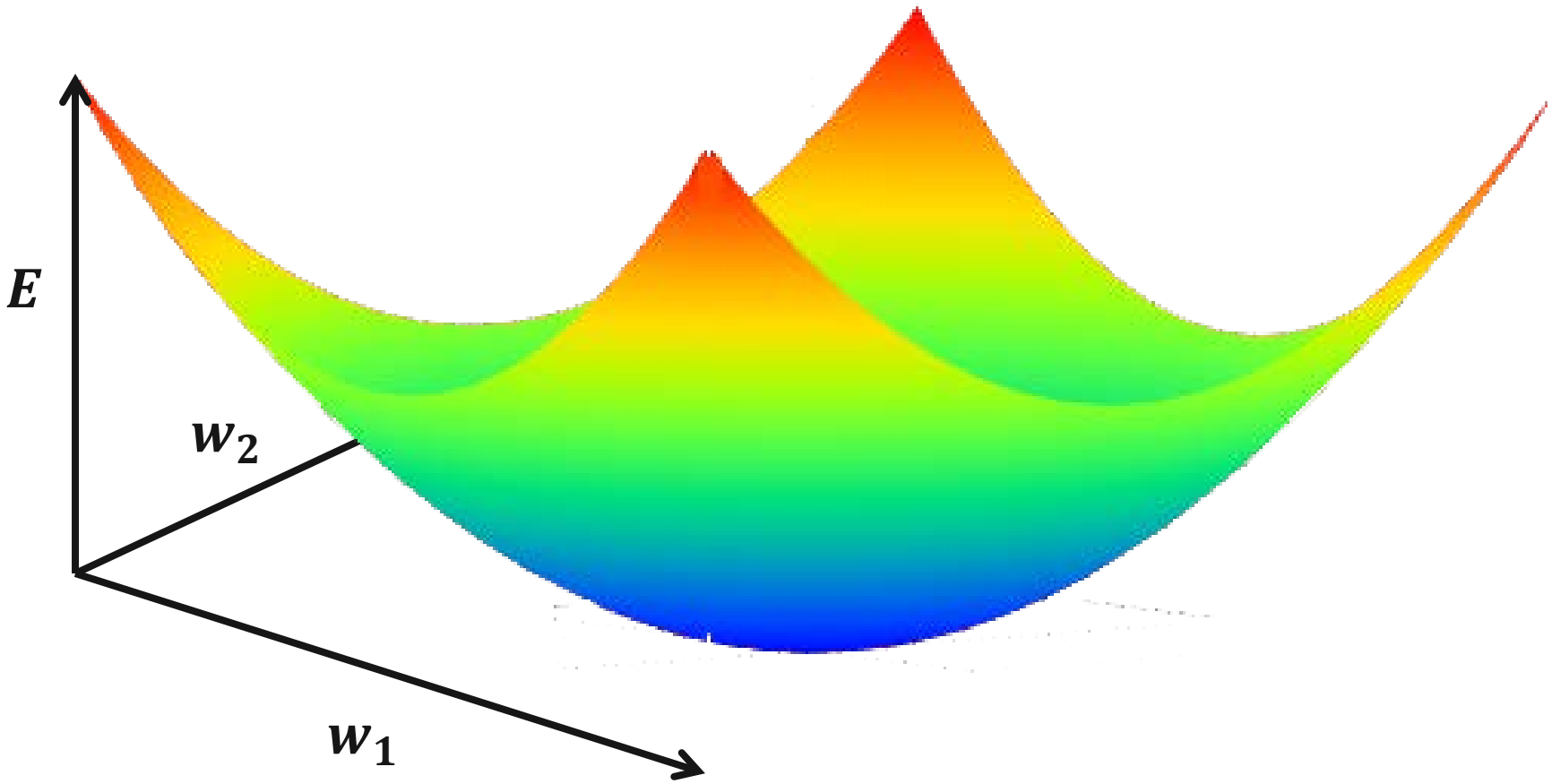
CSCE 689- Deep Learning for Computer Graphics

Basics of deep learning II

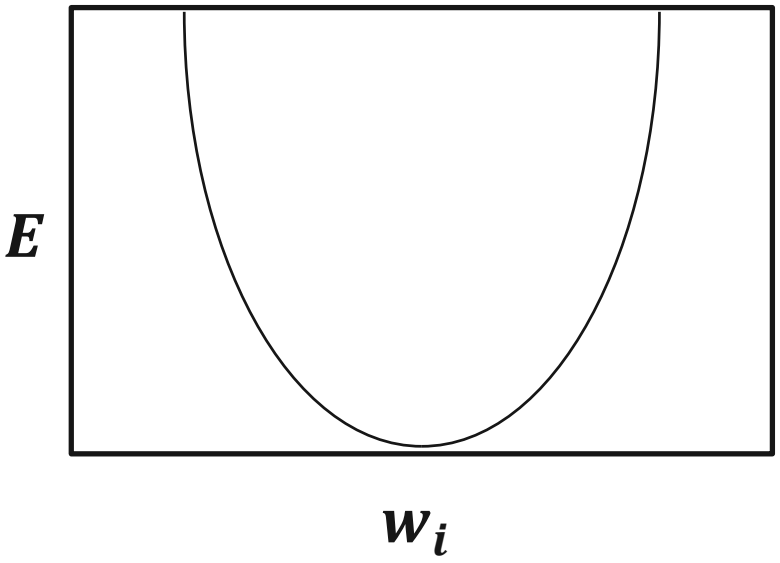
Nima Kalantari

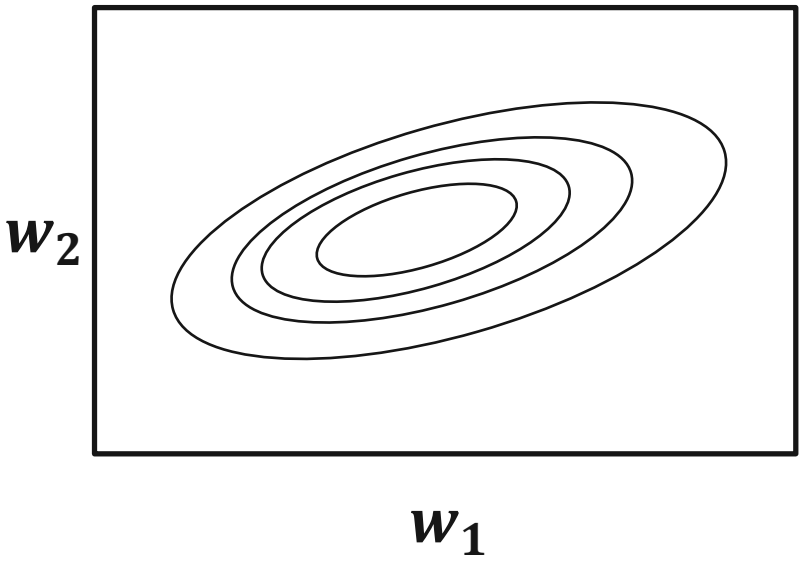
Recap

Error surface



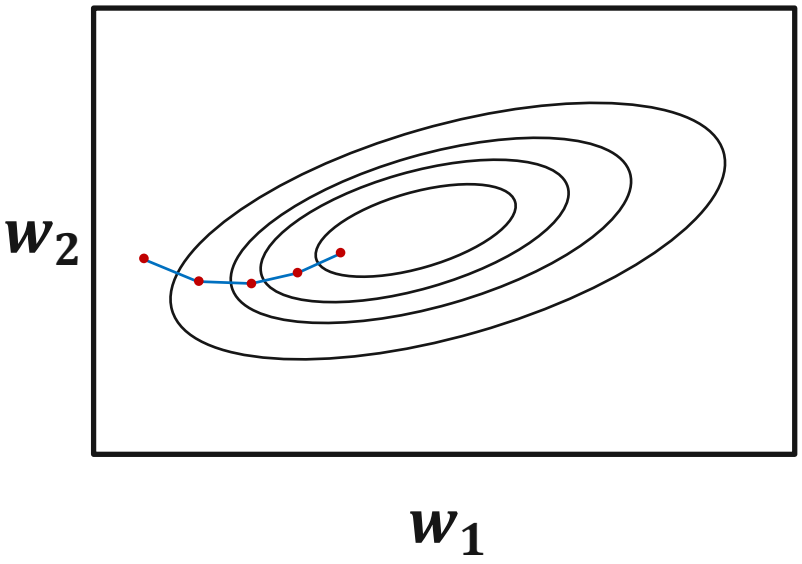
Error surface



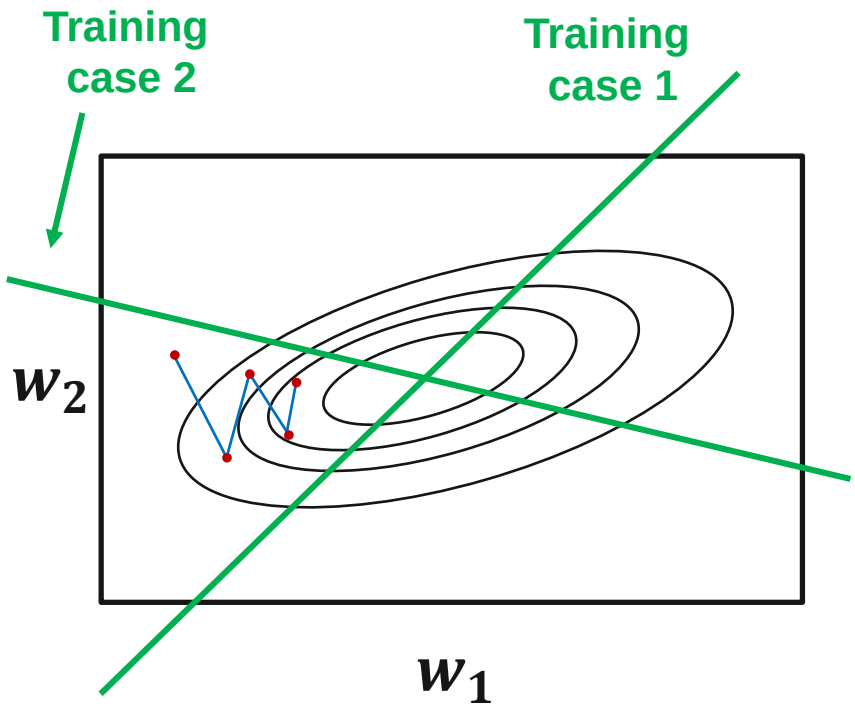


Online vs. batch learning

- Batch learning does steepest descent on the error surface
  - perpendicular to the contour lines



- Online learning zig zags around the direction of steepest descent

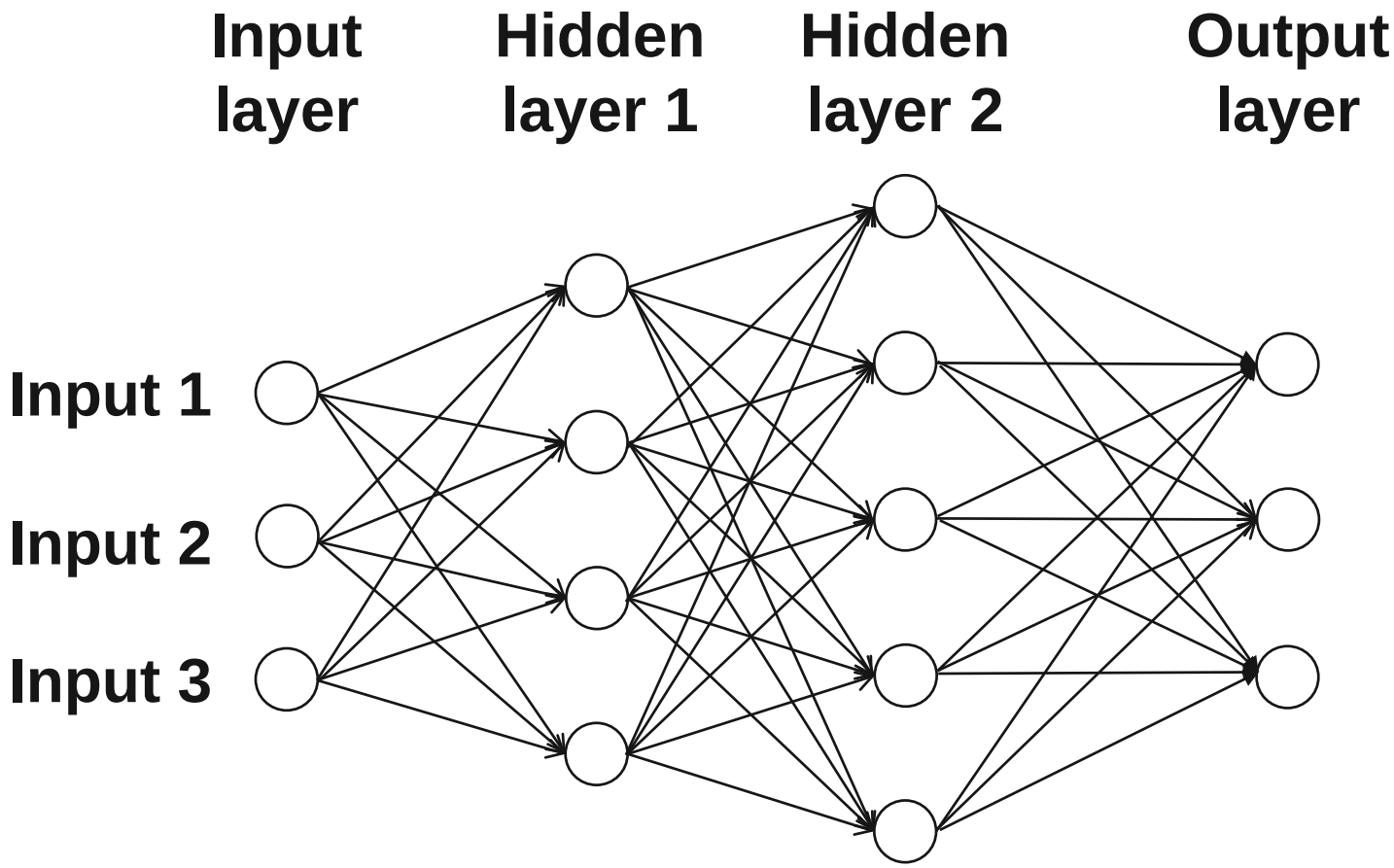


Mini-batch learning

- For highly redundant datasets, the gradient on a subset is identical to the gradient on the entire set
- Compute gradient on a subset and update the weights: “mini-batch”
- Extreme case: update gradient after each example: “online”
- Mini-batch is generally better
  - More accurate gradients
  - More efficient than full batch

This class

Learning with hidden units



Backpropagation

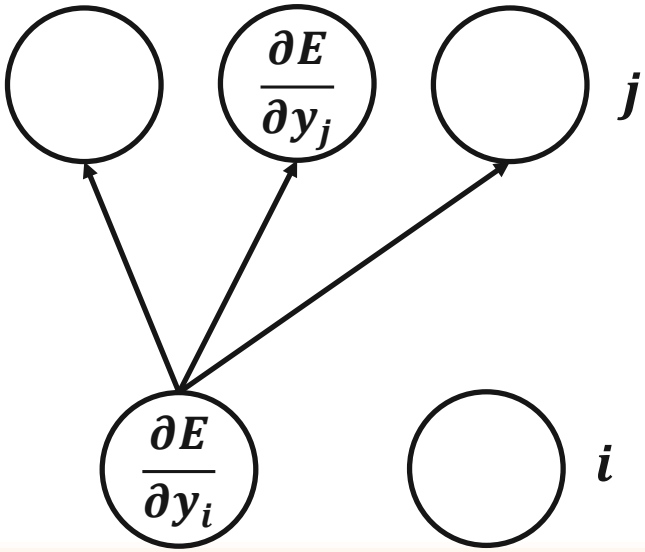
- Error is computed as the squared difference of target and actual output

$$E = \frac{1}{2} \sum_{j \in \text{output}} (t_j - y_j)^2$$

- Compute the error derivative wrt. the output

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j)$$

- Compute the error derivative of each hidden unit from error derivatives in the layer above



Slide based on Geoffrey Hinton course

Backpropagation

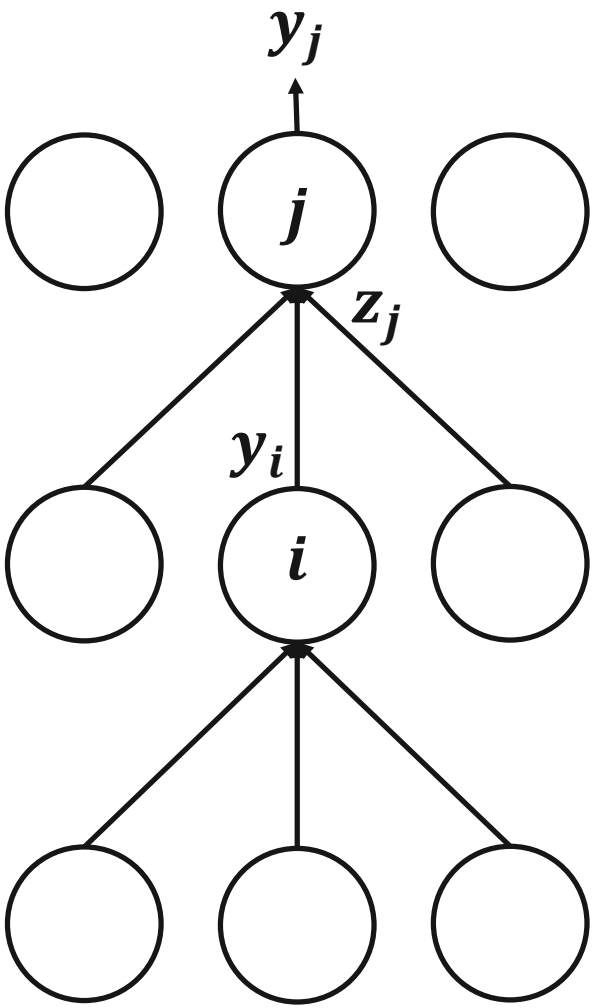
$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}$$

$$-(t_j - y_j)$$

Derivative of the activation function

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial y_i} = \sum_j \frac{\partial E}{\partial z_j} w_{ij}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} y_i$$

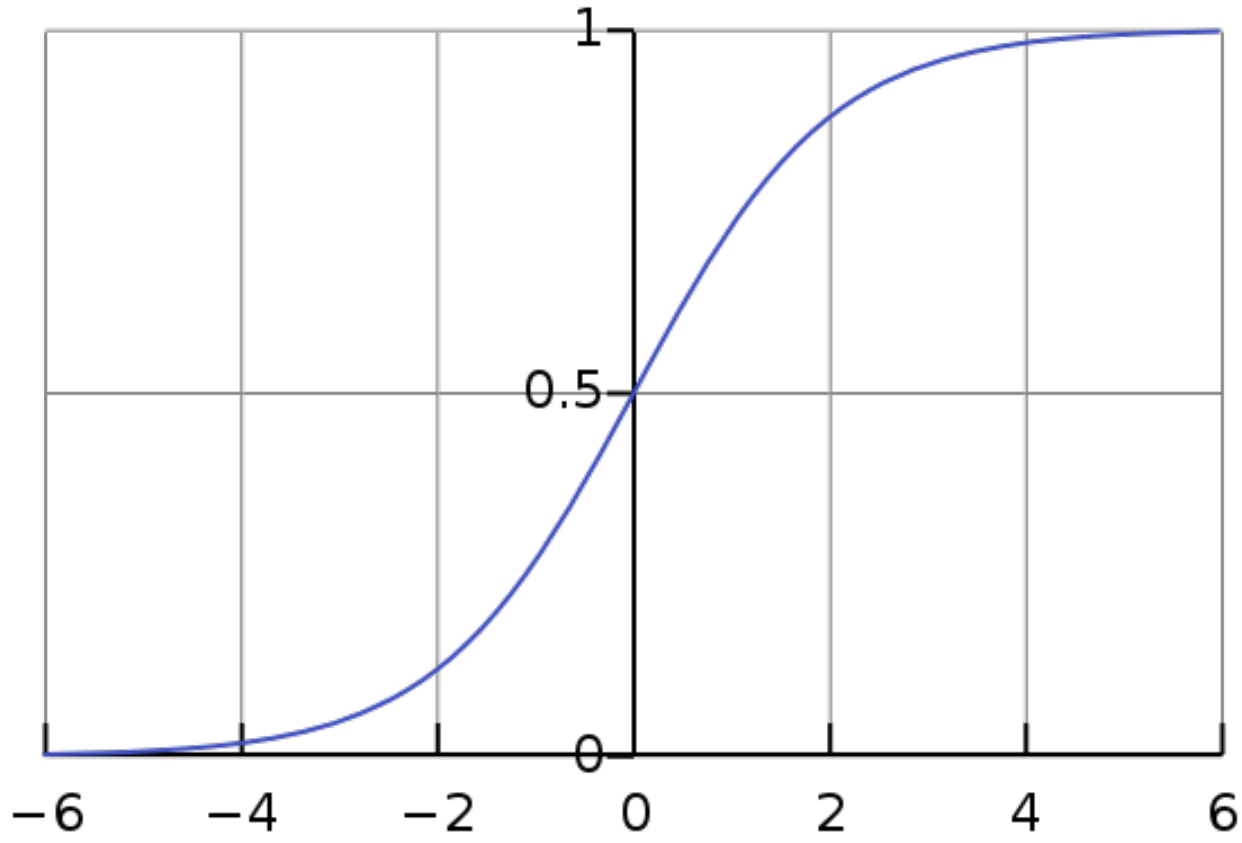


Activation functions

Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$

$$\frac{df}{dx} = f(x)[1 - f(x)]$$



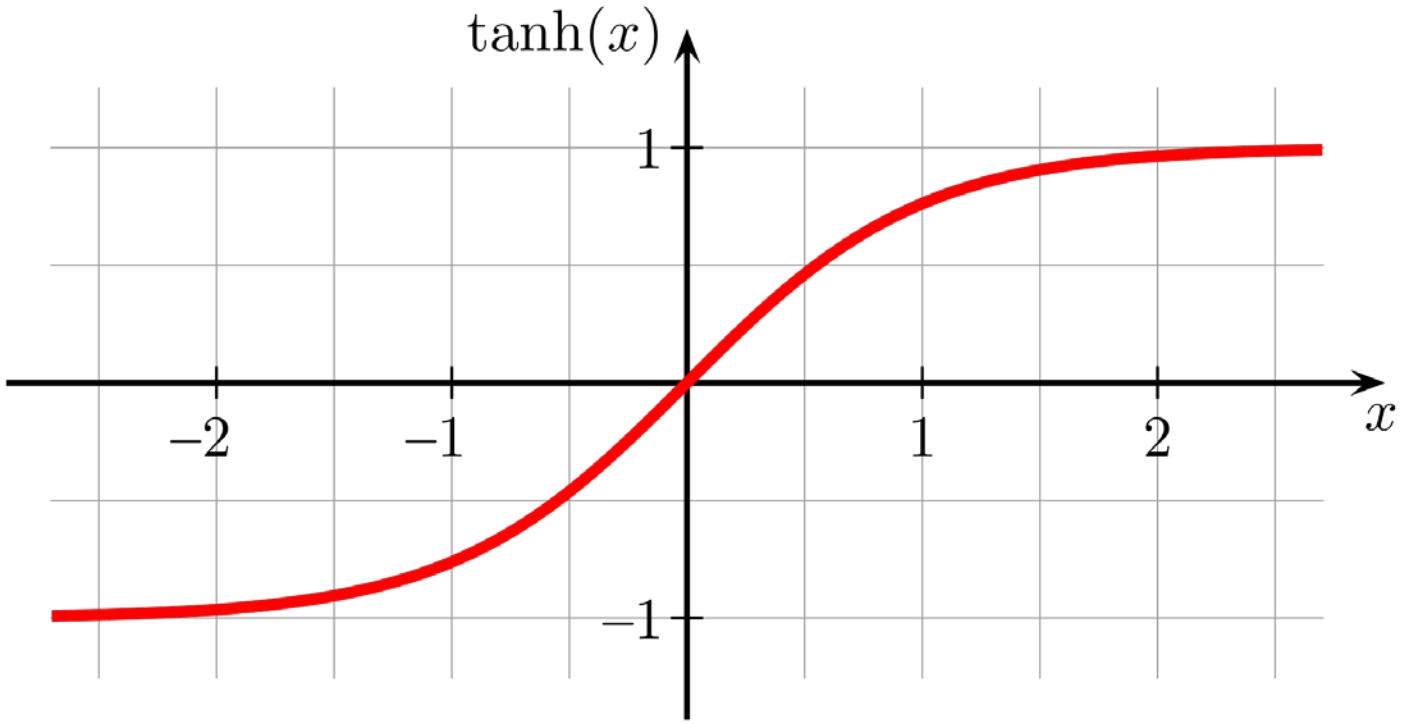
Sigmoid

- Pro
  - Between 0 and 1
- Con
  - Gradient almost zero at the tails

Tanh

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$\frac{df}{dx} = 1 - f(x)^2$$



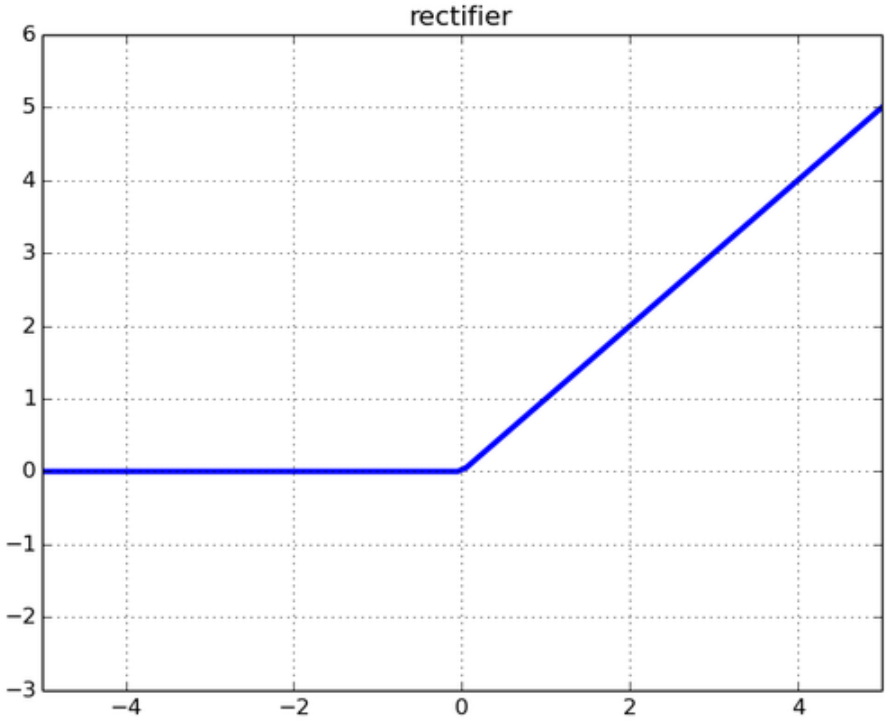
Tanh

- Pro
  - Between -1 and 1
- Con
  - Gradient almost zero at the tails

Rectified linear units (ReLU)

$$f(x) = \max(0, x)$$

$$\frac{df}{dx} = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$



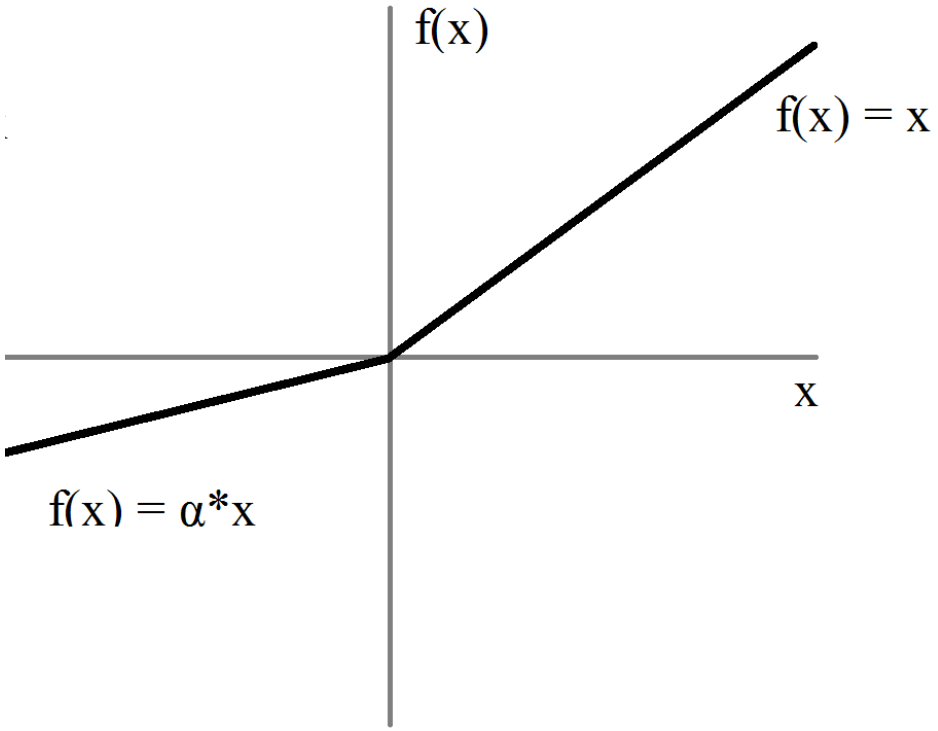
Rectified linear units (ReLU)

- Pros
  - Better gradient
  - Efficient
- Cons
  - Unbounded
  - Dying neurons

Most popular activation function!

Leaky rectified linear units

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases} \qquad \frac{df}{dx} = \begin{cases} 1, & x > 0 \\ \alpha, & x \leq 0 \end{cases}$$



Rectified linear units (ReLU)

- Pros
  - Better gradient
  - Efficient
- Con
  - Unbounded

Multilayer perceptron

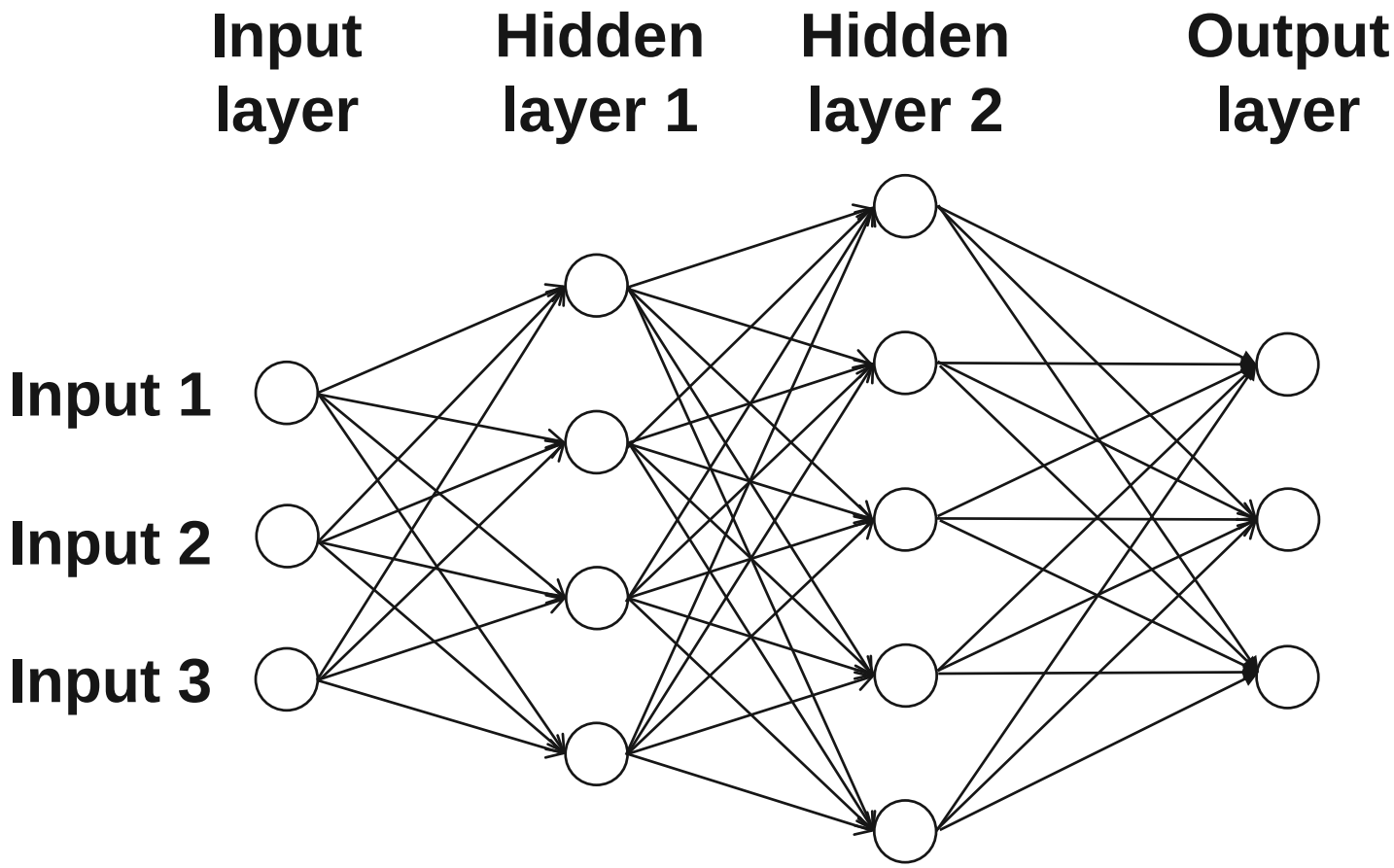


Image classification

- Hand crafted features
  - Not optimal!

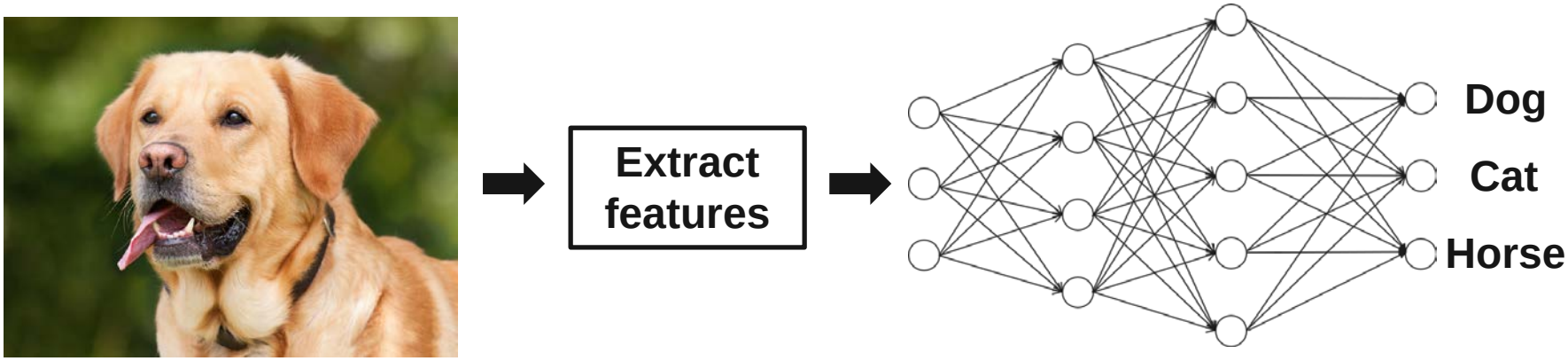
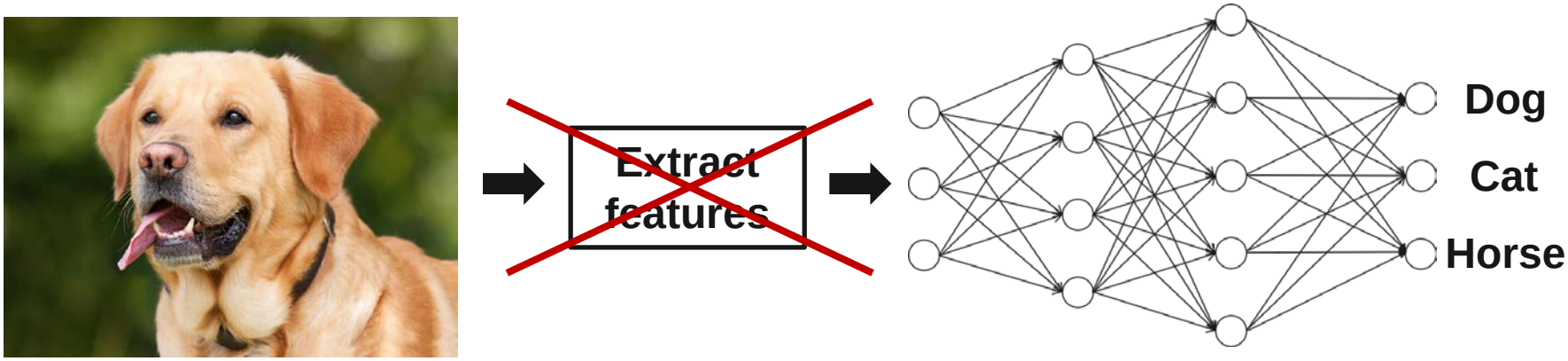


Image classification

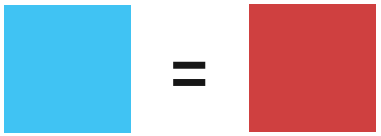


Requires input for every pixel of the image

Optimization is very difficult!

Convolutional neural network (CNN)

- Features are spatially invariant



Convolutional neural network (CNN)

- Compute dot product of the filter and the image in a sliding manner

32×32×3

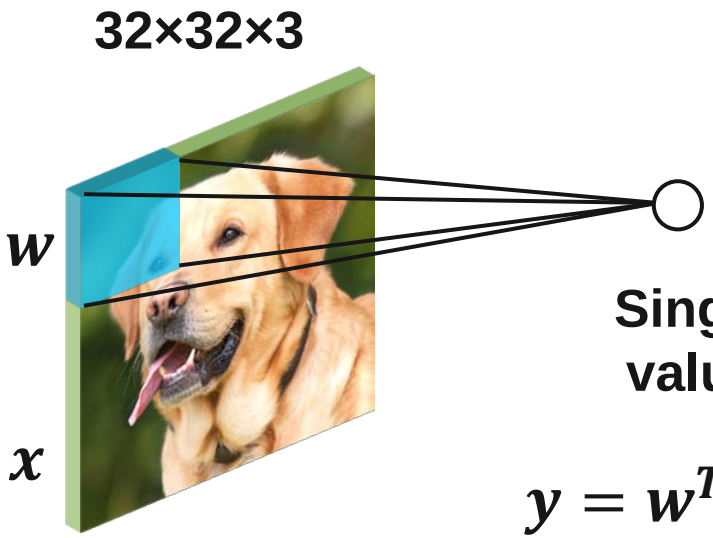
Input image

5×5×3

Filter

Convolutional neural network (CNN)

- Compute dot product of the filter and the image in a sliding manner

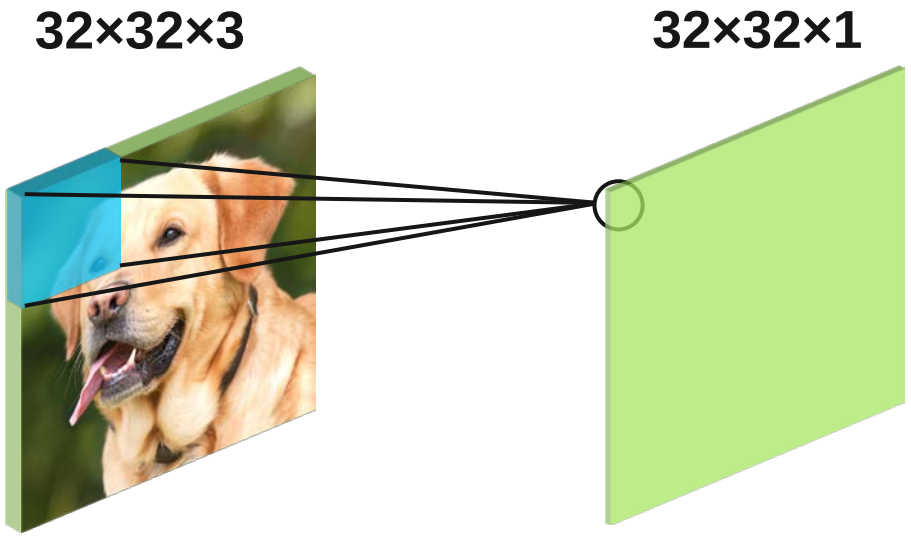


$$y = w^T x + b$$

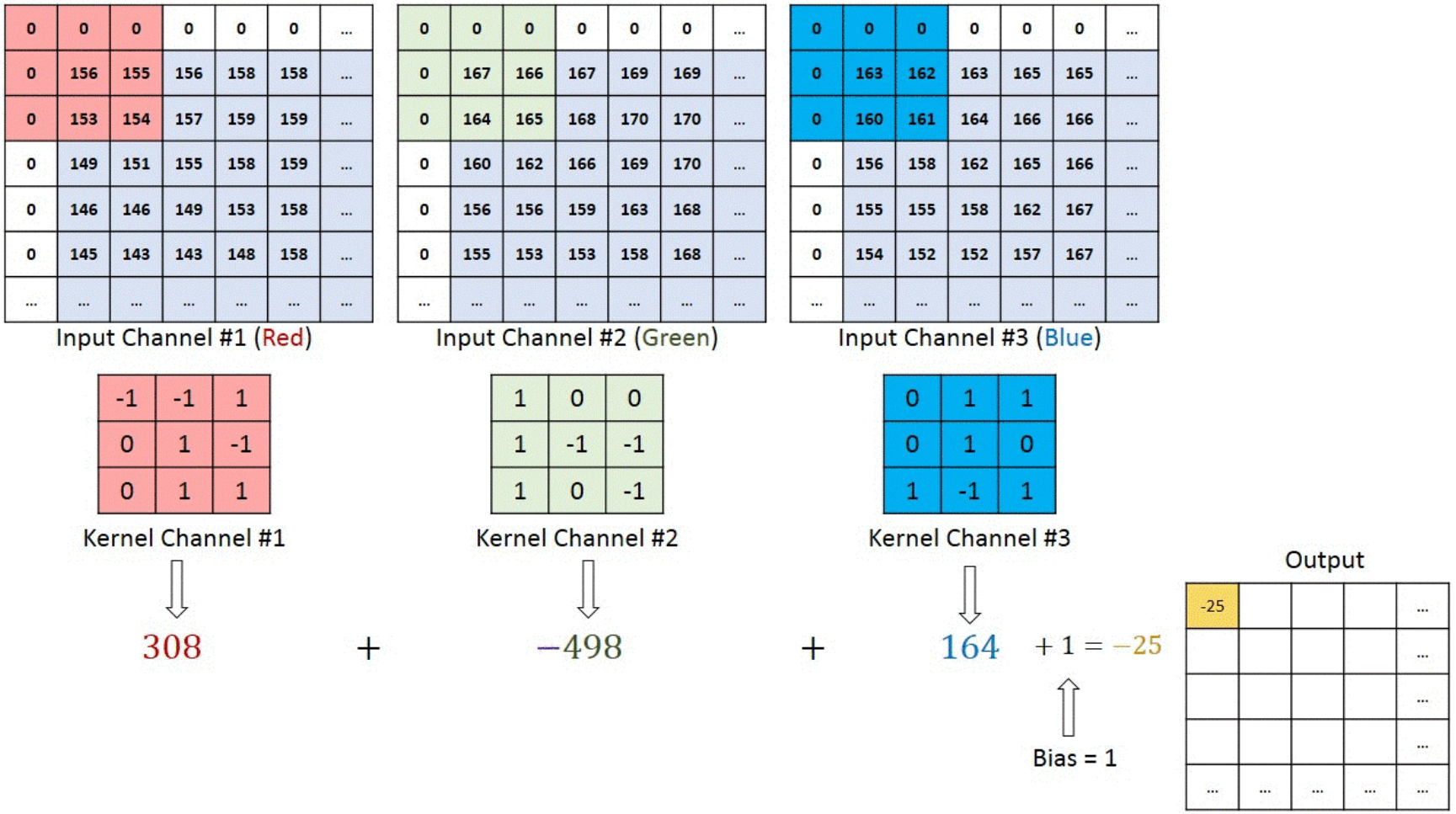
A non-linear activation function, e.g., ReLU, is applied to get the final value

Convolutional neural network (CNN)

- Compute dot product of the filter and the image in a sliding manner



Convolutional neural network (CNN)



Animation from Machine Learning Guru

Convolutional neural network (CNN)

32×32×3

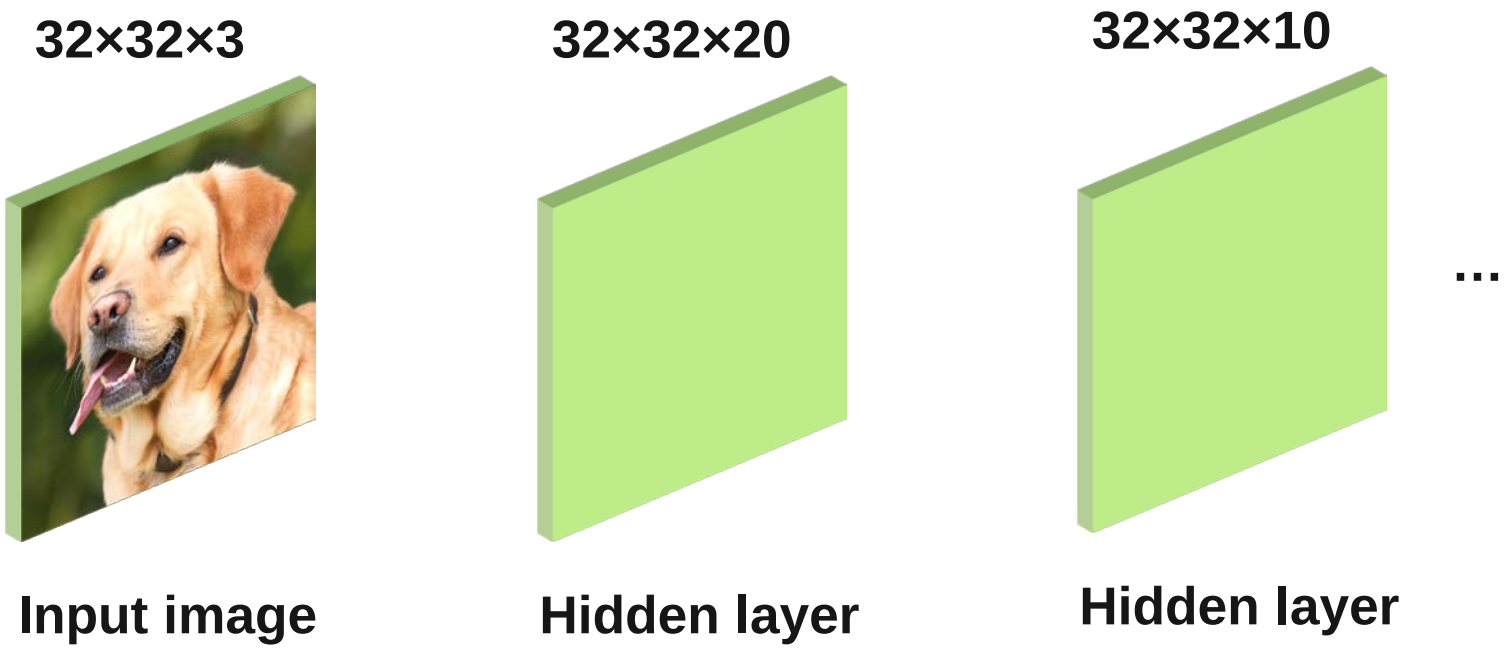
4×5×5×3

Input image

32×32×4

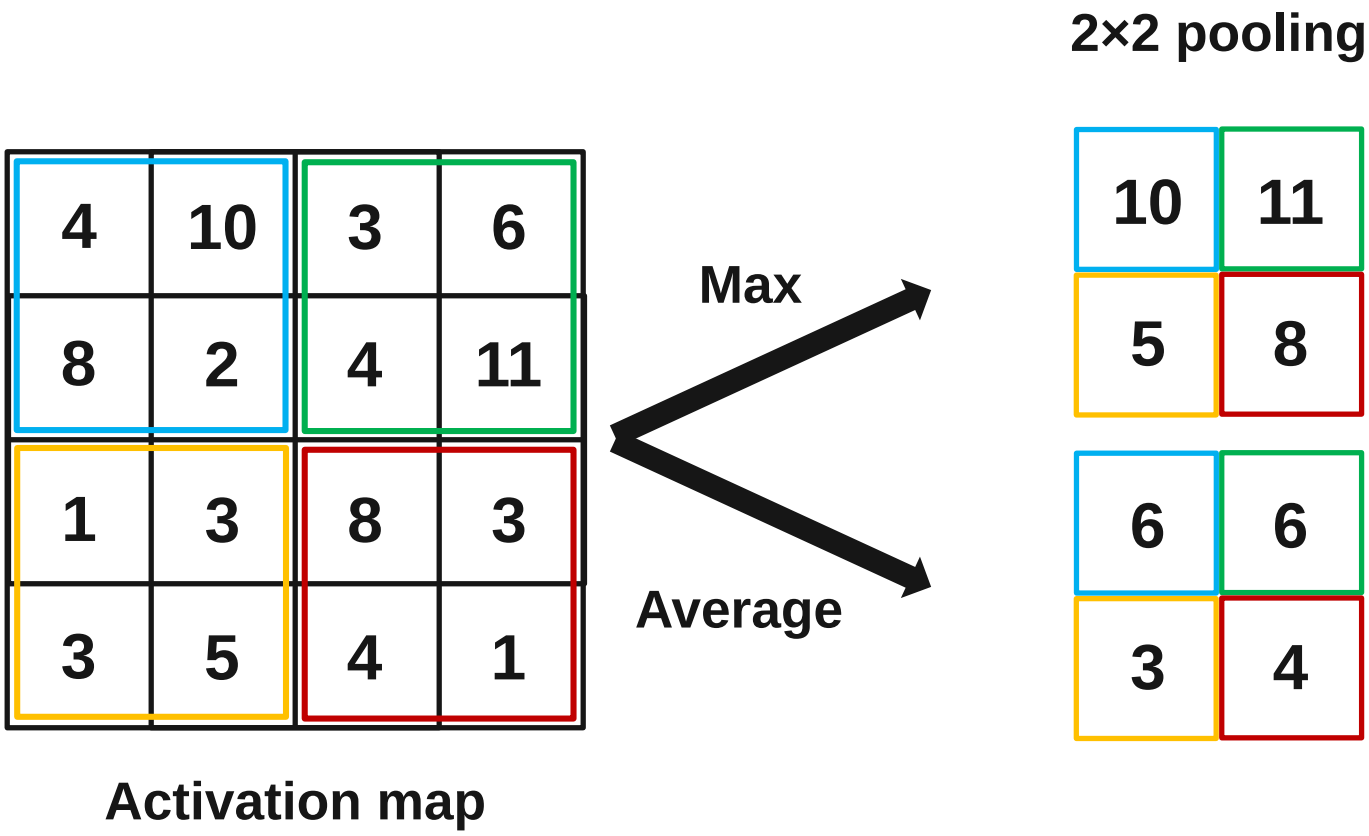
Activation map

Convolutional neural network (CNN)



Pooling layer

- A way to compress the feature maps



Stride

- Another way of compressing the features
- Stride of *m* means we slide the filter *m* pixels over when computing the convolutions

LeNet (1998)

- Digit recognition

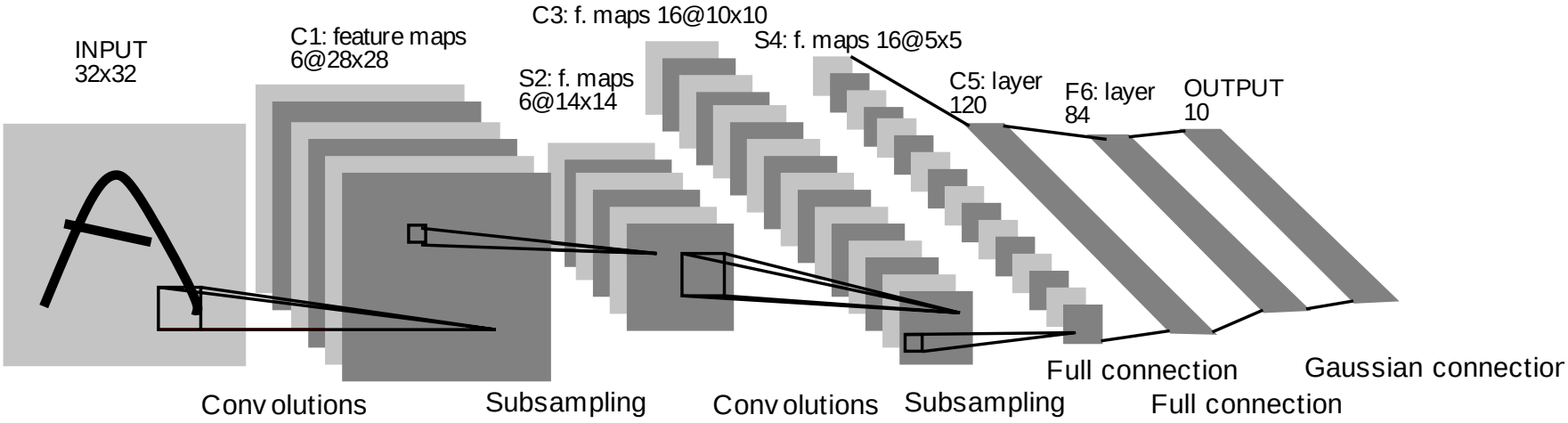
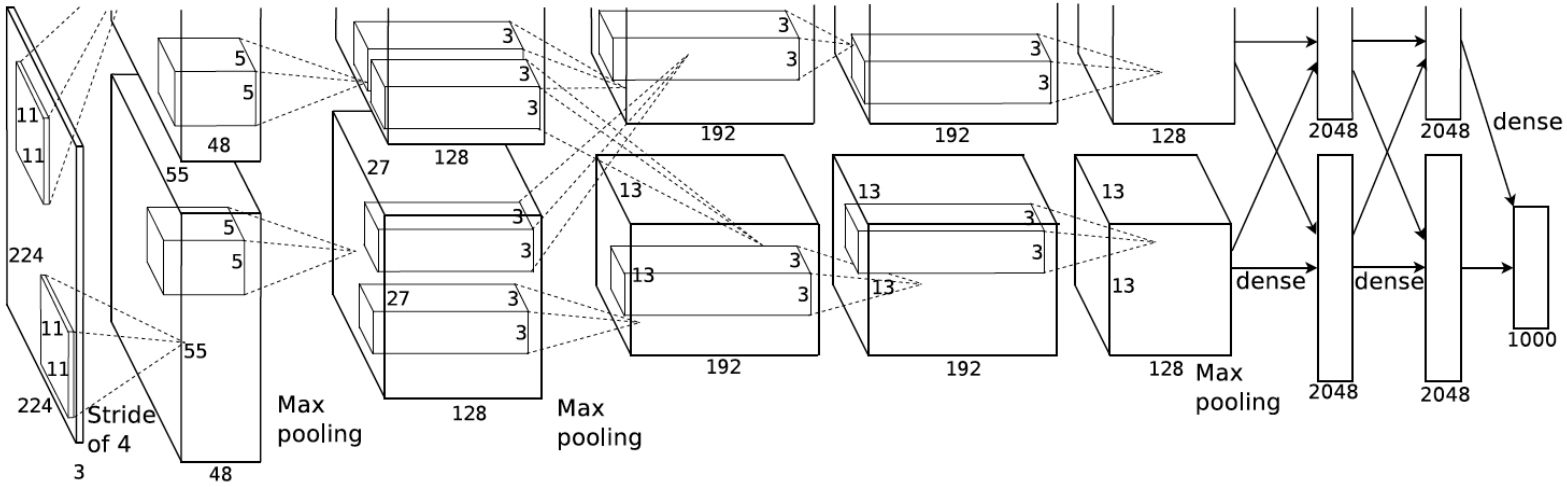


Image classification

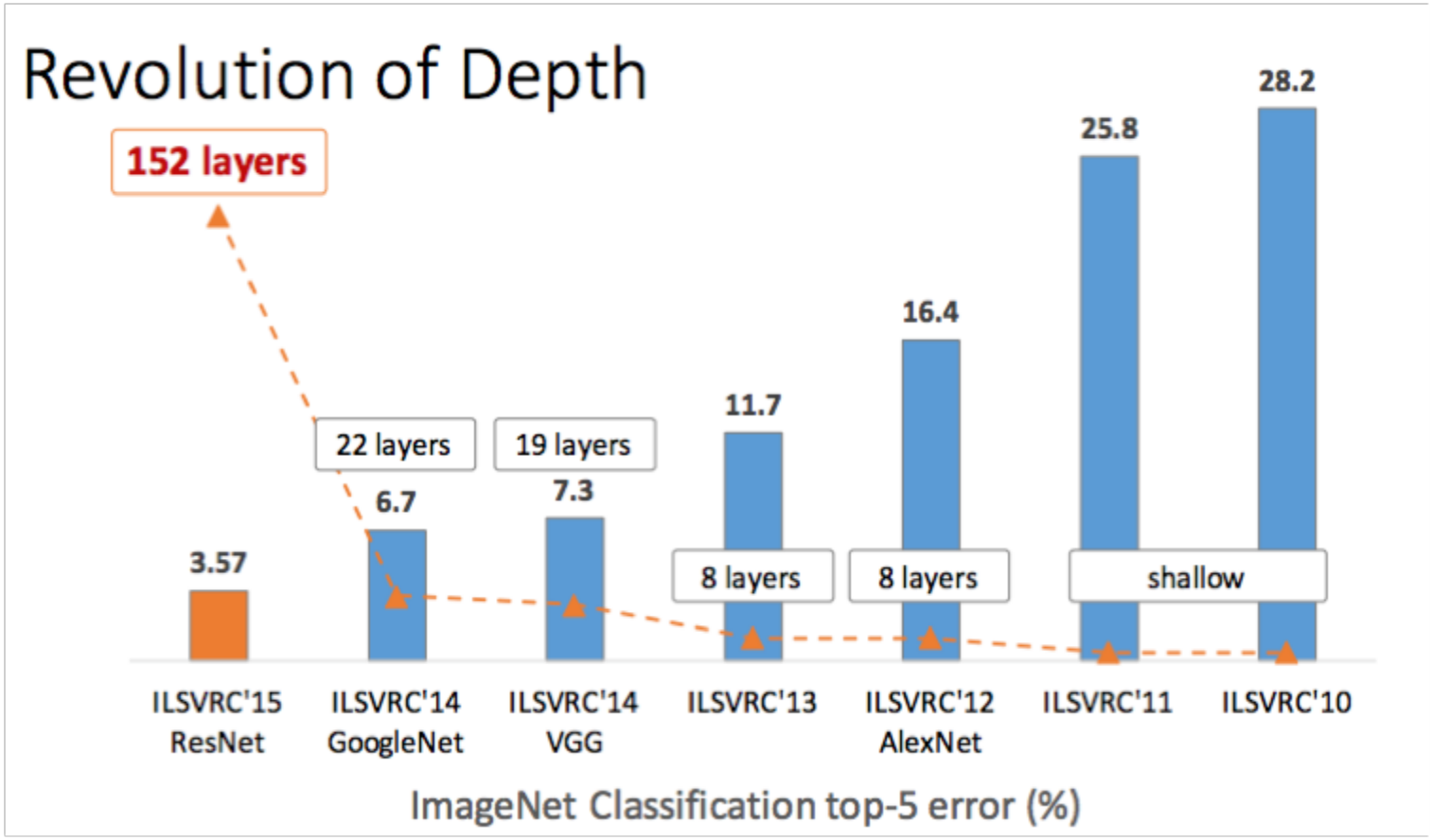
- Recognizing objects in color images is more complicated than digit recognition
  - Hundred times as many classes (1000 vs 10)
  - Hundred times as many pixels (256×256 color vs. 28×28 gray)
  - 2D images of 3D scenes
  - Cluttered scenes requiring segmentation
  - Multiple objects in each image

AlexNet (2012)

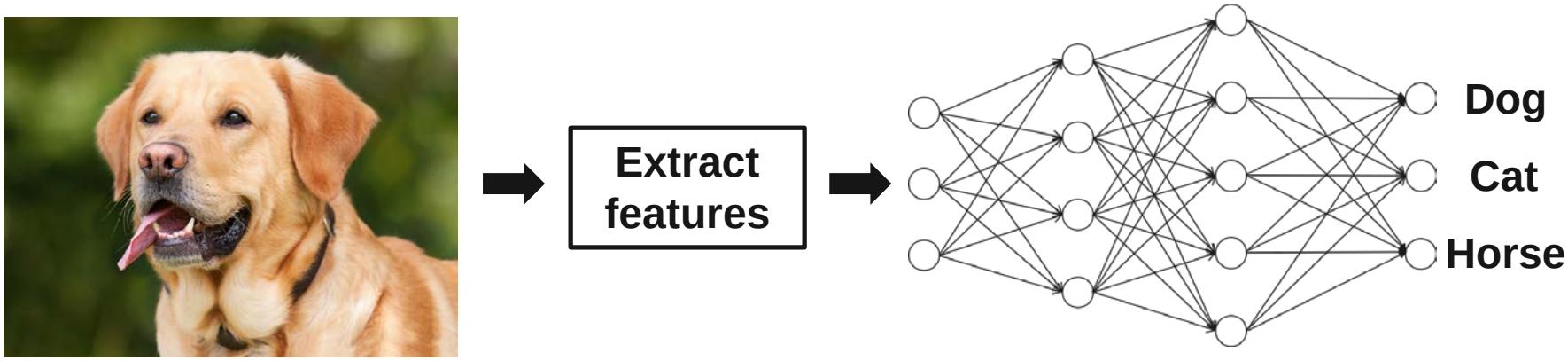
- 8 layers
- First few layers convolutional, the rest fully connected
- Improved the classification accuracy on ImageNet from 25.8% to 16.4%



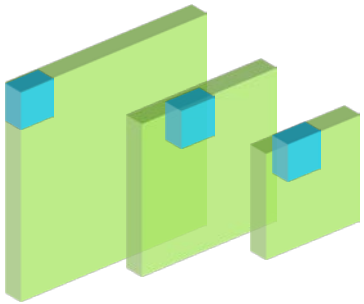
ILSVRC winners



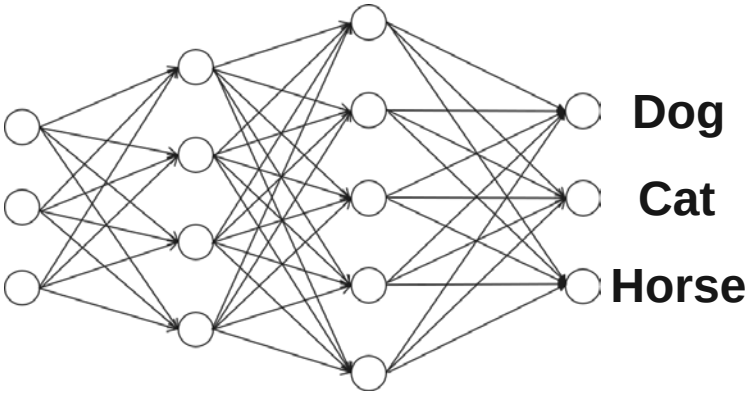
So far…



So far…

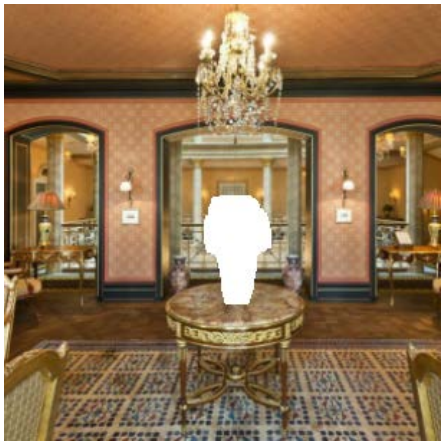


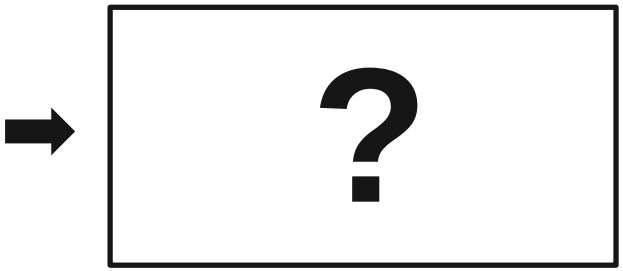
CNN



However…

- In most graphics applications the goal is to synthesize an image



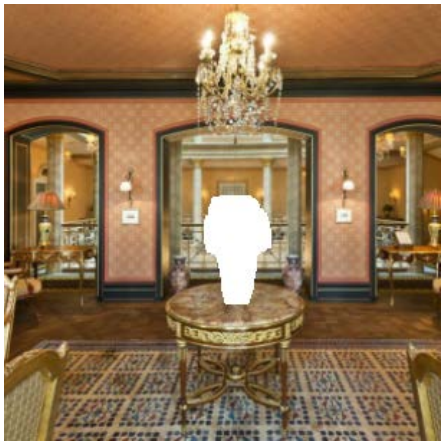


Input

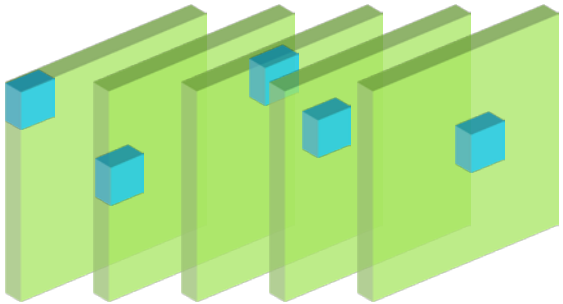
Output

One solution

- Use convolutional layers without pooling
- Problem
  - Small receptive field

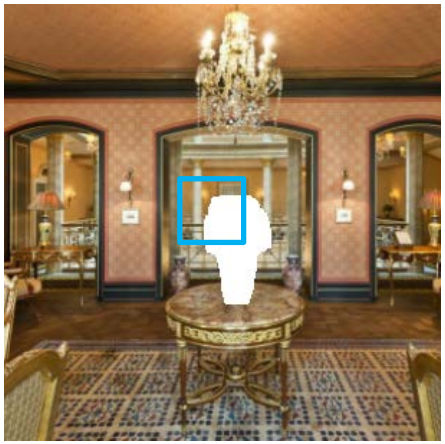


Input

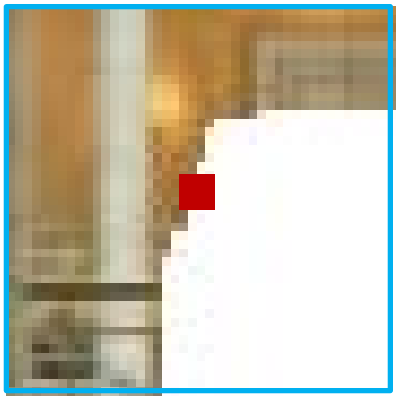


Output

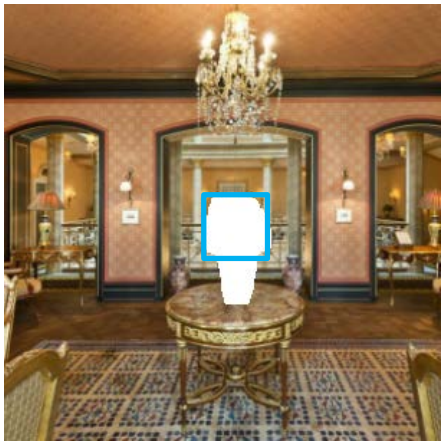
Small receptive field problem



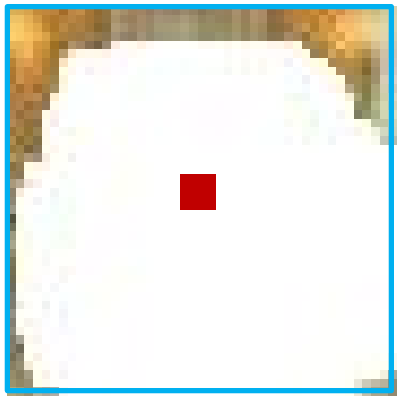
Input



Small receptive field problem

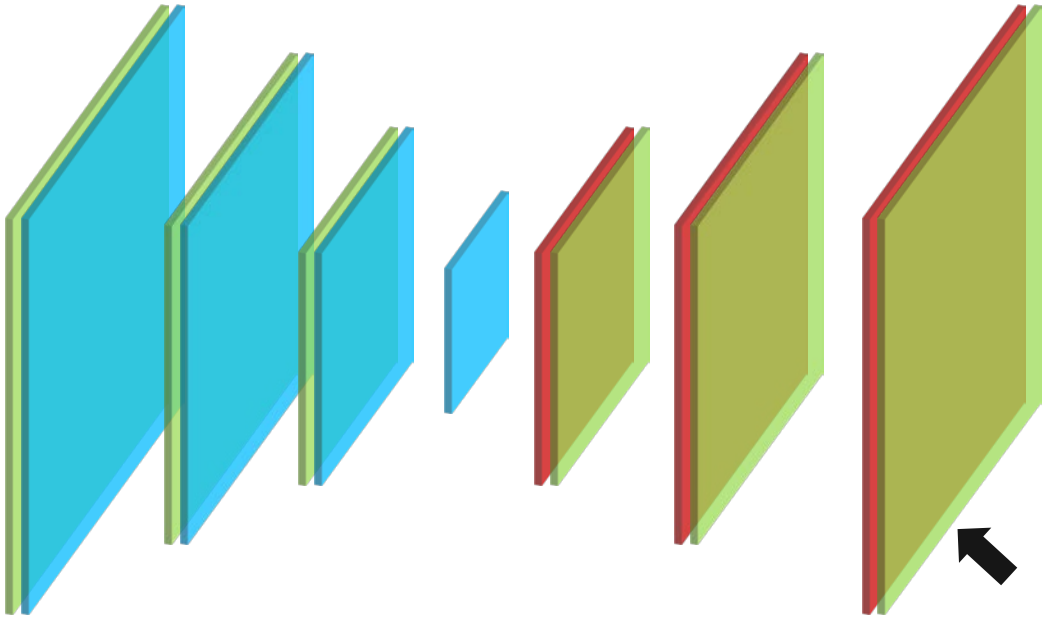


Input



Encoder-decoder architecture

- Problem: losing spatial details

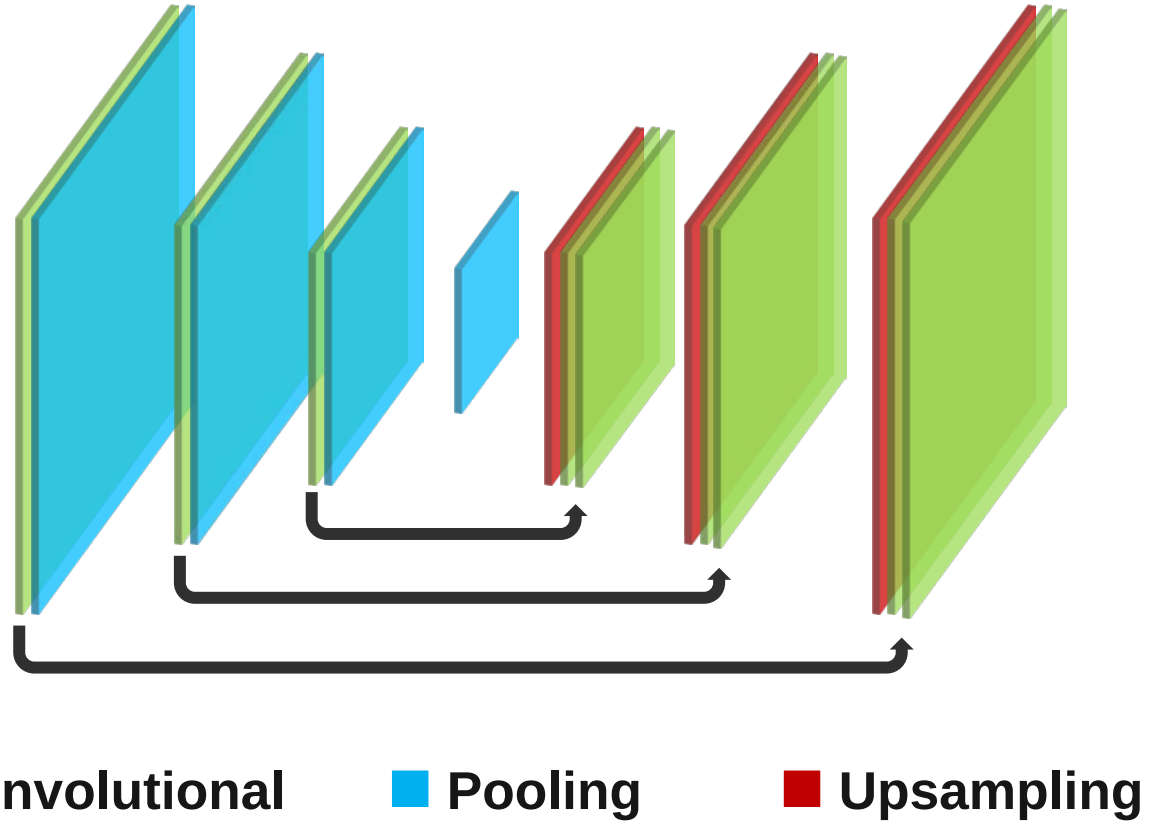


Deconvolutional with stride *m*

Convolutional Pooling Upsampling

Skip connections

- U-Net

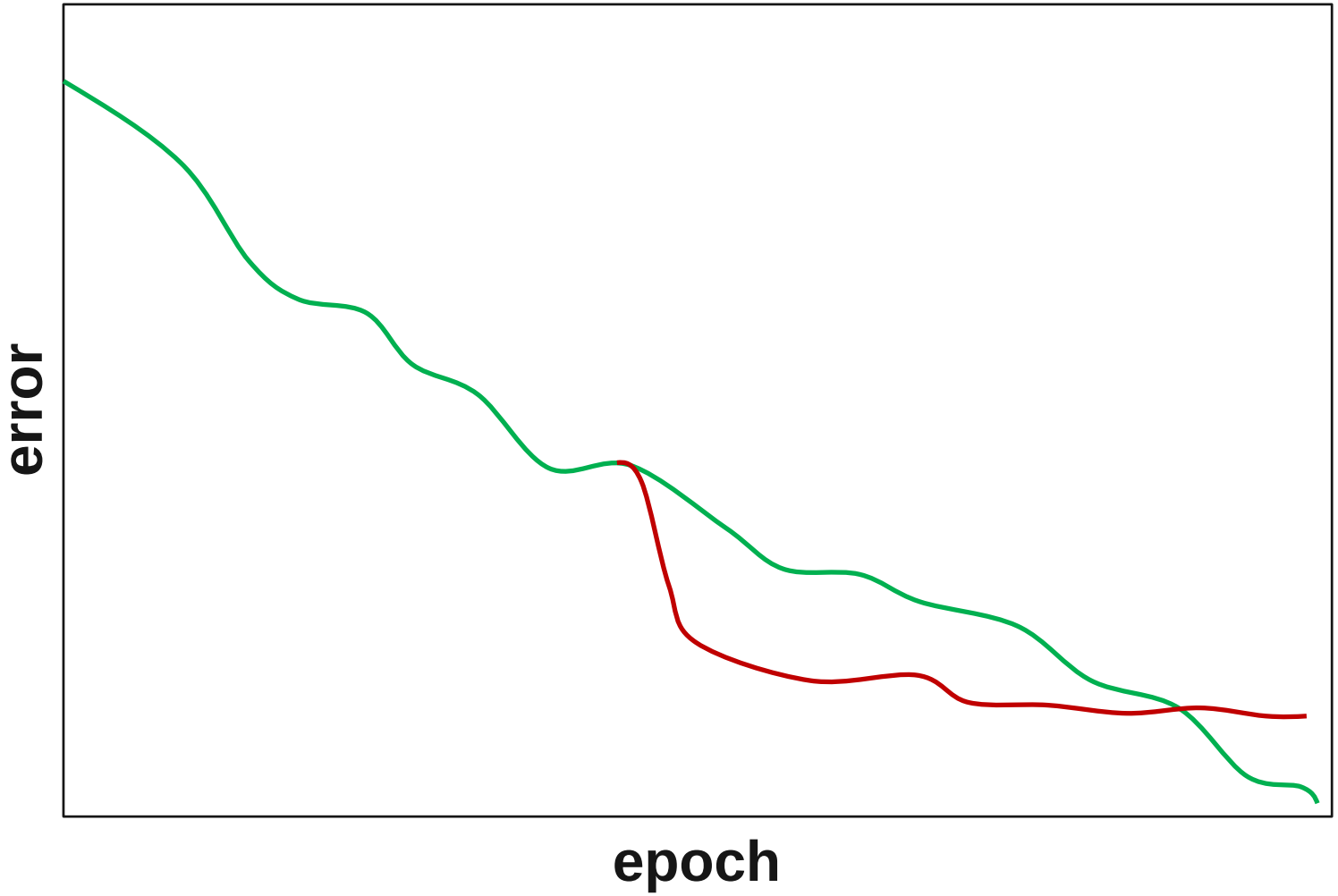


Tricks for effective training

Setting the learning rate

- Guess an initial learning rate
  - If the error gets worse or oscillate, reduce the rate
  - If the error decreases consistently increase the rate
  - Towards the end reduce the rate

But be careful

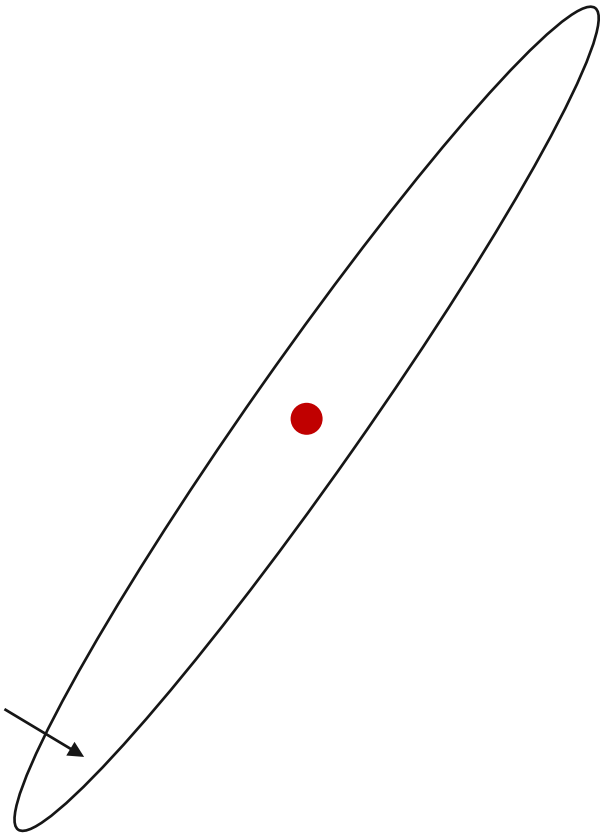


Initialization

- If hidden units have exactly the same weights and biases, they will always get the same gradient
  - Initialize with small random values
- If a hidden unit has a big fan-in, small changes on many incoming weights can cause the learning to overshoot
  - Initialize the weights proportional to sqrt(fan-in)
  - Scale the learning rates

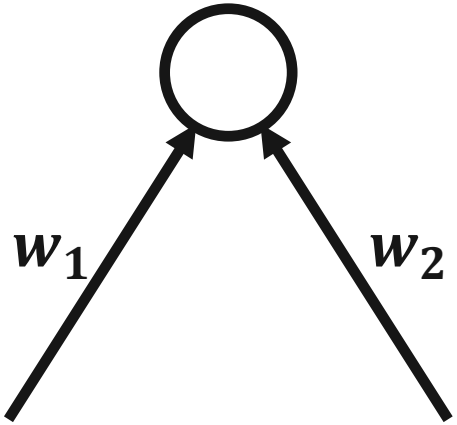
Learning

- **Direction of steepest descent does not point at the minimum unless the ellipse is a circle**



Shifting the inputs

- Make sure input vector has zero mean over the whole training set

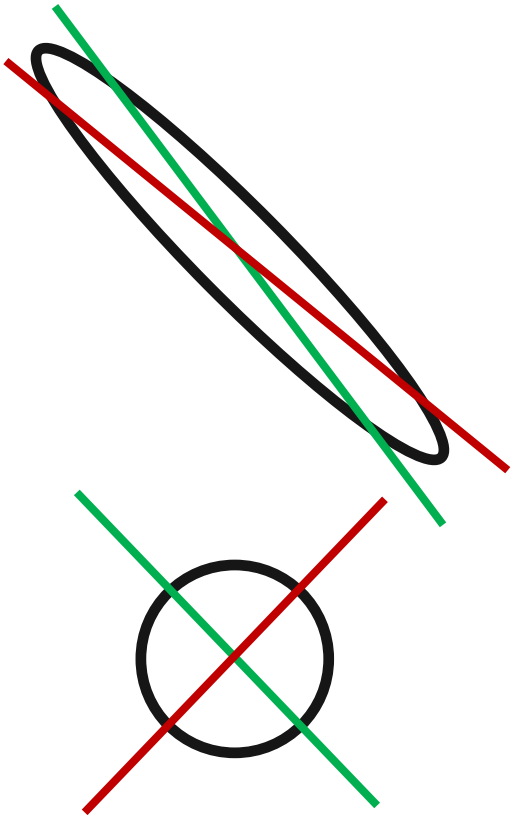


101, 101 → 2

101, 99 → 0

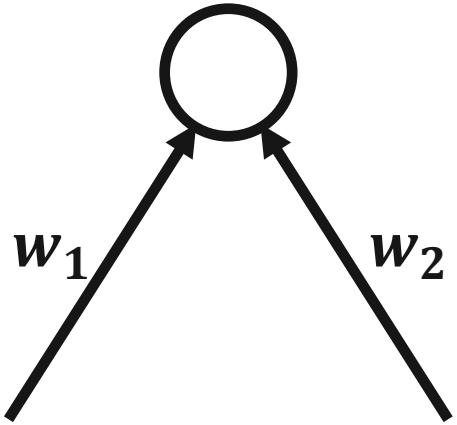
1, 1 → 2

1, -1 → 0



Scaling the inputs

- Make sure input vector has unit variance over the whole training set

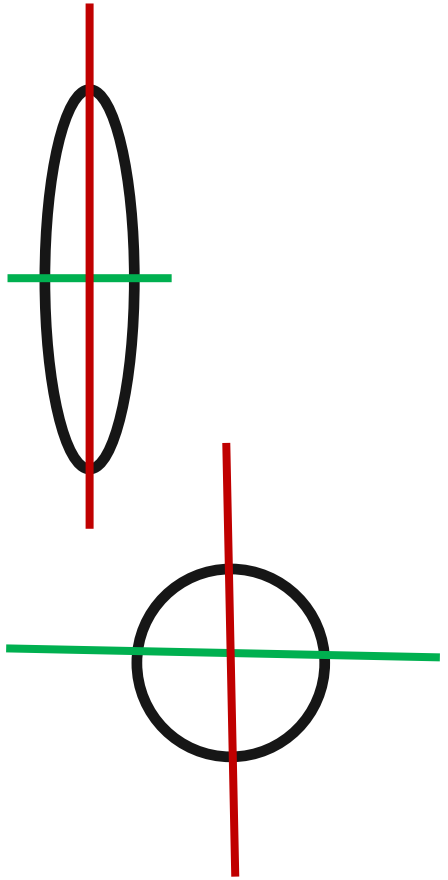


0.1, 10 ⟶ 2

0.1, -10 ⟶ 0

1, 1 ⟶ 2

1, -1 ⟶ 0

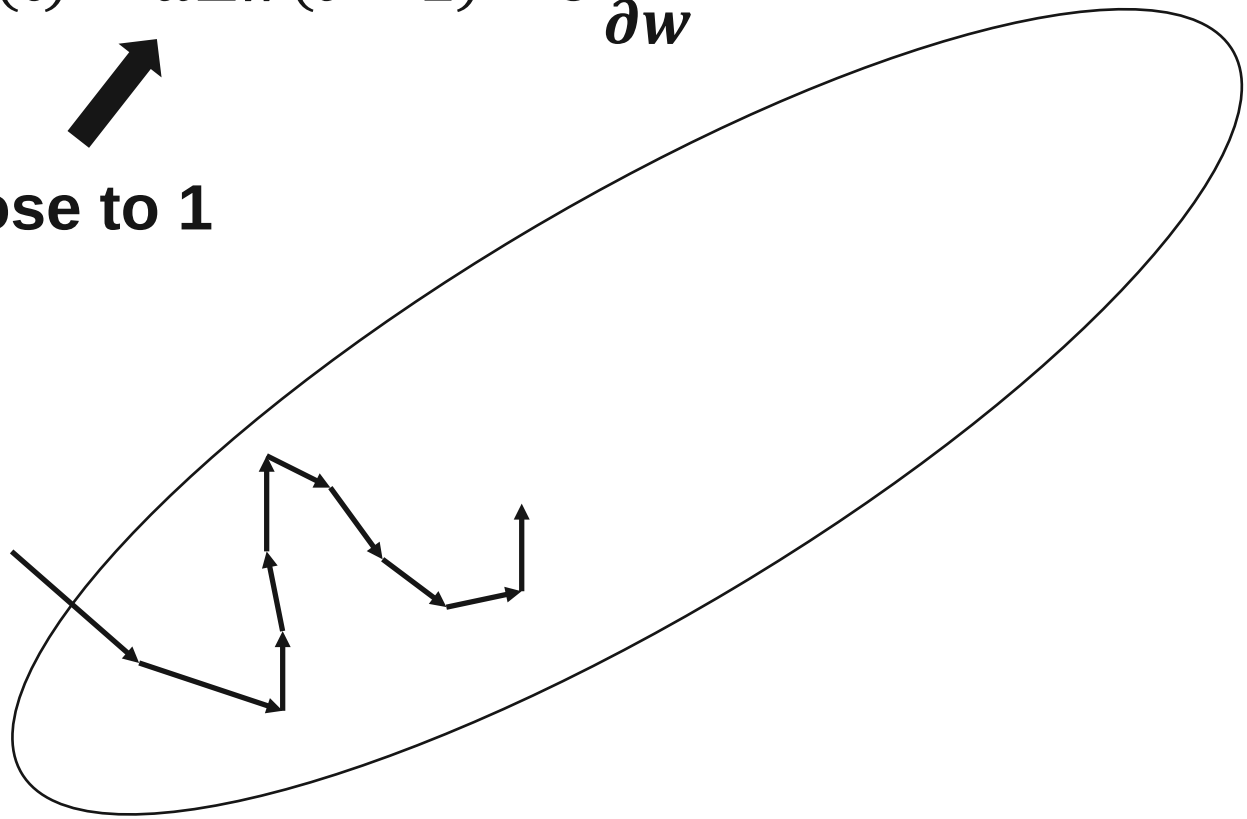


Optimization methods

Momentum

$$\Delta w(\mathrm{t}) = \alpha \Delta w(t-1) - \varepsilon \frac{\partial E}{\partial w}$$

Close to 1



rprop and rmsprop

- rprop: a full batch learning approach
  - Adaptive step size for each weight
- rmsprop: extension of rprop for mini-batch

rprop

- Gradient's magnitude can be different for different weights and can change during training
  - Difficult to choose a single global learning rate
- Use the sign of the gradient for full batch learning
  - The weight updates are all the same size
  - Easier to escape from plateaus
- Additionally adapt the step size
  - Increase the step size multiplicatively (e.g., 1.2 times) if the sign of last two gradients agree
  - Otherwise decrease it multiplicatively (e.g., 0.5 times)
  - Limit the step sizes

Does it work with mini-batches?

- In stochastic gradient descent, the gradient is averaged over successive mini-batches
  - Consider a case where a weight gets a gradient of +0.1 nine times and -0.9 once
  - The weight should not change after these ten mini-batches
- rprop increases the weight 9 times and decreases it only once
  - So the weight would grow a lot

rmsprop

- rprop basically uses the gradient but also divides it by the size of the gradient
  - **Problem:** for each mini-batch we divide the gradient by a different number
  - **Solution:** force the number we divide by to be similar to adjacent mini-batches
- rmsprop: keep a moving average of square gradient of each weight

$$M(w,t) = 0.9\, M(w,t-1) + 0.1 \left(\frac{\partial E}{\partial w}\right)^2$$

- Divide the gradient by $\sqrt{M(w,t)}$

Recent approaches

- Adam
- Adadelta
- Adagrad
- Most of them are very similar to rmsprop