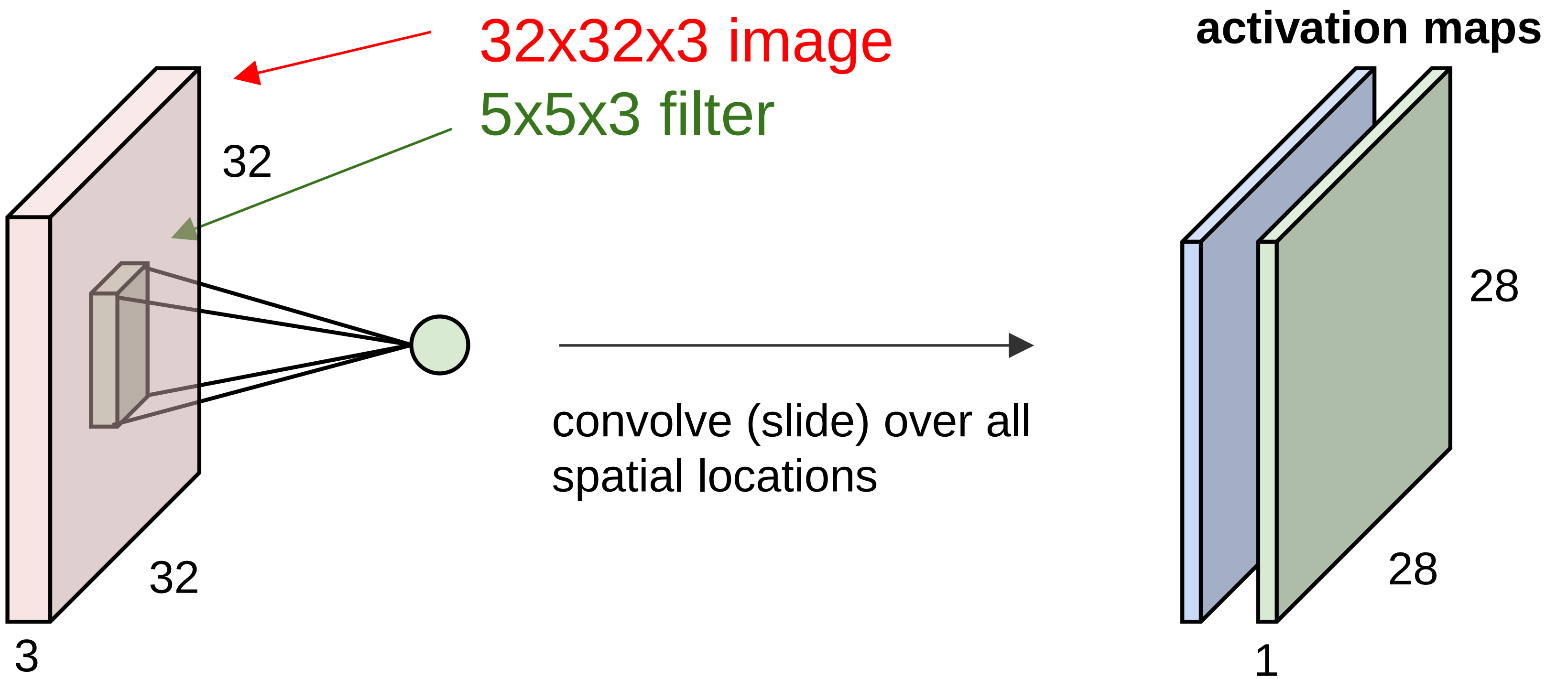
Convolutional Layer



Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Joh

Convolutional Layer

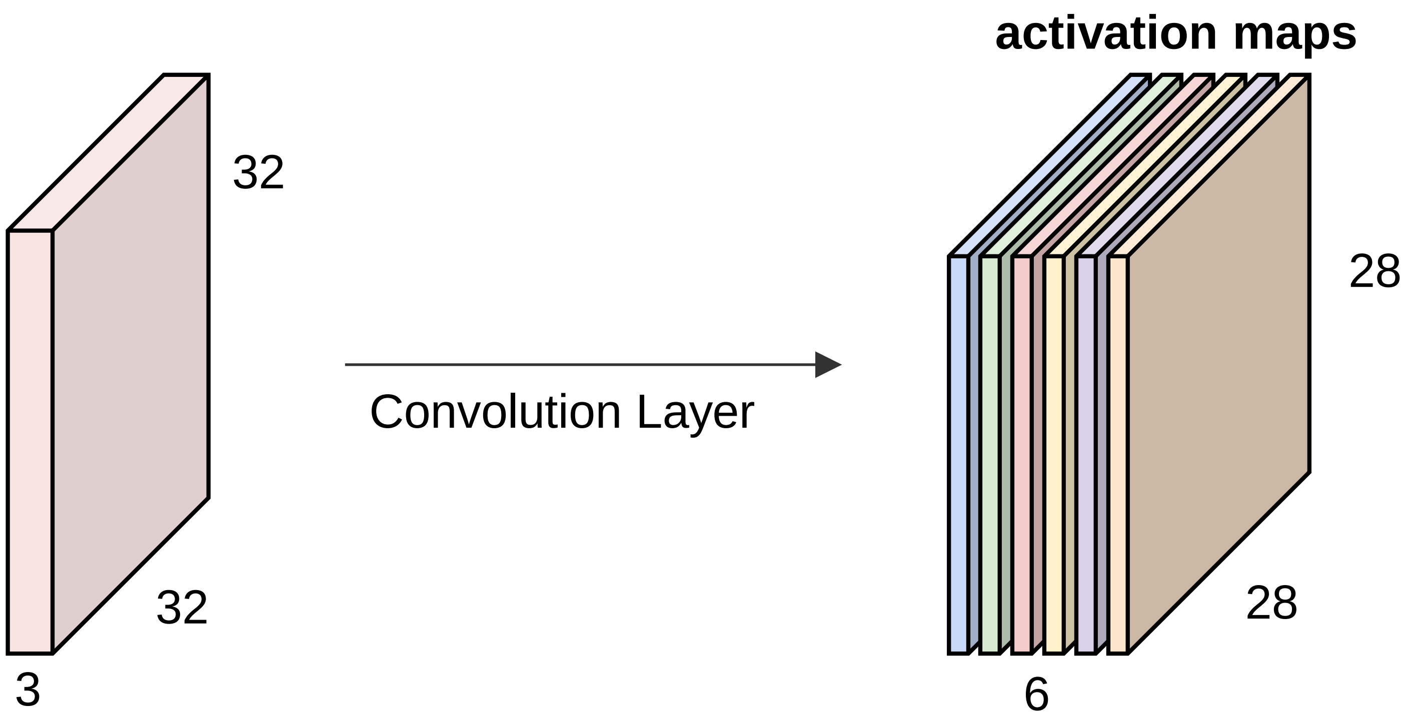
consider a second, green filter



Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson

Convolutional Layer

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

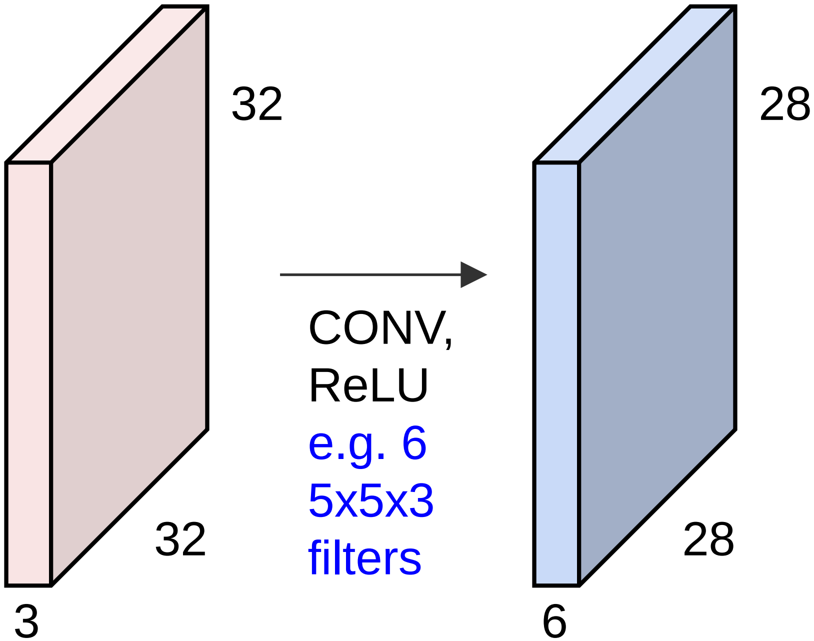


We stack these up to get a "new image" of size 28x28x6!

Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson

Convolutional Layer

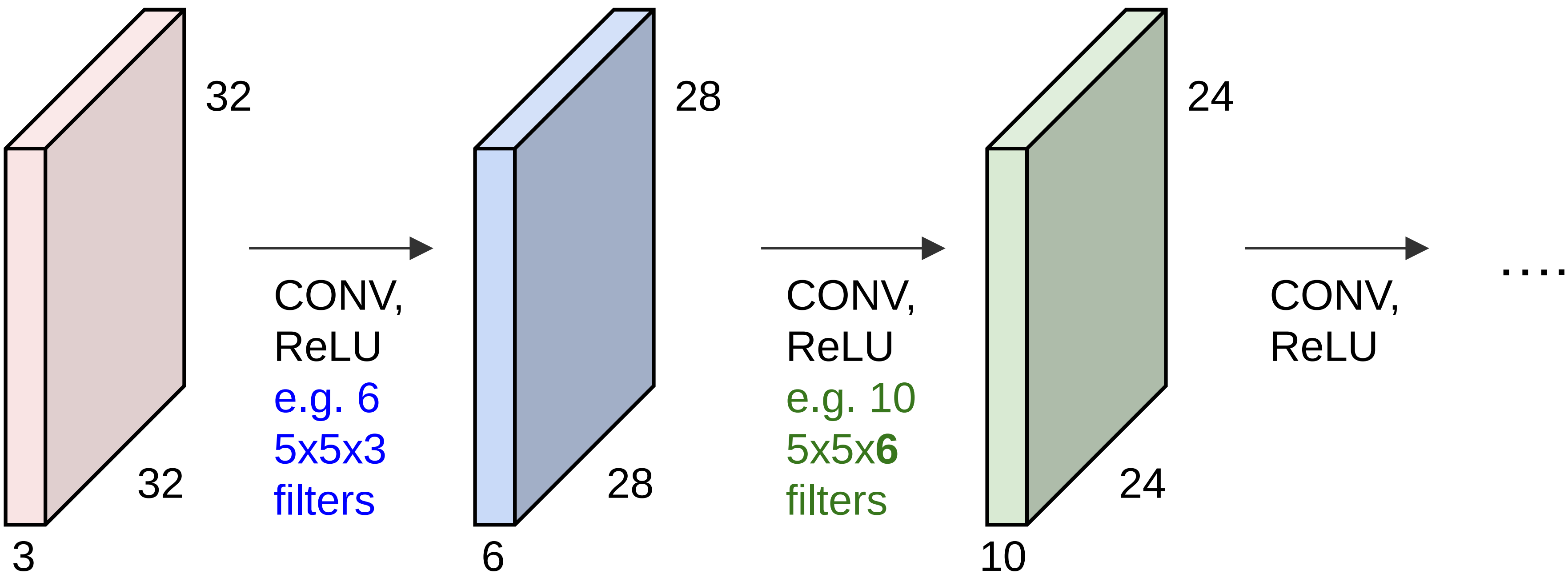
ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson

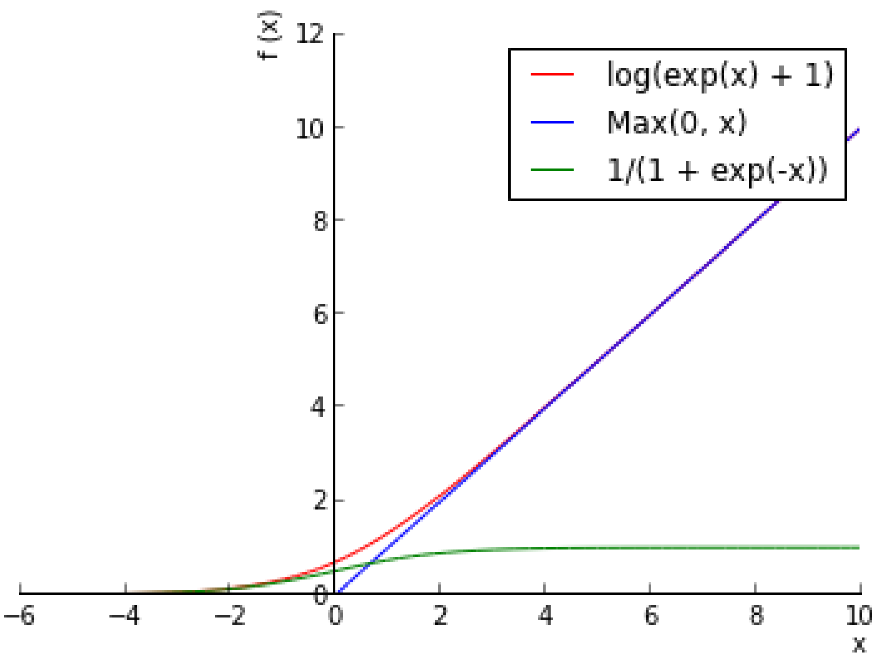
Convolutional Layer

ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson

Rectified Linear Units (ReLU)



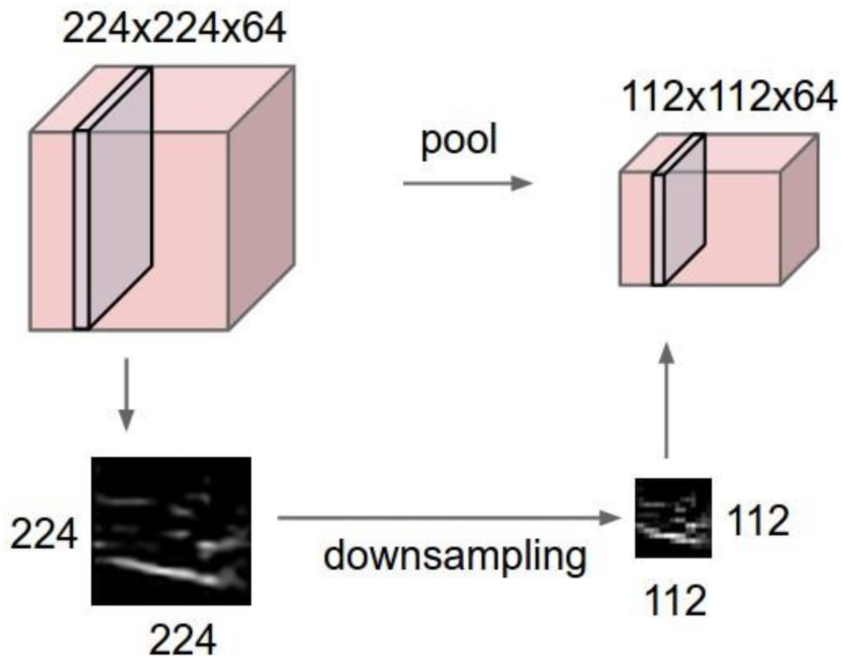
- Use rectified linear function instead of sigmoid

  ReL(x) = max (0,x)

- Advantages
  - Fast
  - No vanishing gradients

Pooling Layer

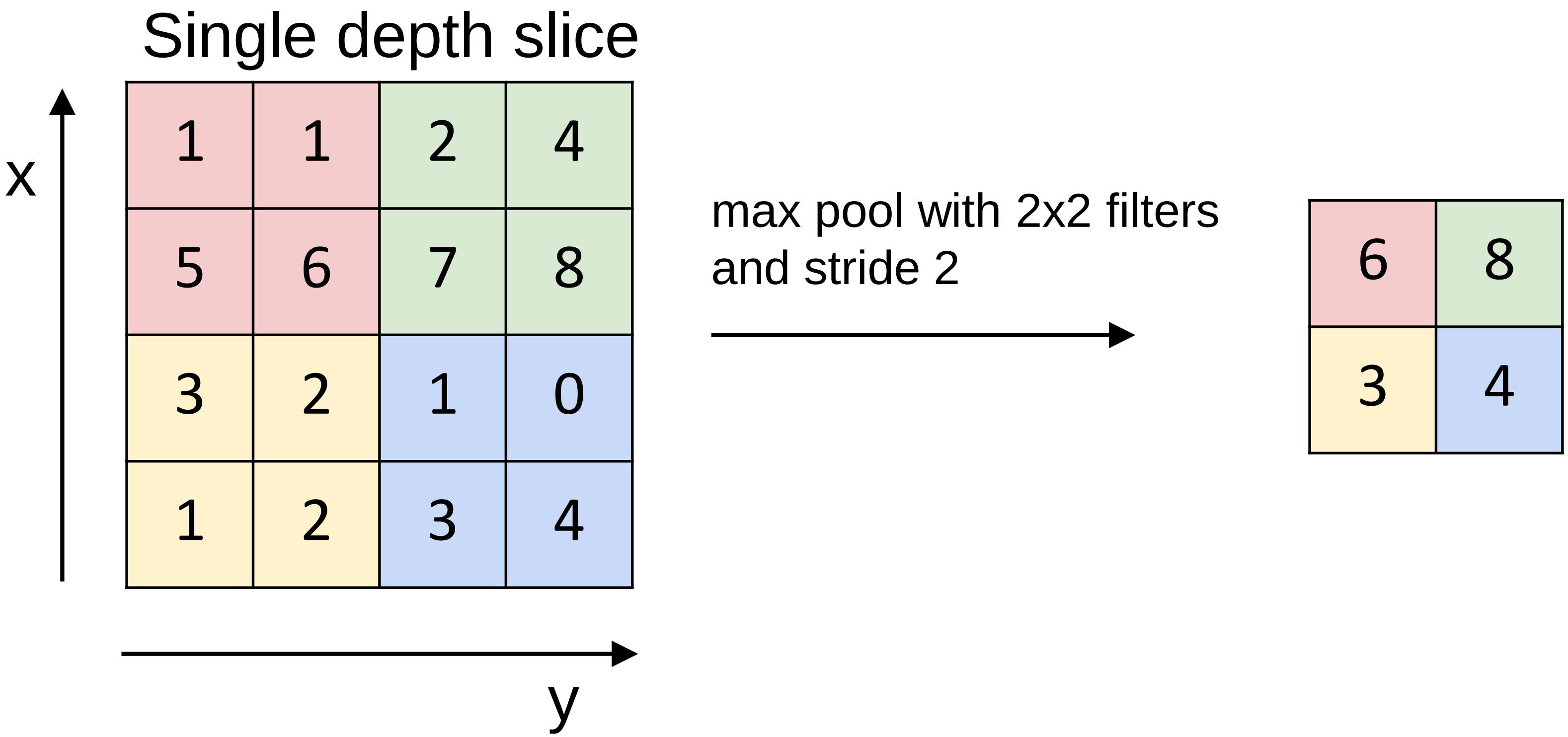
- makes the representations smaller and more manageable
- operates over each activation map independently



Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson

Pooling Layer

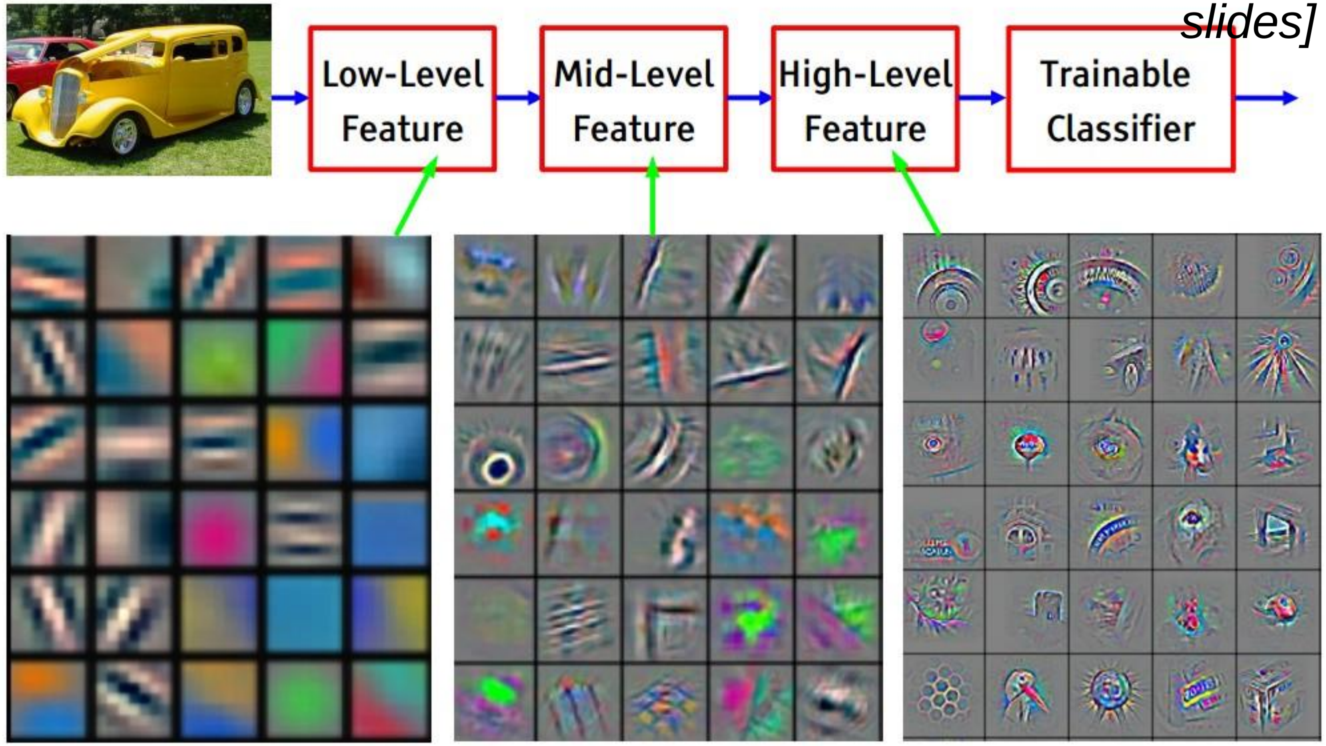
MAX POOLING



Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson

Convolutional filter visualization

[From recent Yann LeCun slides]

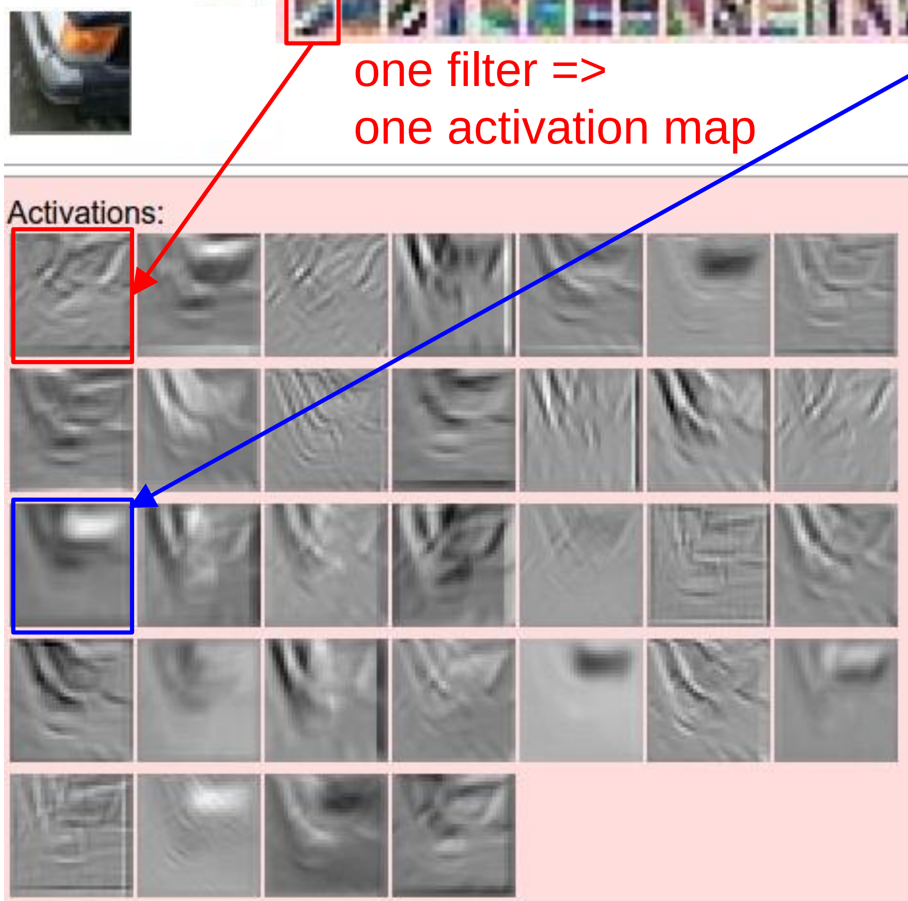


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson

Convolutional filter visualization

example 5x5 filters
(32 total)



We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

Slide credit: Fei-Fei Li, Andrej Karpathy, and Justin Johnson

Today’s lecture: key concepts

- Convolutional Neural Networks

- Revisiting Backpropagation and Gradient Descent for Deep Networks

Multi-Layer Perceptron (MLP)

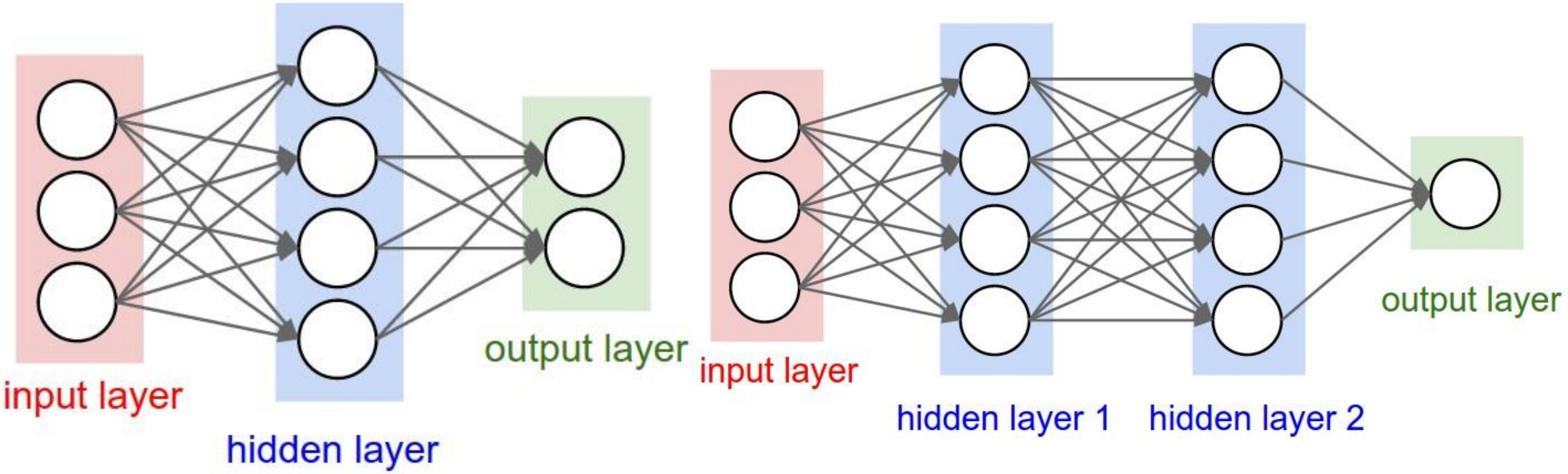
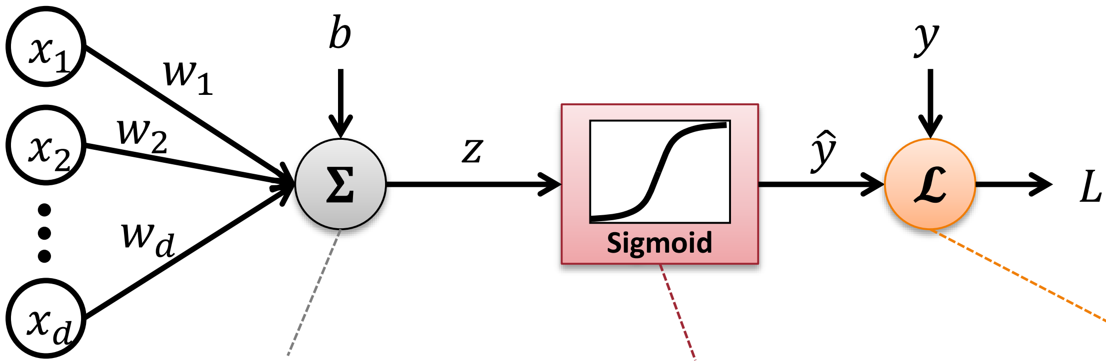


Image source: http://cs231n.github.io/neural-networks-1/

Single neuron gradient



$$z = b + \sum_i w_i x_i \qquad \hat{y} = \frac{1}{1+e^{-z}} \qquad L = \frac{1}{2}\sum_n (y^n - \hat{y}^n)^2$$

$$\frac{\partial z}{\partial w_i} = x_i \qquad \frac{d\hat{y}}{dz} = \hat{y}(1-\hat{y}) \qquad \frac{\partial L}{\partial \hat{y}^n} = -(y^n - \hat{y}^n)$$

Chain rule: If $y = f(x),\ z = g(y),$ then $\dfrac{dz}{dx} = \dfrac{dy}{dx}\dfrac{dz}{dy}$

$$\frac{\partial L}{\partial w_i} = \sum_n \frac{\partial \hat{y}^n}{\partial w_i}\frac{\partial L}{\partial \hat{y}^n} = \sum_n \frac{\partial z^n}{\partial w_i}\frac{d\hat{y}^n}{dz^n}\frac{\partial L}{\partial \hat{y}^n} = -\sum_n x_i^n \hat{y}^n(1-\hat{y}^n)(y^n - \hat{y}^n)$$

Slide credit: Adapted from Bohyung Han

Single neuron training

for $t = 1, \dots, T$

$\hat{y}^n = f(\boldsymbol{x}^n, \boldsymbol{w}_t) \quad (n = 1, \dots, N)$

$\dfrac{\partial L}{\partial w_i} = -\sum_n x_i^n \hat{y}^n (1 - \hat{y}^n)(y^n - \hat{y}^n) \quad (i = 1, \dots, d)$

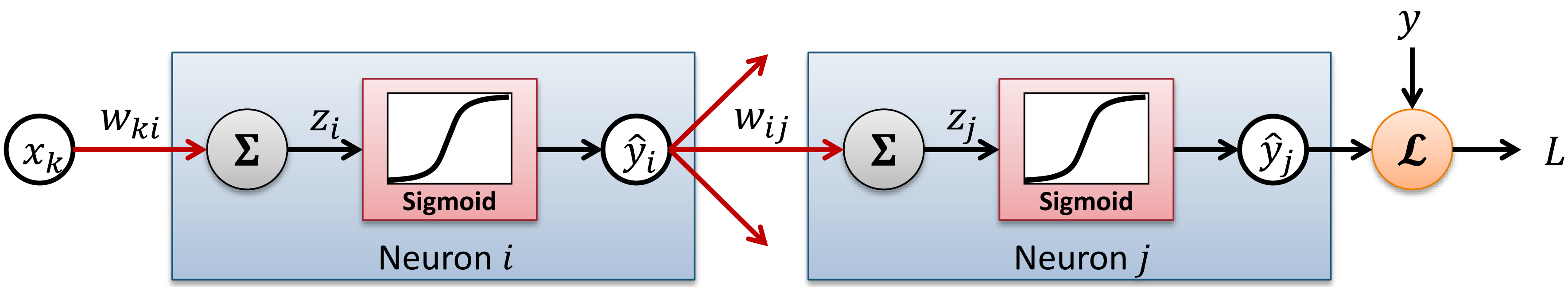
$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{\Delta w}$

endfor

an epoch

Slide credit: Adapted from Bohyung Han

Multi-Layer: Backpropagation



$$\frac{\partial L}{\partial z_j} = \frac{d\hat{y}_j}{dz_j}\frac{\partial L}{\partial \hat{y}_j}$$

$$\frac{\partial L}{\partial \hat{y}_i} = \sum_j \frac{dz_j}{d\hat{y}_i}\frac{\partial L}{\partial z_j} = \sum_j w_{ij}\frac{\partial L}{\partial z_j} = \sum_j w_{ij}\frac{d\hat{y}_j}{dz_j}\frac{\partial L}{\partial \hat{y}_j}$$

$$\frac{\partial L}{\partial w_{ki}} = \sum_n \frac{\partial z_i^n}{\partial w_{ki}}\frac{d\hat{y}_i^n}{dz_i^n}\frac{\partial L}{\partial \hat{y}_i^n} = \sum_n \frac{\partial z_i^n}{\partial w_{ki}}\frac{d\hat{y}_i^n}{dz_i^n}\sum_j w_{ij}\frac{d\hat{y}_j^n}{dz_j^n}\frac{\partial L}{\partial \hat{y}_j^n}$$

Slide credit: Bohyung Han

Backpropagation in practice

Two passes per iteration:

- **Forward pass:** compute value of loss function (and intermediate neurons) given inputs

- **Backward pass:** propagate gradient of loss (error) backwards through the network using the chain rule

Stochastic Gradient Descent (SGD)

- Update weights for each sample

$$E = \frac{1}{2}(y^n - \hat{y}^n)^2 \qquad \boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) - \epsilon \frac{\partial E^n}{\partial \boldsymbol{w}_i}$$

+ **Fast, online**
− **Sensitive to noise**

- Minibatch SGD: Update weights for a small set of samples

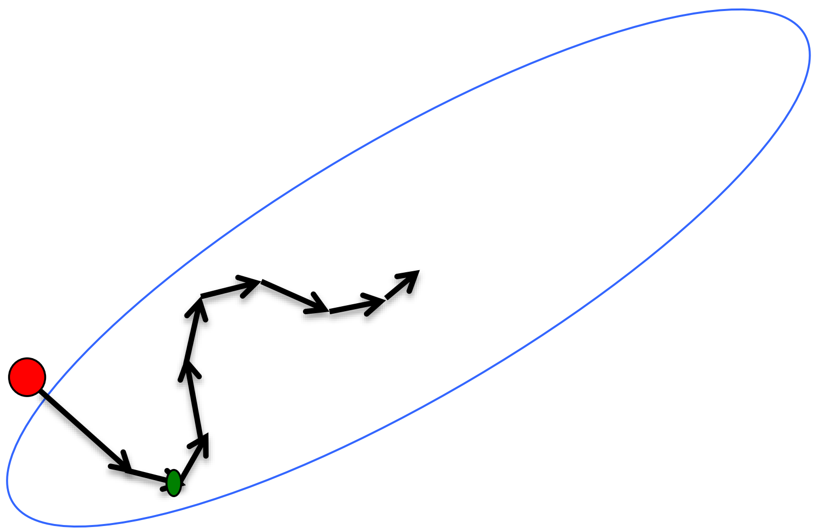
$$E = \frac{1}{2}\sum_{n \in B}(y^n - \hat{y}^n)^2 \qquad \boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) - \epsilon \frac{\partial E^B}{\partial \boldsymbol{w}_i}$$

+ **Fast, online**
+ **Robust to noise**

Slide credit: Bohyung Han

SGD improvements: Momentum

- Remember the previous direction



+ **Converge faster**
+ **Avoid oscillation**

$$v_i(t) = \alpha v_i(t-1) - \epsilon \frac{\partial E}{\partial w_i}(t)$$

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \boldsymbol{v}(t)$$

Slide credit: Bohyung Han

SGD improvements: Weight Decay

- Penalize the size of the weights

$$C = E + \frac{1}{2}\sum_i w_i^2$$

$$w_i(t+1) = w_i(t) - \epsilon\frac{\partial C}{\partial w_i} = w_i(t) - \epsilon\frac{\partial E}{\partial w_i} - \lambda w_i$$

+ Improve generalization a lot!

Slide credit: Bohyung Han

Key concepts

- Convolutional Neural Networks

- Revisiting Backpropagation and Gradient Descent for Deep Networks

History: NN Revival in the 1980's

Backpropagation discovered in 1970's but popularized in 1986

- David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams. "Learning representations by back-propagating errors." In Nature, 1986.

MLP is a universal approximator

- Can approximate any non-linear function in theory, given enough neurons, data
- Kurt Hornik, Maxwell Stinchcombe, Halbert White. "Multilayer feedforward networks are universal approximators." Neural Networks, 1989

Generated lots of excitement and applications

http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

Neural Networks Applied to Vision

LeNet – vision application

- LeCun, Y; Boser, B; Denker, J; Henderson, D; Howard, R; Hubbard, W; Jackel, L, "Backpropagation Applied to Handwritten Zip Code Recognition," in Neural Computation, 1989
- USPS digit recognition, later check reading
- Convolution, pooling ("weight sharing"), fully connected layers

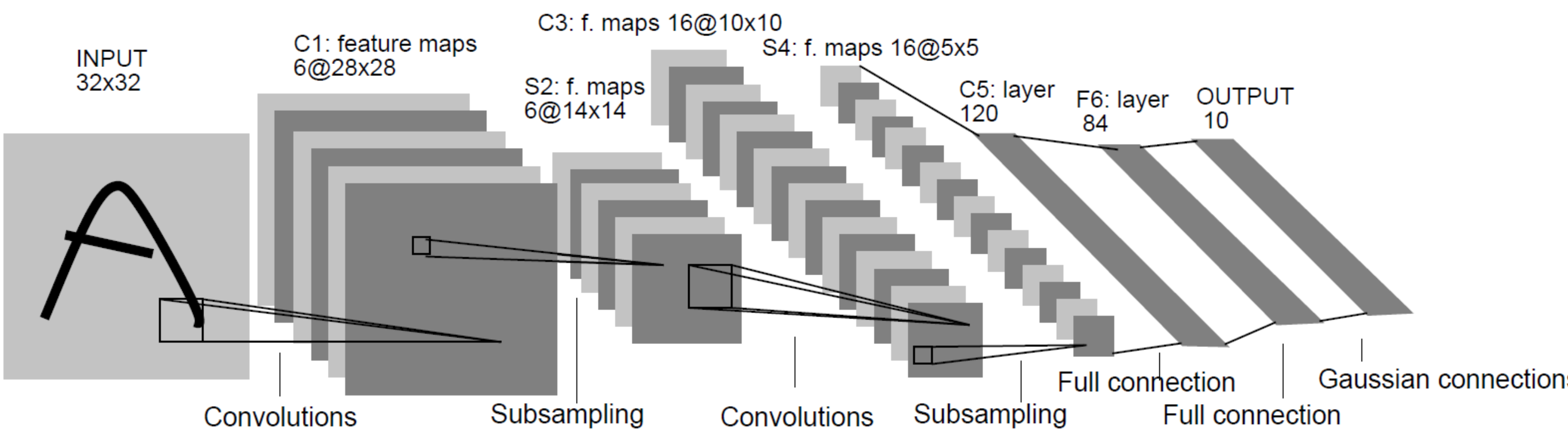


Image credit: LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 1998.

Issues in Deep Neural Networks

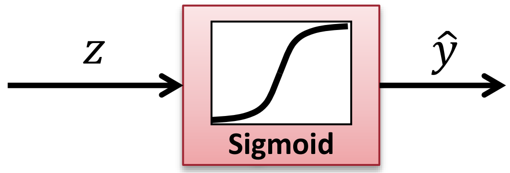
- Prohibitive training time
  - Especially with lots of training data
  - Many epochs typically required for optimization
  - Expensive gradient computations

- Overfitting
  - Learned function fits training data well, but performs poorly on new data (high capacity model, not enough training data)

Slide credit: adapted from Bohyung Han

Issues in Deep Neural Networks

Vanishing gradient problem



$$\frac{\partial E}{\partial w_{ki}} = \sum_n \frac{\partial z_i^n}{\partial w_{ki}} \frac{d\hat{y}_i^n}{dz_i^n} \frac{\partial E}{\partial \hat{y}_i^n} = \sum_n \frac{\partial z_i^n}{\partial w_{ki}} \boxed{\frac{d\hat{y}_i^n}{dz_i^n}} \sum_j w_{ij} \boxed{\frac{d\hat{y}_j^n}{dz_j^n}} \frac{\partial E}{\partial \hat{y}_j^n}$$

- Gradients in the lower layers are typically extremely small
- Optimizing multi-layer neural networks takes huge amount of time

Slide credit: adapted from Bohyung Han

New “winter” and revival in early 2000’s

New “winter” in the early 2000’s due to

- problems with training NNs
- Support Vector Machines (SVMs), Random Forests (RF) – easy to train, nice theory

Revival again by 2011-2012

- Name change (“neural networks” -> “deep learning”)
- + Algorithmic developments
  - unsupervised layer-wise pre-training
  - ReLU, dropout, layer normalizatoin
- + Big data + GPU computing =
- Large outperformance on many datasets (Vision: ILSVRC’12)

http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning-part-4/

Big Data

- ImageNet Large Scale Visual Recognition Challenge
  - 1000 categories w/ 1000 images per category
  - 1.2 million training images, 50,000 validation, 150,000 testing

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei. **ImageNet Large Scale Visual Recognition Challenge**. *IJCV,* 2015.

AlexNet Architecture

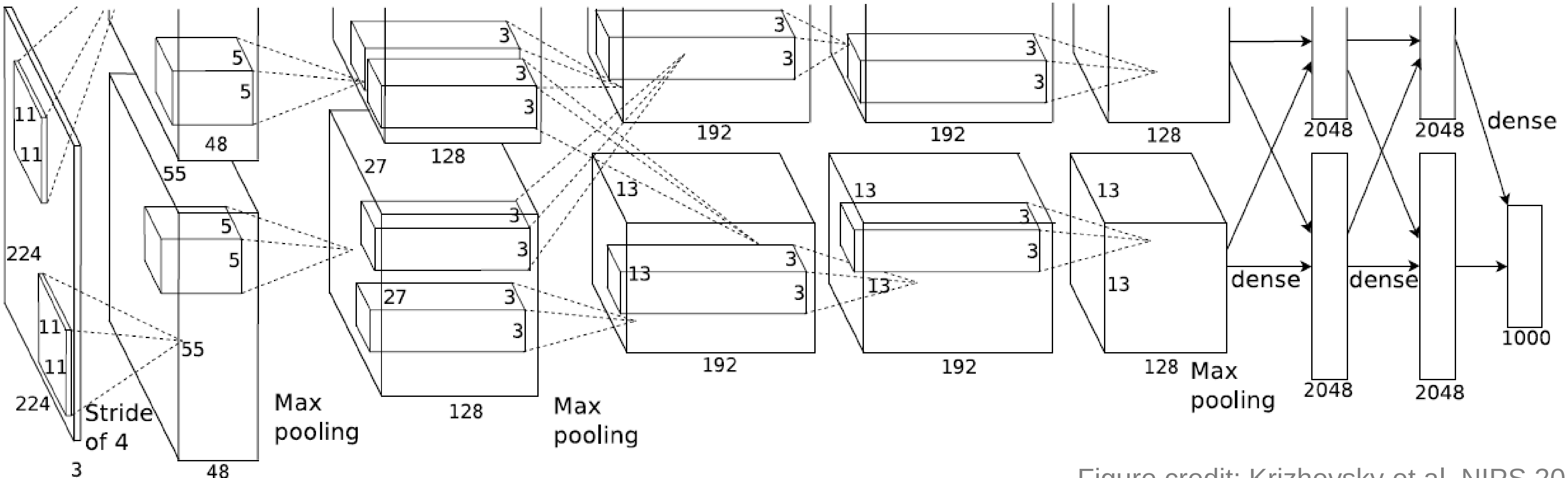


Figure credit: Krizhevsky et al, NIPS 2012.

60 million parameters!

Various tricks

- ReLU nonlinearity
- Overlapping pooling
- Local response normalization
- Dropout – set hidden neuron output to 0 with probability .5
- Data augmentation
- Training on GPUs

Alex Krizhevsky, Ilya Sutskeyer, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. NIPS, 2012.

GPU Computing

- Big data and big models require lots of computational power

- GPUs
  - thousands of cores for parallel operations
  - multiple GPUs
  - still took about 5-6 days to train AlexNet on two NVIDIA GTX 580 3GB GPUs (much faster today)

Recurrent Neural Networks

Networks with loops

- The output of a layer is used as input for the same (or lower) layer
- Can model dynamics (e.g. in space or time)

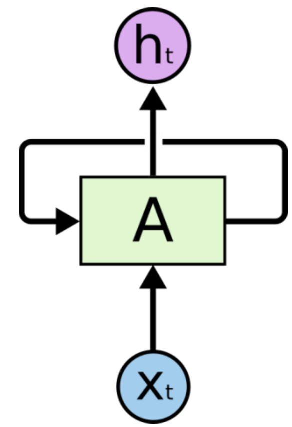
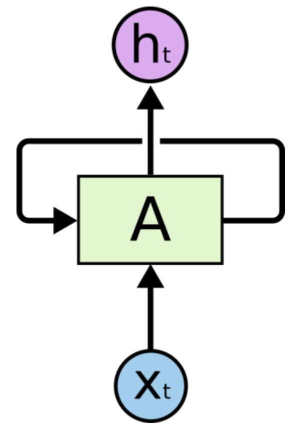


Image credit: Chritopher Olah's blog http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Sepp Hochreiter (1991), Untersuchungen zu dynamischen neuronalen Netzen, Diploma thesis. Institut f. Informatik, Technische Univ. Munich. Advisor: J. Schmidhuber.

Y. Bengio, P. Simard, P. Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. In TNN 1994.

Recurrent Neural Networks



Let's unroll the loops

- Now a standard feed-forward network with many layers
- Suffers from vanishing gradient problem
- In theory, can learn long term memory, in practice not (Bengio et al, 1994)

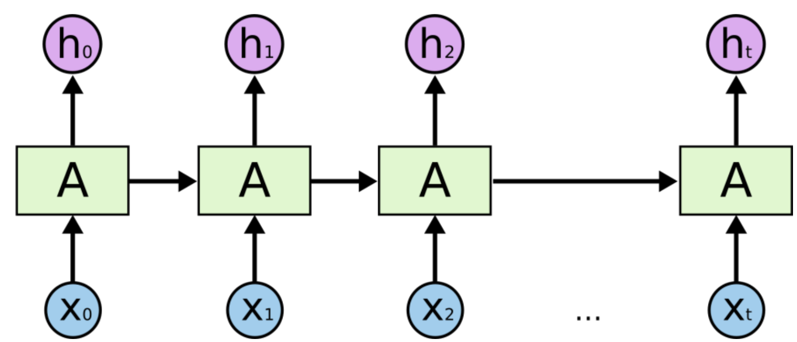
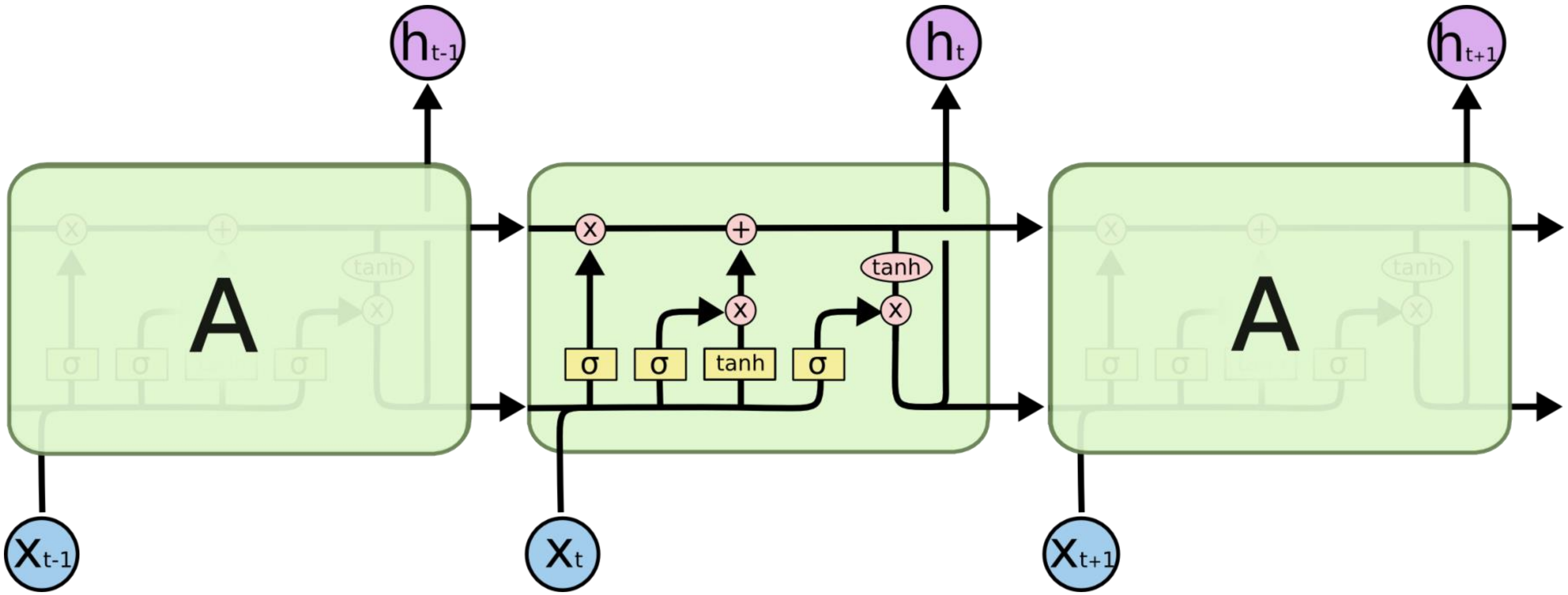


Image credit: Chritopher Olah's blog http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Sepp Hochreiter (1991), Untersuchungen zu dynamischen neuronalen Netzen, Diploma thesis. Institut f. Informatik, Technische Univ. Munich. Advisor: J. Schmidhuber.

Y. Bengio, P. Simard, P. Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. In TNN 1994.

Long Short Term Memory (LSTM)



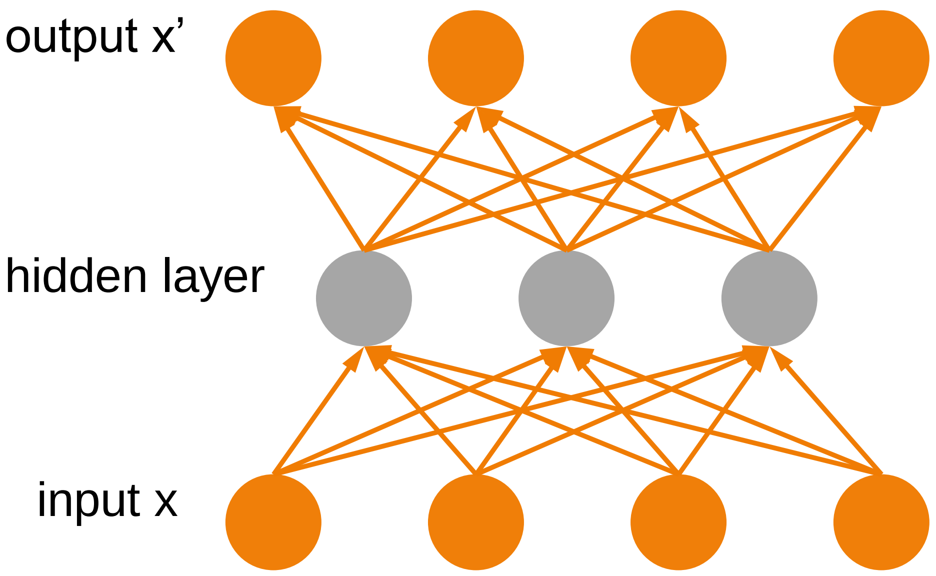
- A type of RNN explicitly designed not to have the vanishing or exploding gradient problem
- Models long-term dependencies
- Memory is propagated and accessed by gates
- Used for speech recognition, language modeling …

Hochreiter, Sepp; and Schmidhuber, Jürgen. “Long Short-Term Memory.” Neural Computation, 1997.
Image credit: Christopher Colah’s blog, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Unsupervised Neural Networks

Autoencoders

- Encode then decode the same input
- No supervision needed



H. Bourlard and Y. Kamp. 1988. Auto-association by multilayer perceptrons and singular value decomposition. Biol. Cybern. 59, 4-5 (September 1988), 291-294.