The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

Spring 2021

ADVANCED TOPICS IN COMPUTER VISION

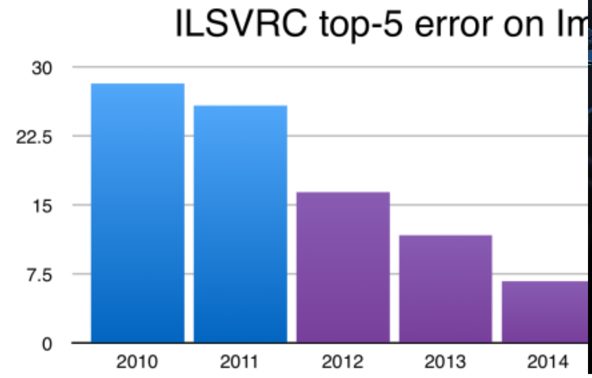
Atlas Wang
Assistant Professor, The University of Texas at Austin

Visual Informatics Group@UT Austin
https://vita-group.github.io/

2019 Turing Award winners, left to right, Yann LeCun, Geoff Hinton, and Yoshua Bengio, reoriented artificial intelligence around neural networks

A Triumph of Deep Learning: 2012 - present

Top-performers in many tasks, over many domains



nature

THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE

At last — a computer program that can beat a champion Go player PAGE 484

ALL SYSTEMS GO

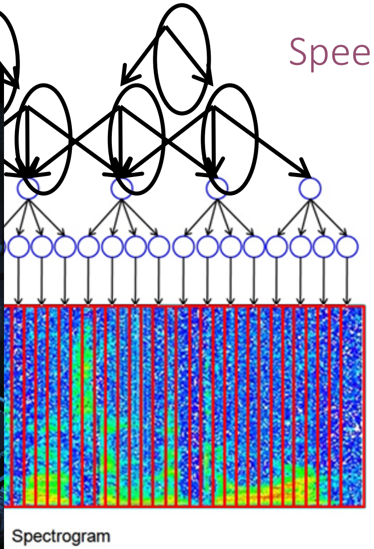
CONSERVATION
SONGBIRDS À LA CARTE
Illegal harvest of millions of Mediterranean birds
PAGE 452

RESEARCH ETHICS
SAFEGUARD TRANSPARENCY
Don't let openness backfire on individuals
PAGE 459

POPULAR SCIENCE
WHEN GENES GOT 'SELFISH'
Dawkins's calling card forty years on
PAGE 462

NATURE.COM/NATURE
28 January 2016 £10
Vol. 529, No. 7587

Speech Recognition



NLP

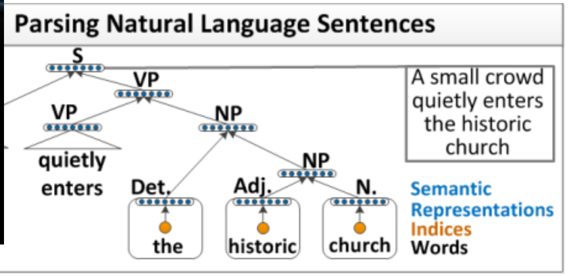
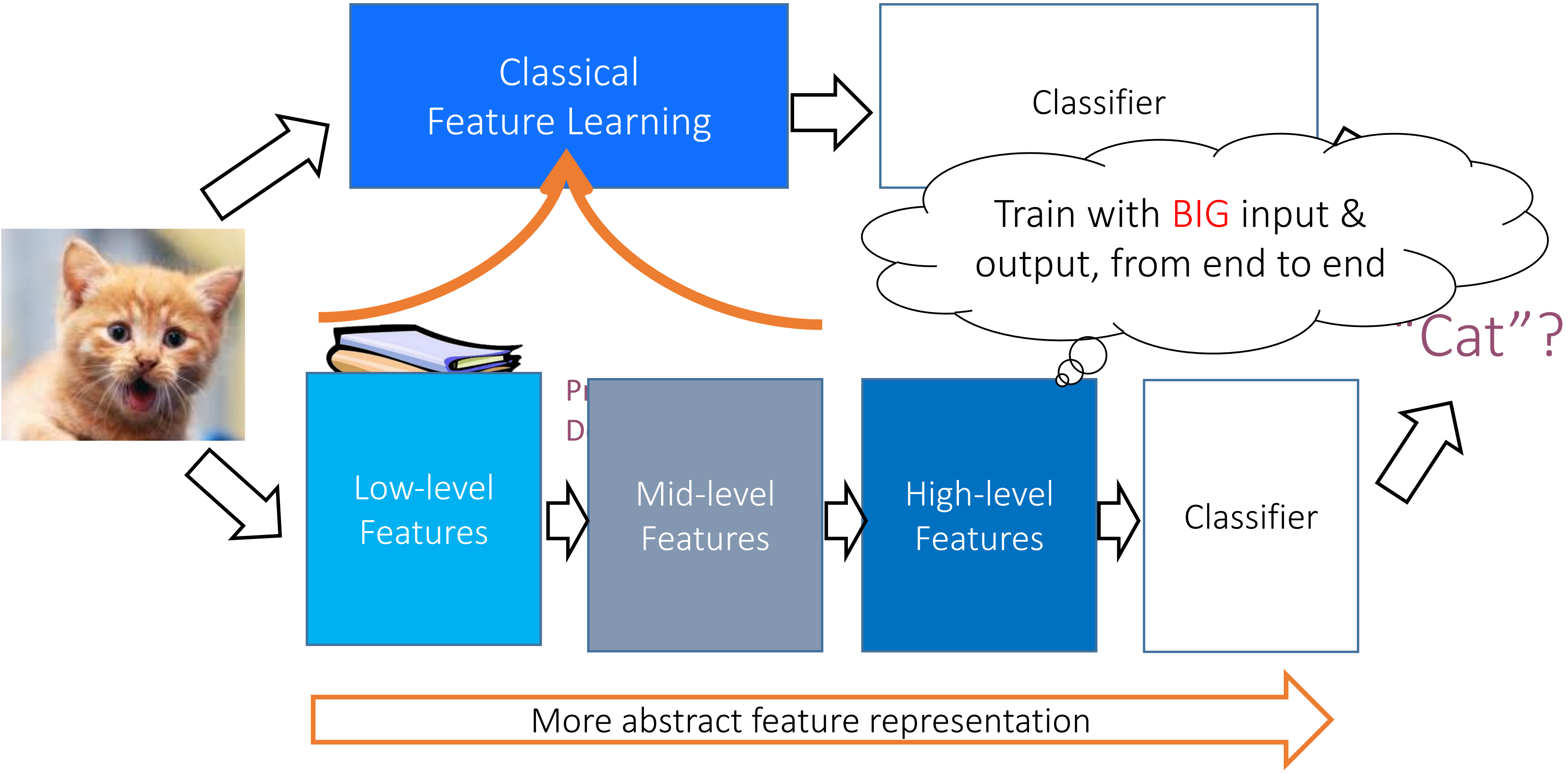


Image classification, detection, localization...

Feature learning: Going Deep



Classical Feature Learning

Classifier

Train with BIG input & output, from end to end

"Cat"?

Pr

D

Low-level Features

Mid-level Features

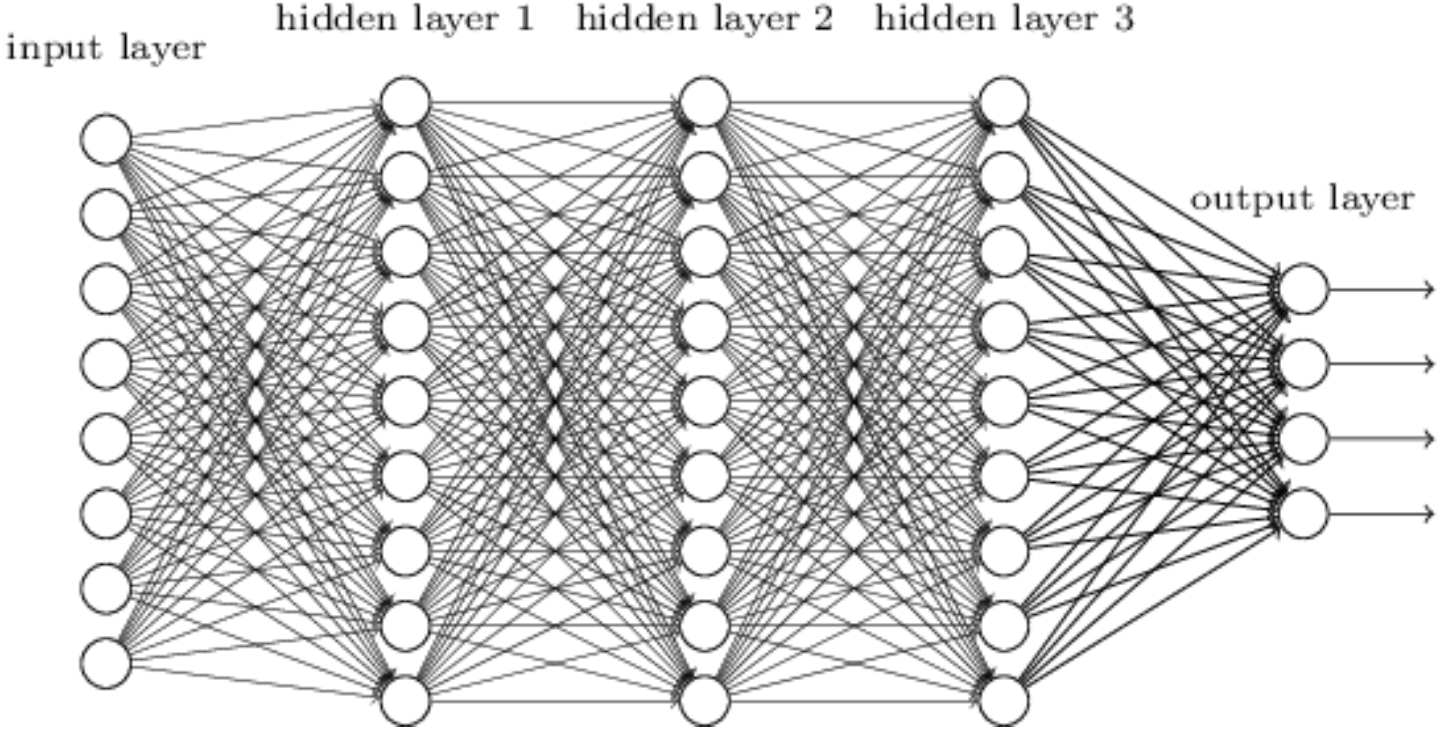
High-level Features

Classifier

More abstract feature representation

Deep learning

- Learn a *feature hierarchy* all the way from raw inputs (e.g. pixels) to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly



Status Quo

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

ResNet, 152 layers
(ILSVRC 2015)

Current Trend:

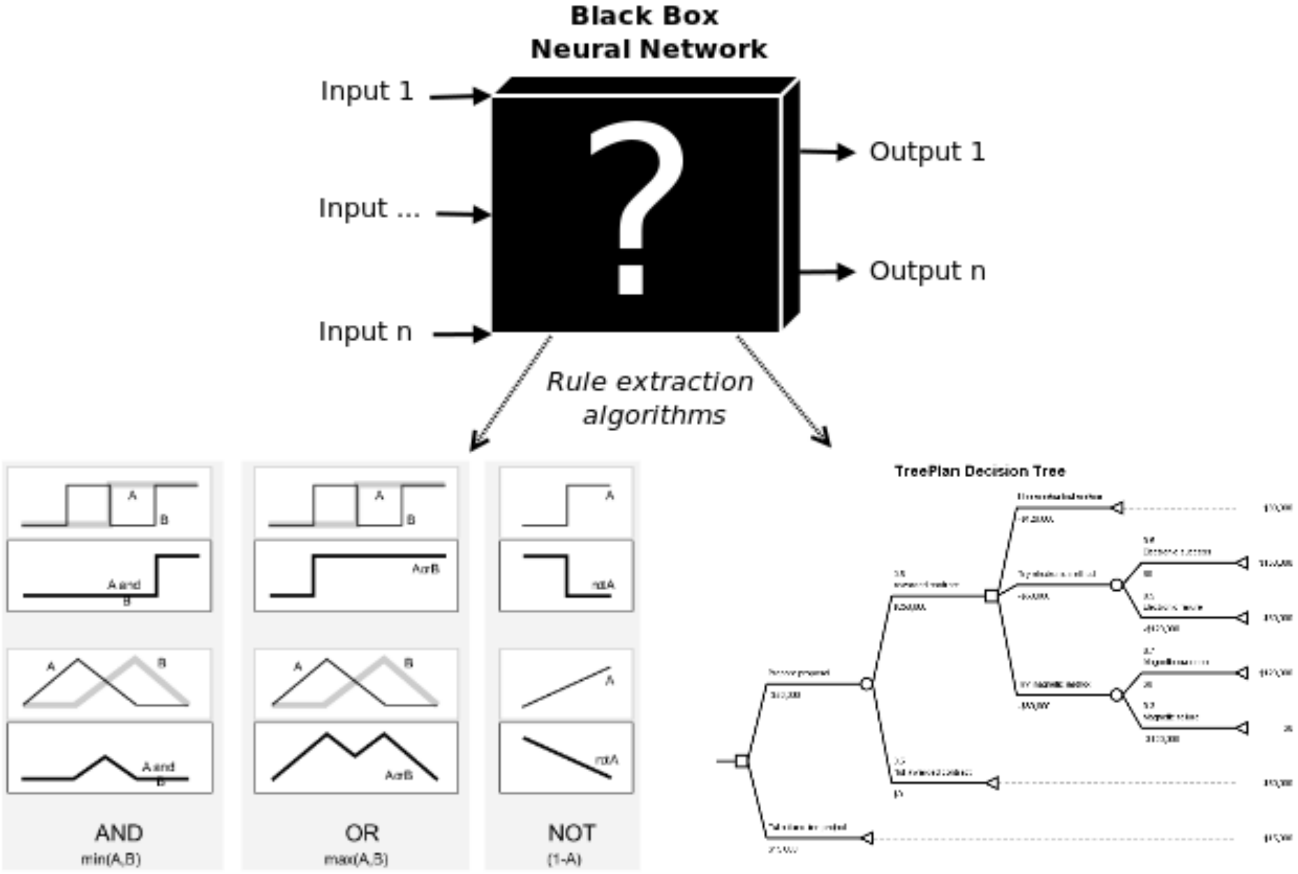
- To build increasingly larger, deeper networks, trained with more massive data, based on the benefits of high-performance computing.
- Play with the connectivity and add "skips"


I WAS WINNING IMAGENET
UNTIL A DEEPER MODEL CAME ALONG
imgflip.com


专业深度学习调试
caffe安装 10元
CNN 5元/层
RNN 8元/层
专业

Grand Challenges

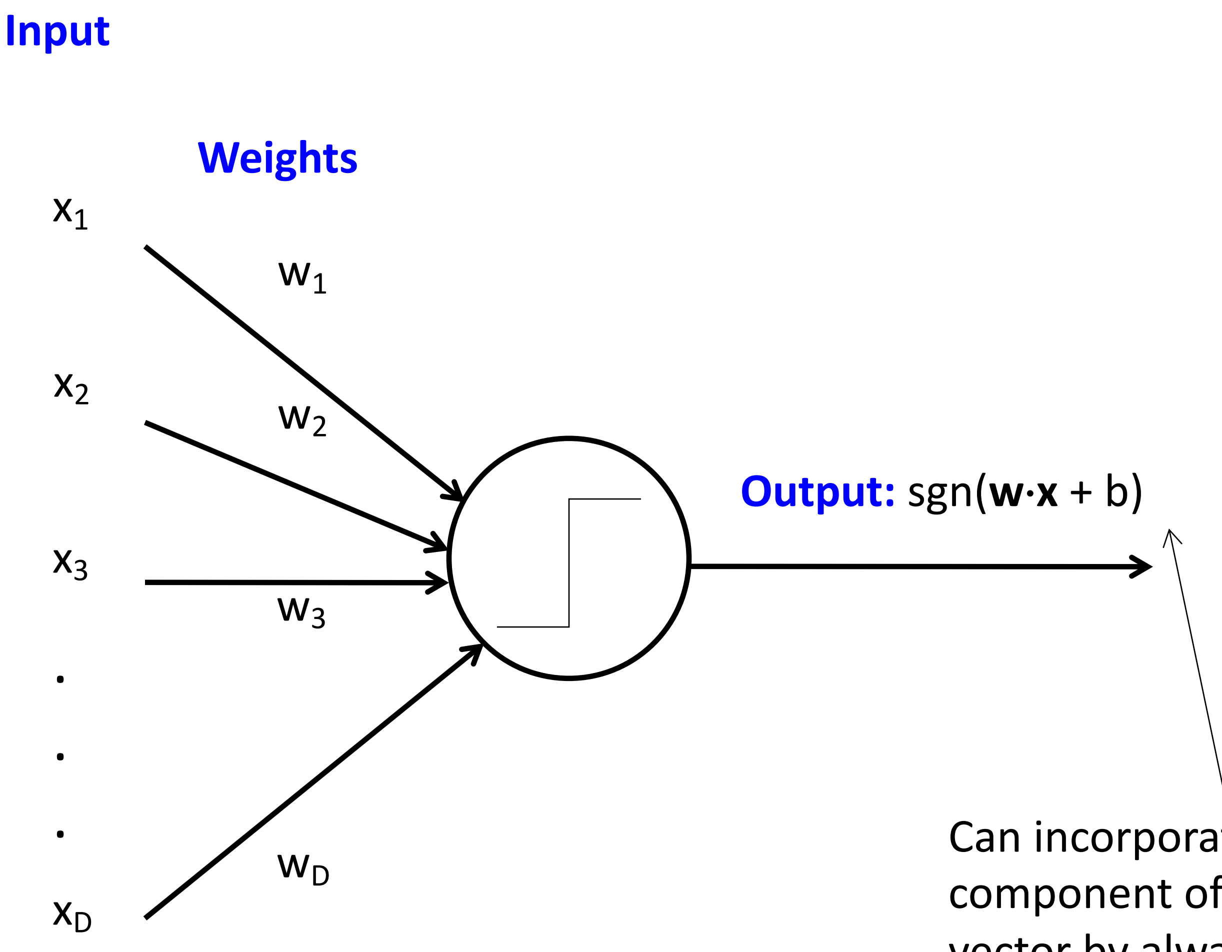
- Why/how deep learning works?
  - *In theory, many cases shouldn’t even work…*
  - Gap between engineering (or art) and science: Lack of theoretical understandings & guarantees, and analytical tools
  - Training is computationally expensive and difficult, relying on many “magics”
  - No principled way to incorporate domain expertise, or to interpret the model behaviors



Extract Logic in either symbolic or fuzzy form

Extract Decision Trees contraining Logic

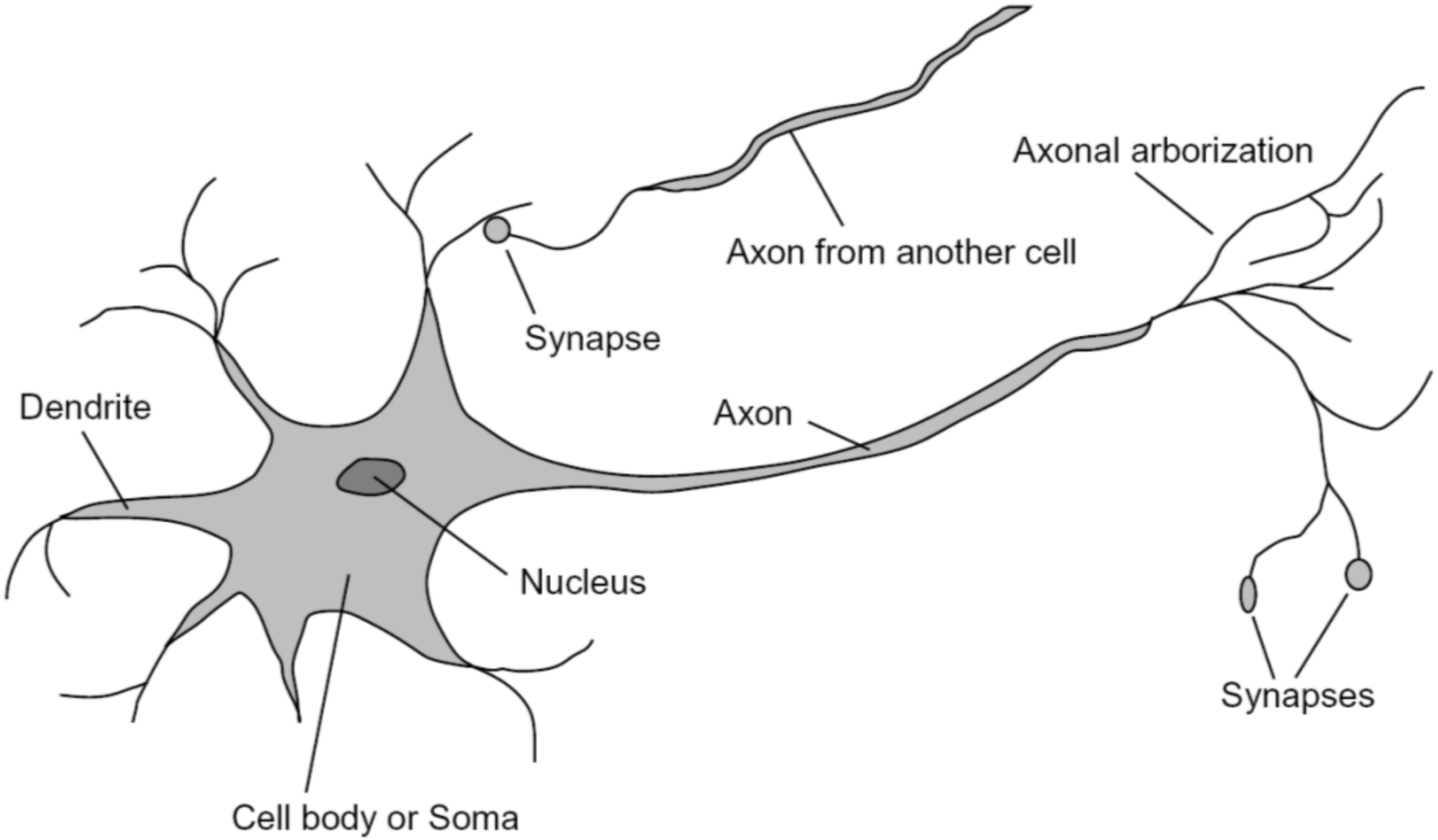
Perceptron



Output: sgn($\mathbf{w}\cdot\mathbf{x}$ + b)

Can incorporate bias as component of the weight vector by always including a feature with value set to 1

Loose inspiration: Human neurons



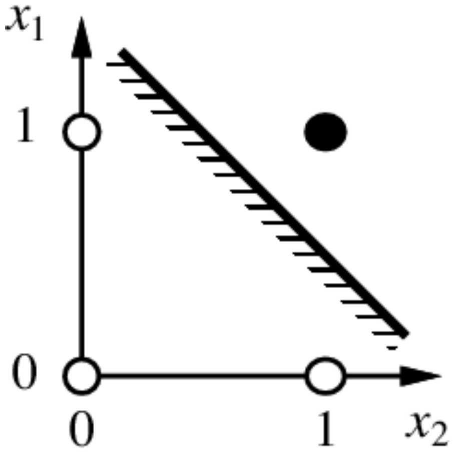
Perceptron training algorithm

- Initialize weights
- Cycle through training examples in multiple passes (*epochs*)
- For each training example:
    - Classify with current weights: $y' = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x})$
    - If classified incorrectly, update weights:

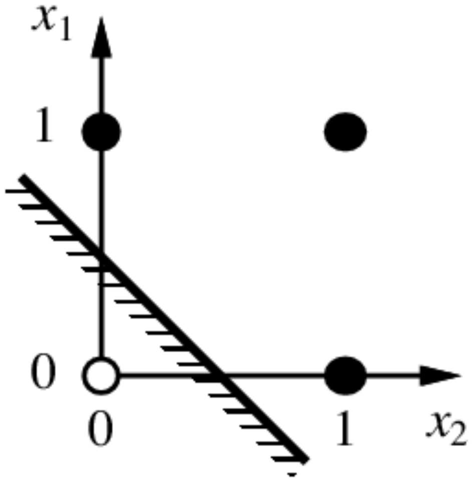
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left(y - y'\right)\mathbf{x}$$

    - α is a *learning rate* that should decay as a function of epoch t, e.g., 1000/(1000+t)

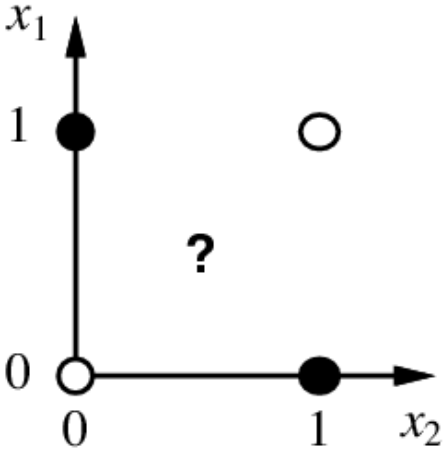
Linear separability



x_1 **and** x_2



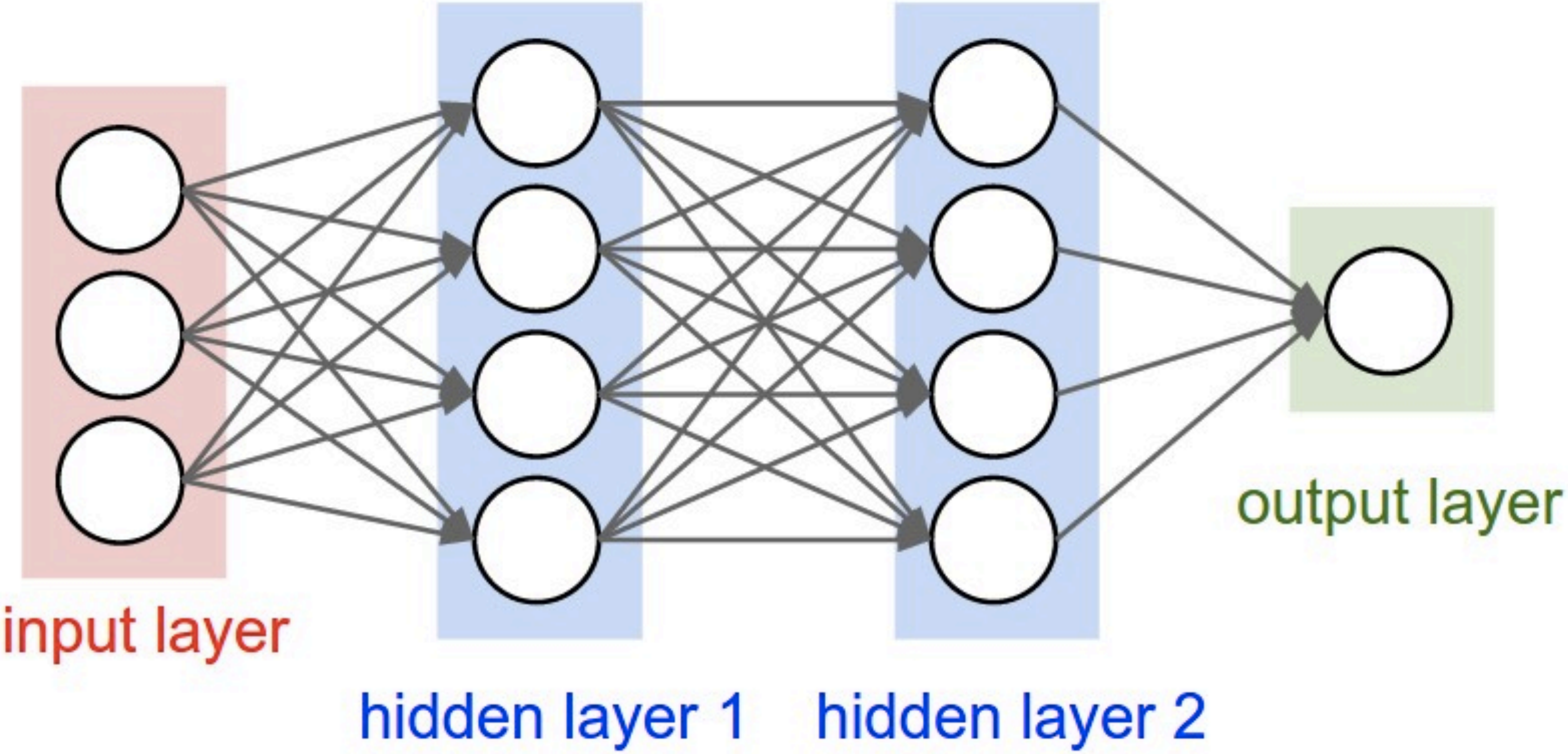
x_1 **or** x_2



x_1 **xor** x_2

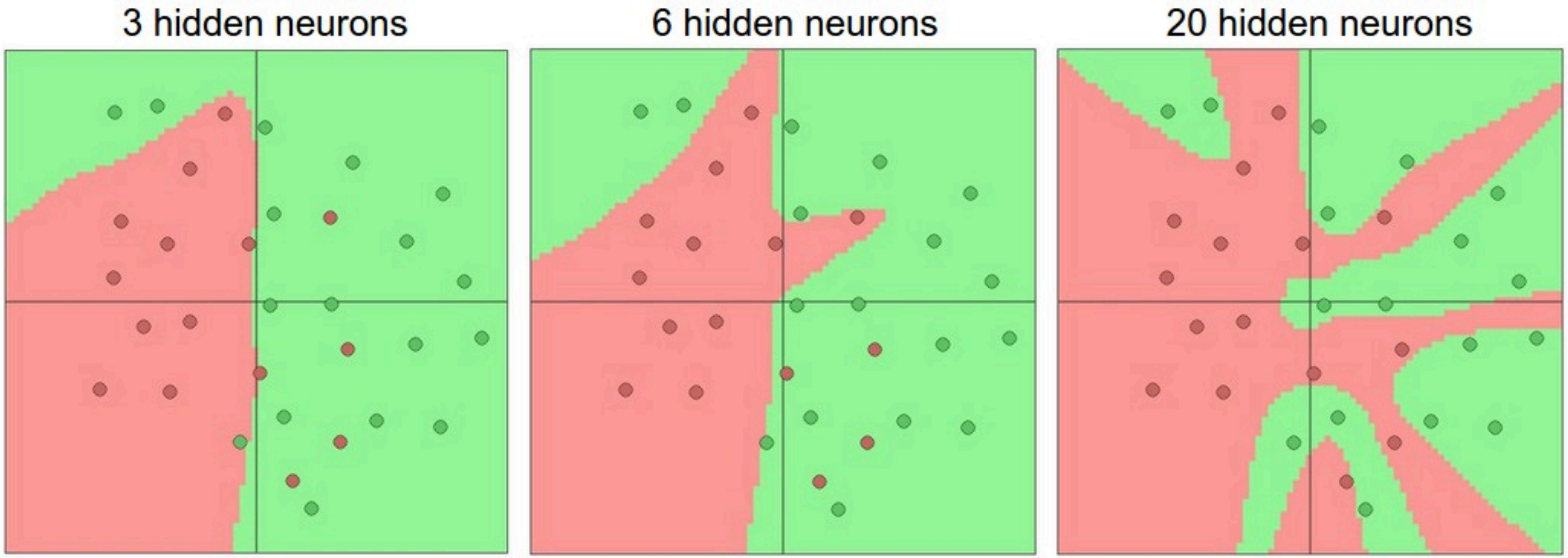
How do we make nonlinear classifiers out of perceptrons?

- Build a multi-layer neural network!



Network with a single hidden layer

- Hidden layer size and *network capacity*:



Source: http://cs231n.github.io/neural-networks-1/

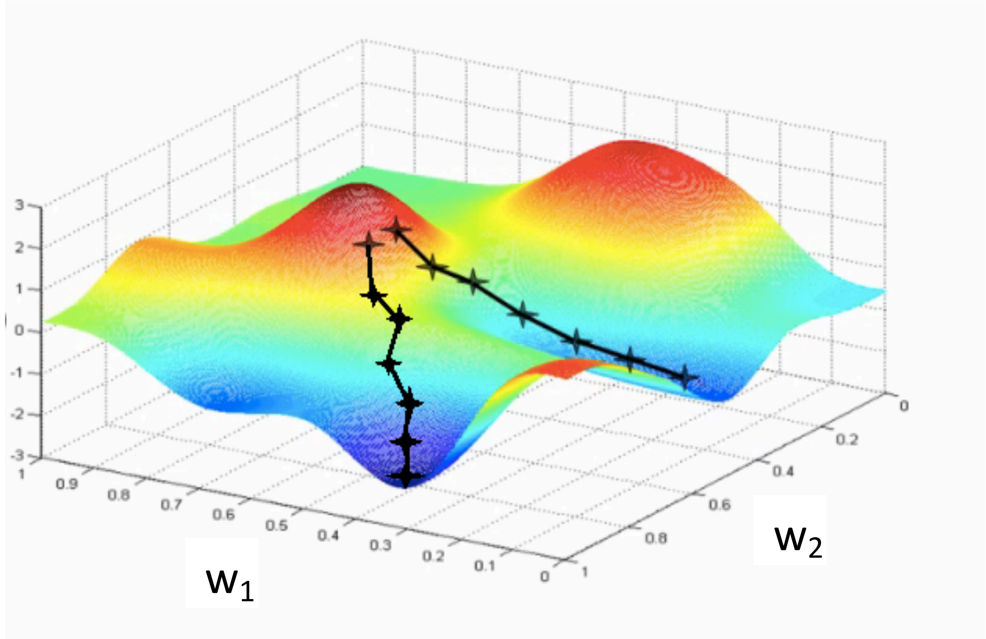
Training of multi-layer networks

- Find network weights to minimize the error between true and estimated labels of training examples:

$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j) \right)^2$$

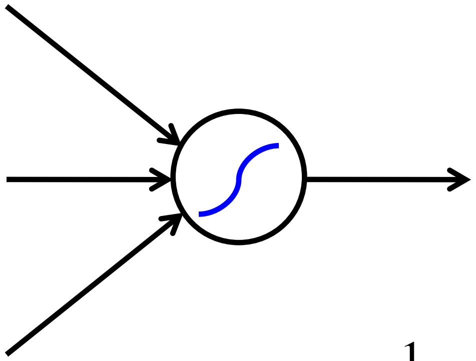
- Update weights by **gradient descent:**

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$$

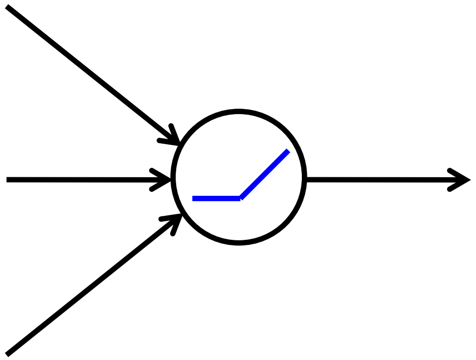


Training of multi-layer networks

- **Gradient descent** requires neural networks to be equipped with a (nearly) differentiable nonlinearity function, called neuron

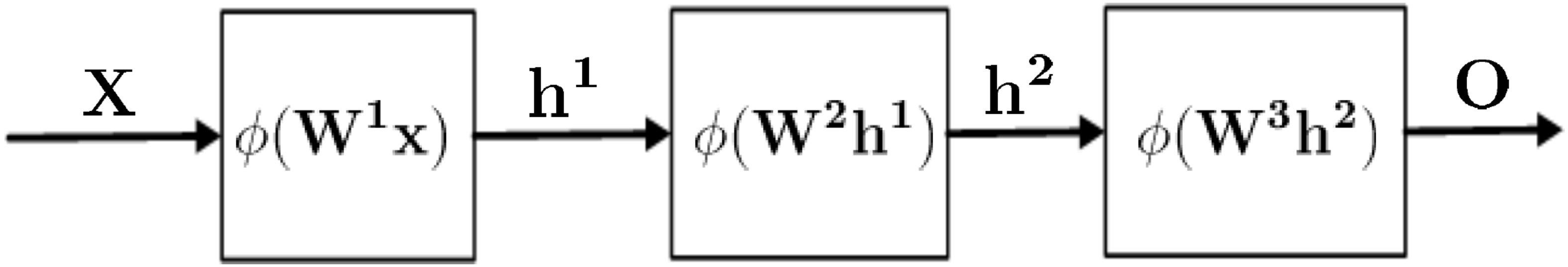


Sigmoid: $g(t) = \frac{1}{1+e^{-t}}$

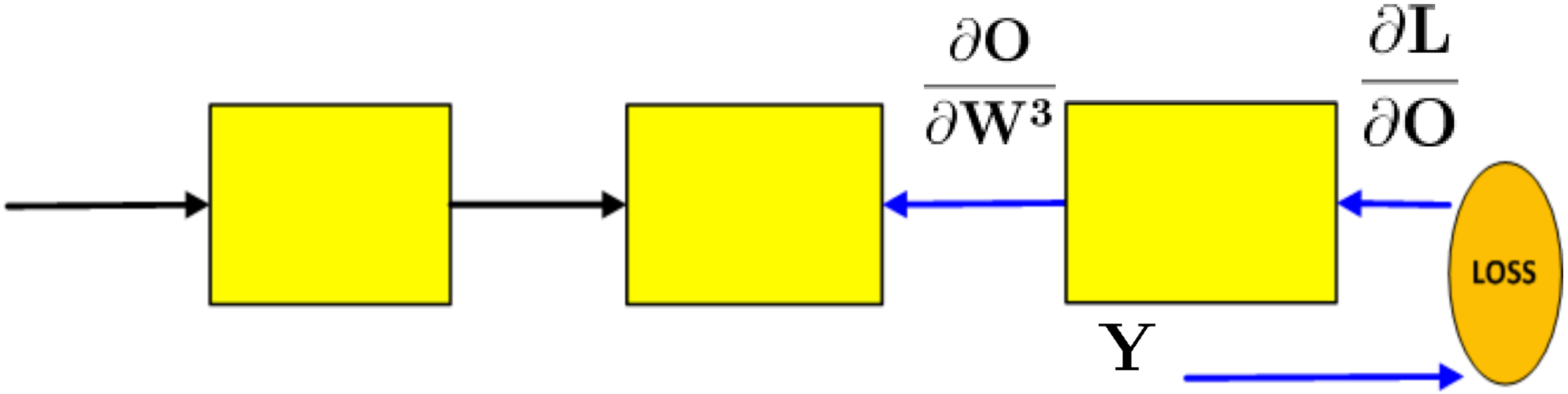


Rectified linear unit (ReLU): $g(t) = \max(0,t)$

Forward-Backward Propagation

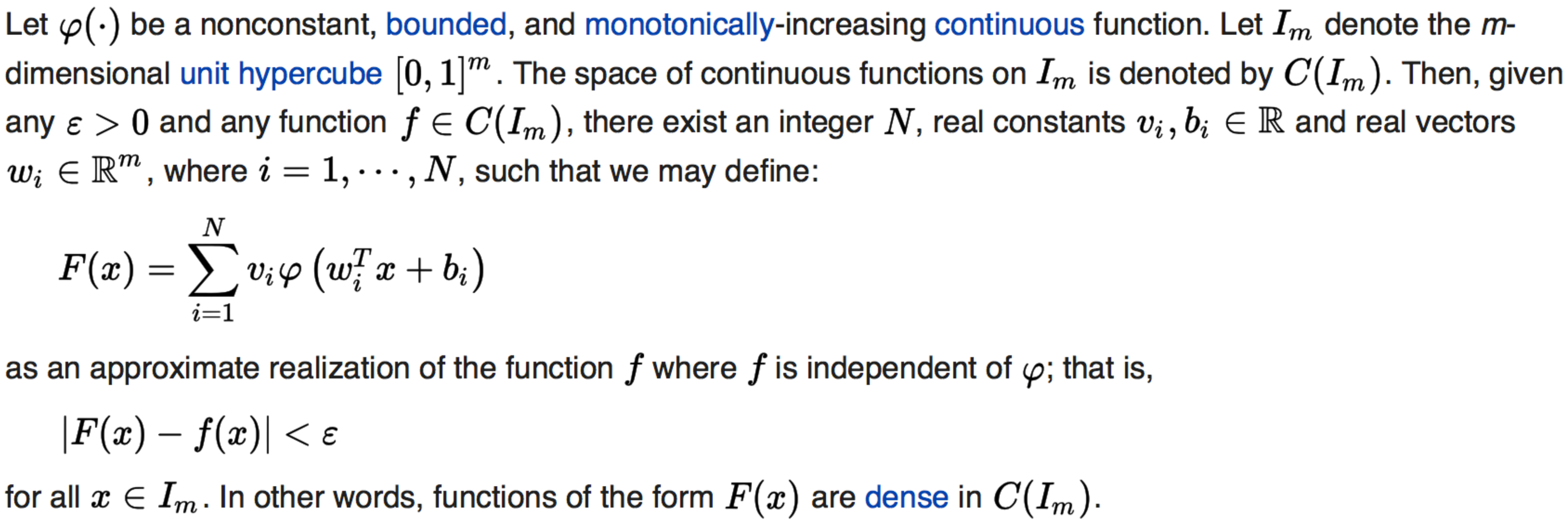


Forward propagation: $h(\mathbf{x}) = \phi(\mathbf{W}\mathbf{x})$



Backward propagation: $\frac{\partial \mathbf{L}}{\partial \mathbf{W}^3} = \frac{\partial \mathbf{L}}{\partial \mathbf{O}} \frac{\partial \mathbf{O}}{\partial \mathbf{W}^3}$ **(Chain Rule)**

NNs are Universal Approximators (in theory)



Let $\varphi(\cdot)$ be a nonconstant, bounded, and monotonically-increasing continuous function. Let I_m denote the *m*-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, there exist an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$, where $i = 1, \cdots, N$, such that we may define:

$$F(x) = \sum_{i=1}^{N} v_i \varphi\left(w_i^T x + b_i\right)$$

as an approximate realization of the function f where f is independent of φ; that is,

$$|F(x) - f(x)| < \varepsilon$$

for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.

- A feed-forward network with **a single hidden layer** containing **a finite number of *nonlinear* neurons**, can approximate **any continuous function on compact subsets of R^n**, under mild assumptions.
- It is not the specific choice of the activation function, but rather the **multilayer feedforward architecture** itself which gives neural networks the potential of being universal approximators.
- It does not touch upon the **algorithmic learnability** of those parameters.

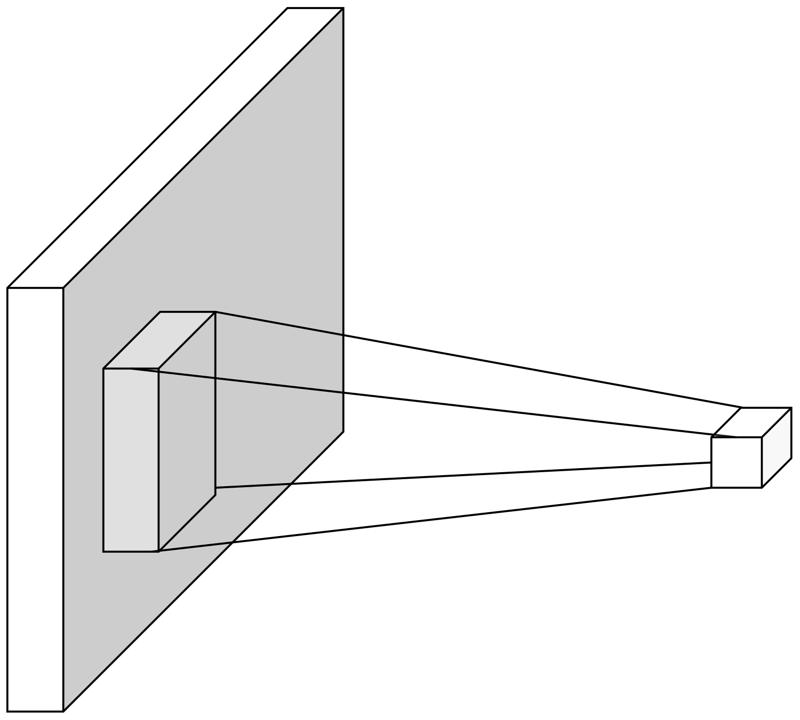
From NNs to Convolution NNs

The most important building block in modern deep learning

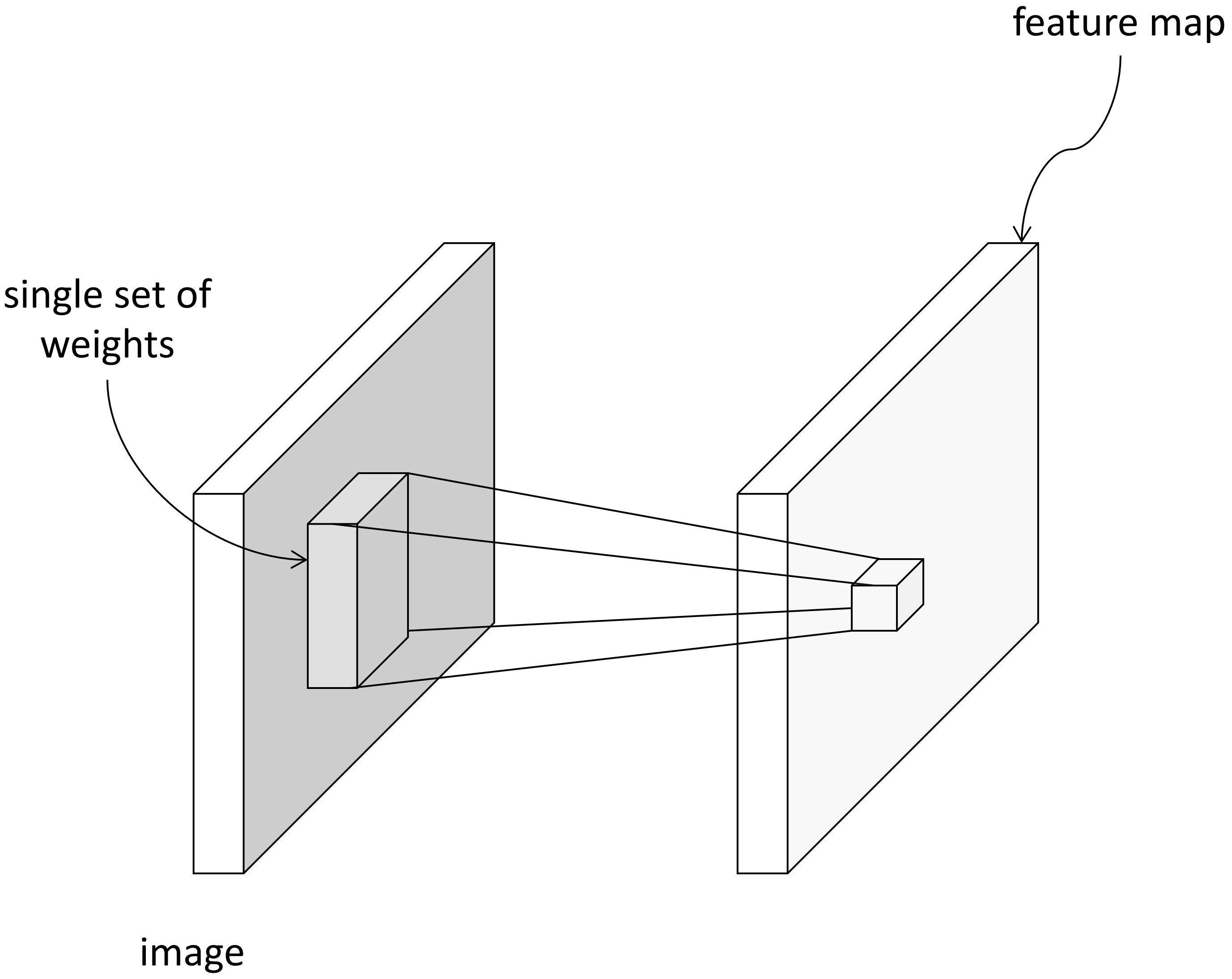
Review: Spatial Low-Level Vision

- **Critical Building Block:** Filters and filter banks
  - Often implemented via **convolution (yes, that guy in deep learning)**
  - Detection of edges, corners, and other local features
  - Texture pattern recognition and synthesis
  - Can include multiple orientations
  - Can include **multiple scales:** “filter pyramids” – a key feature in deep learning

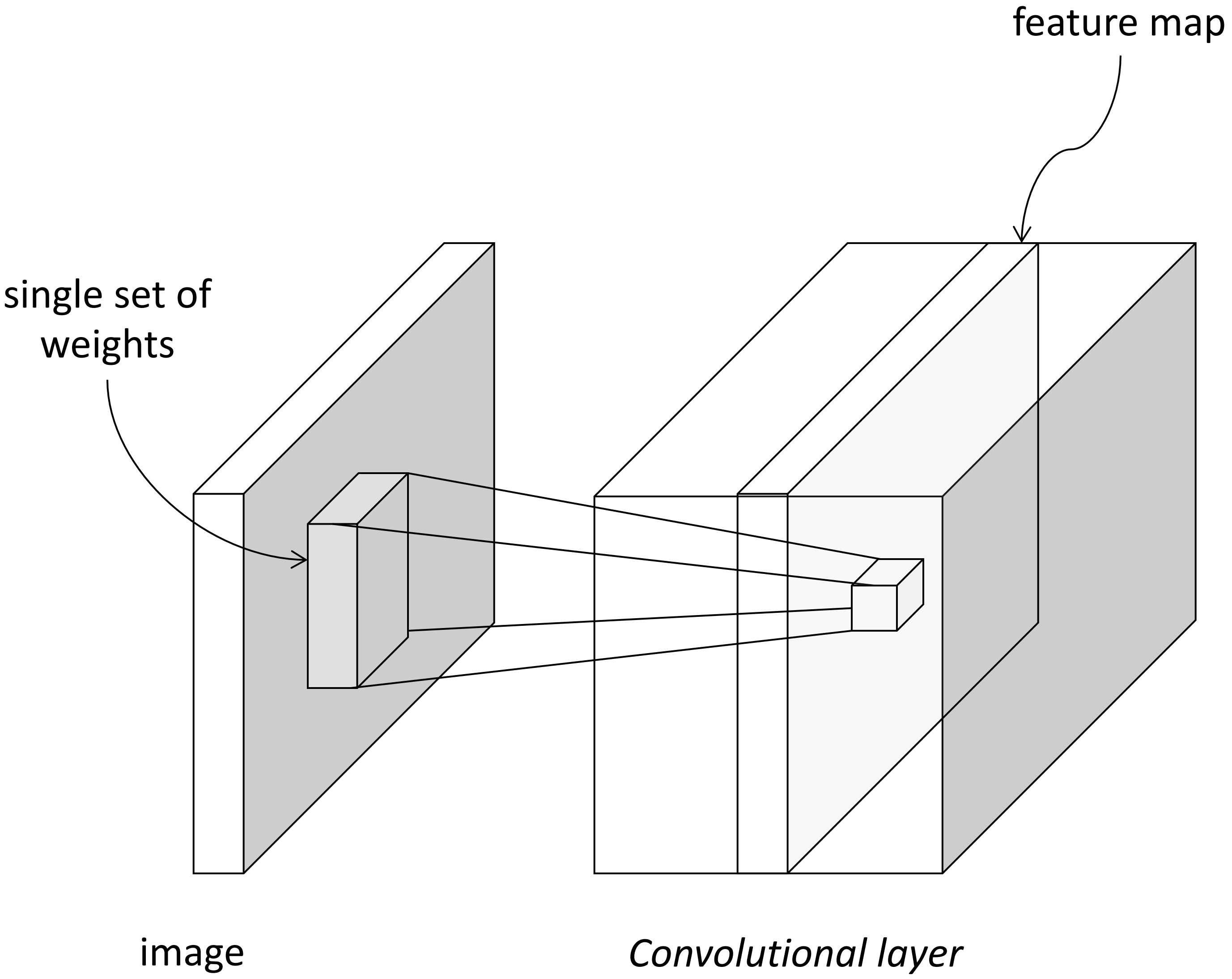
From fully connected to convolutional networks



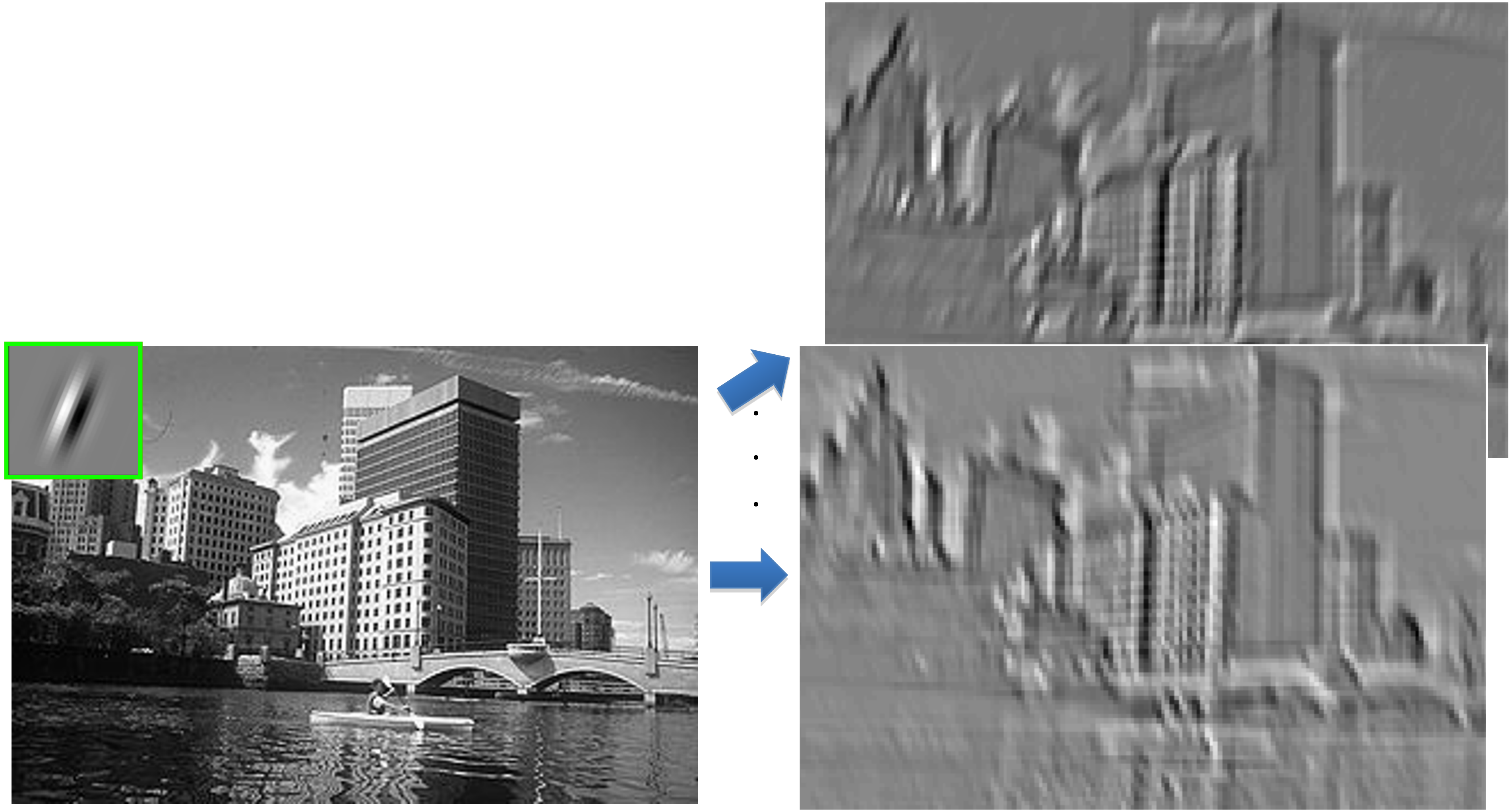
From fully connected to convolutional networks



From fully connected to convolutional networks



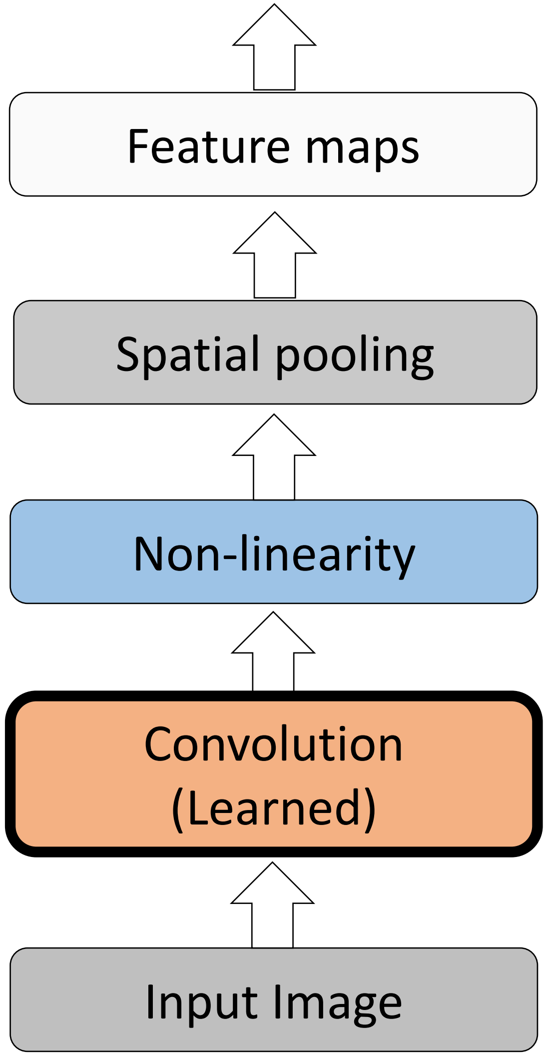
Convolution as feature extraction



Input

Feature Map

Key operations in a CNN



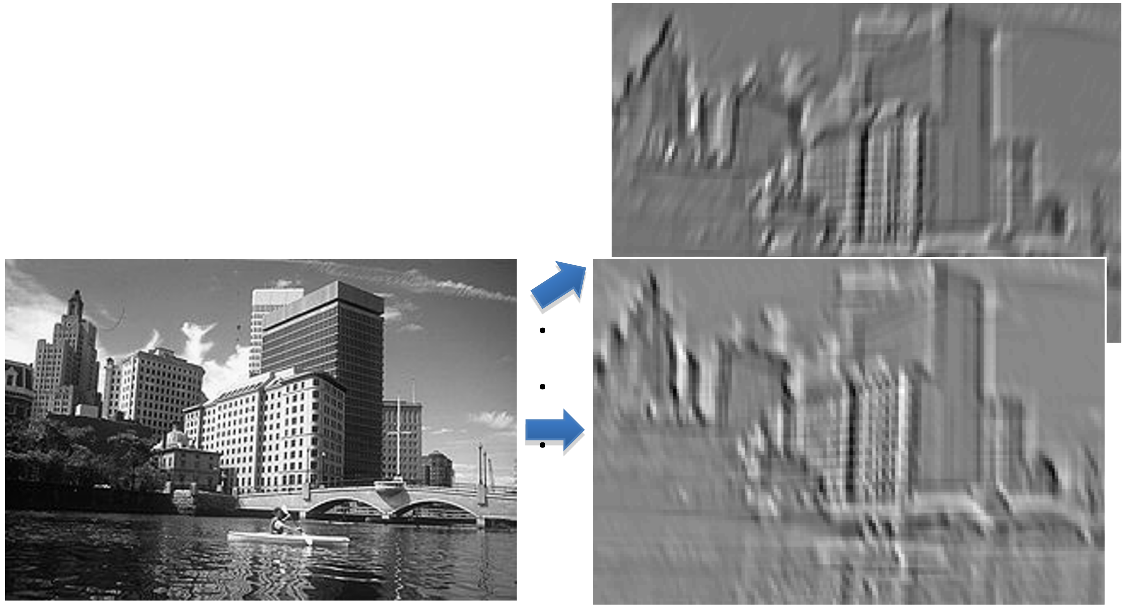
Feature maps

Spatial pooling

Non-linearity

Convolution (Learned)

Input Image



Input

Feature Map

Source: R. Fergus, Y. LeCun

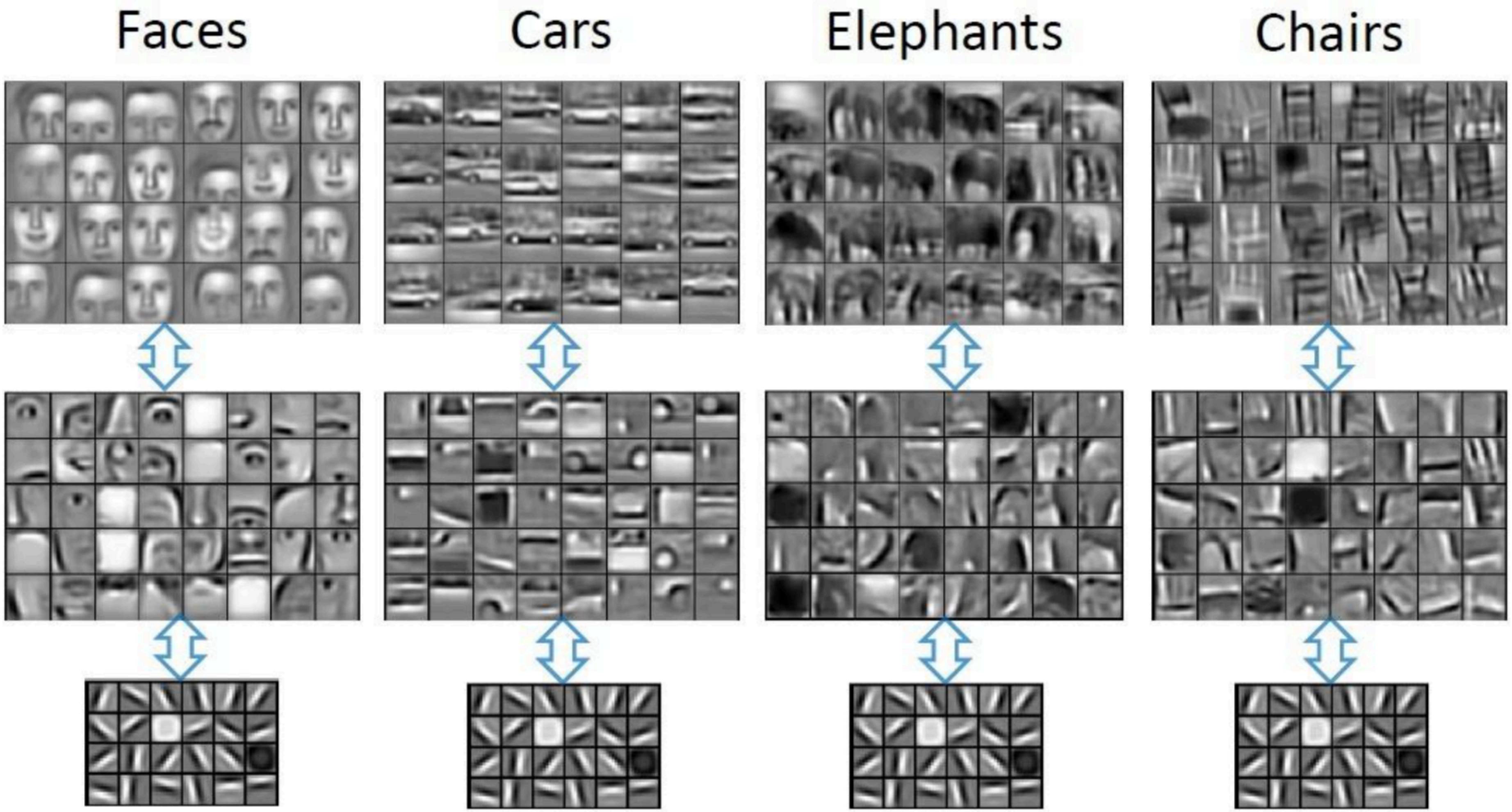
Review: Computer Vision Has "Three Levels"

"There's an edge!"

"There's an object and a background!"

"There's a chair!"

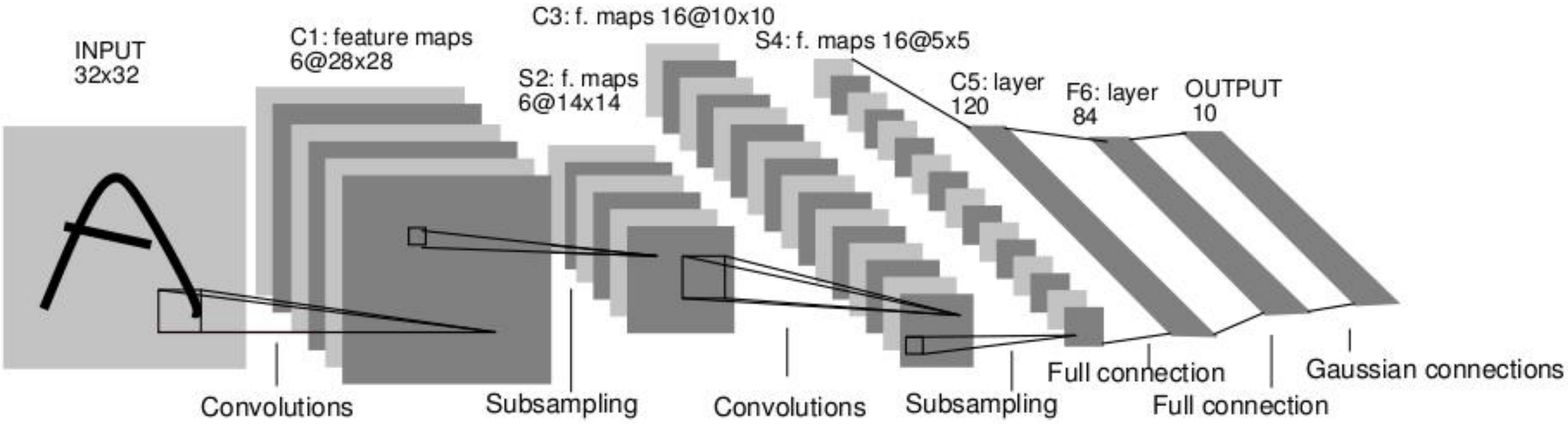
Deep Features (May) Learn Semantic Hierarchy



Popular Backbones: From LeNet to DenseNet

A Remarkable Odyssey to Artificial Intelligence by Human Intelligence

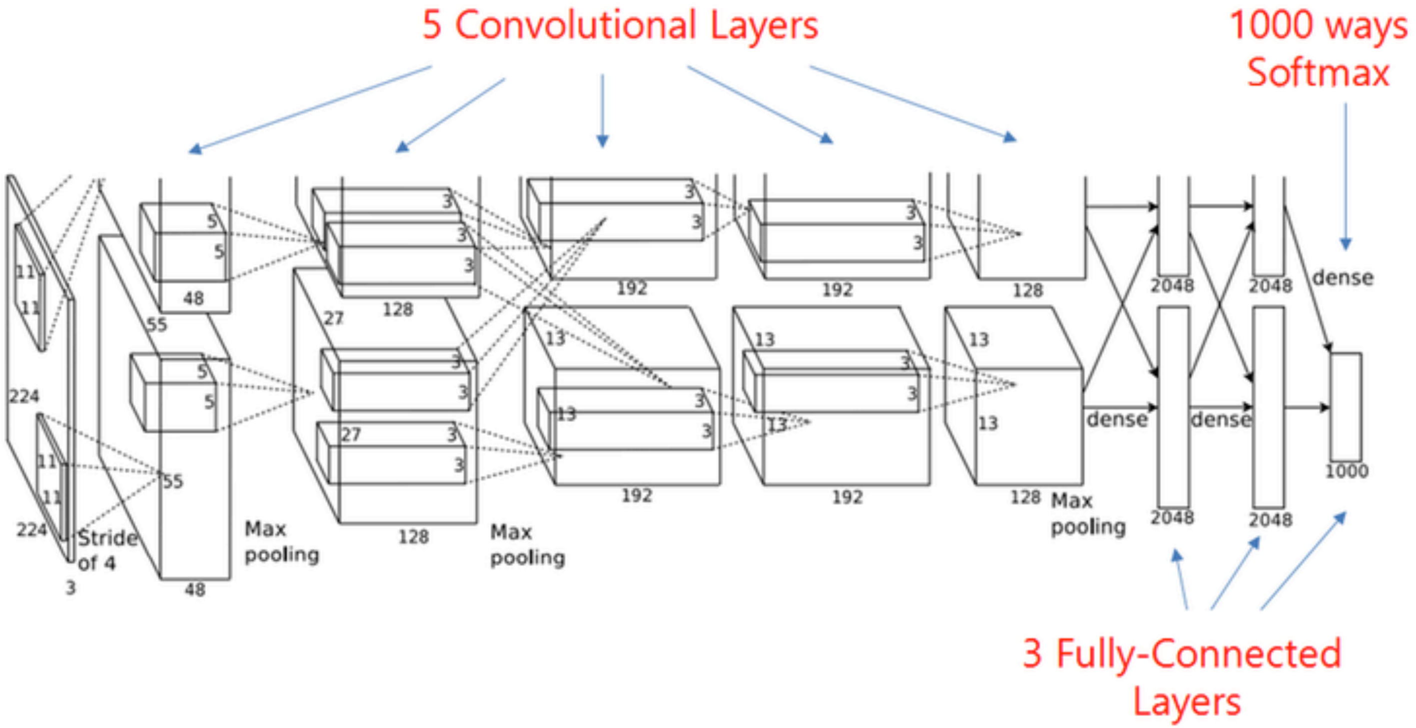
LeNet-5



- Average pooling
- Sigmoid or tanh nonlinearity
- Fully connected layers at the end
- Trained on MNIST digit dataset with 60K training examples

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition,
Proc. IEEE 86(11): 2278–2324, 1998.

AlexNet, 2012



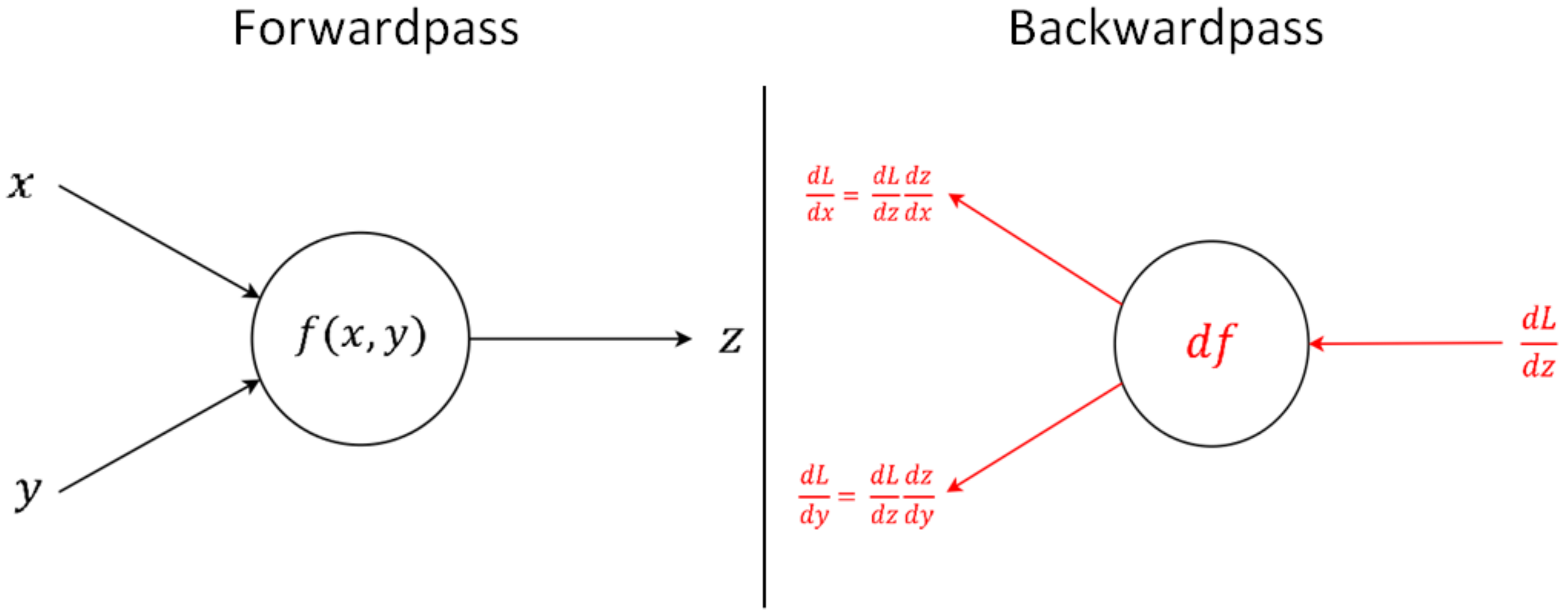
5 Convolutional Layers

1000 ways Softmax

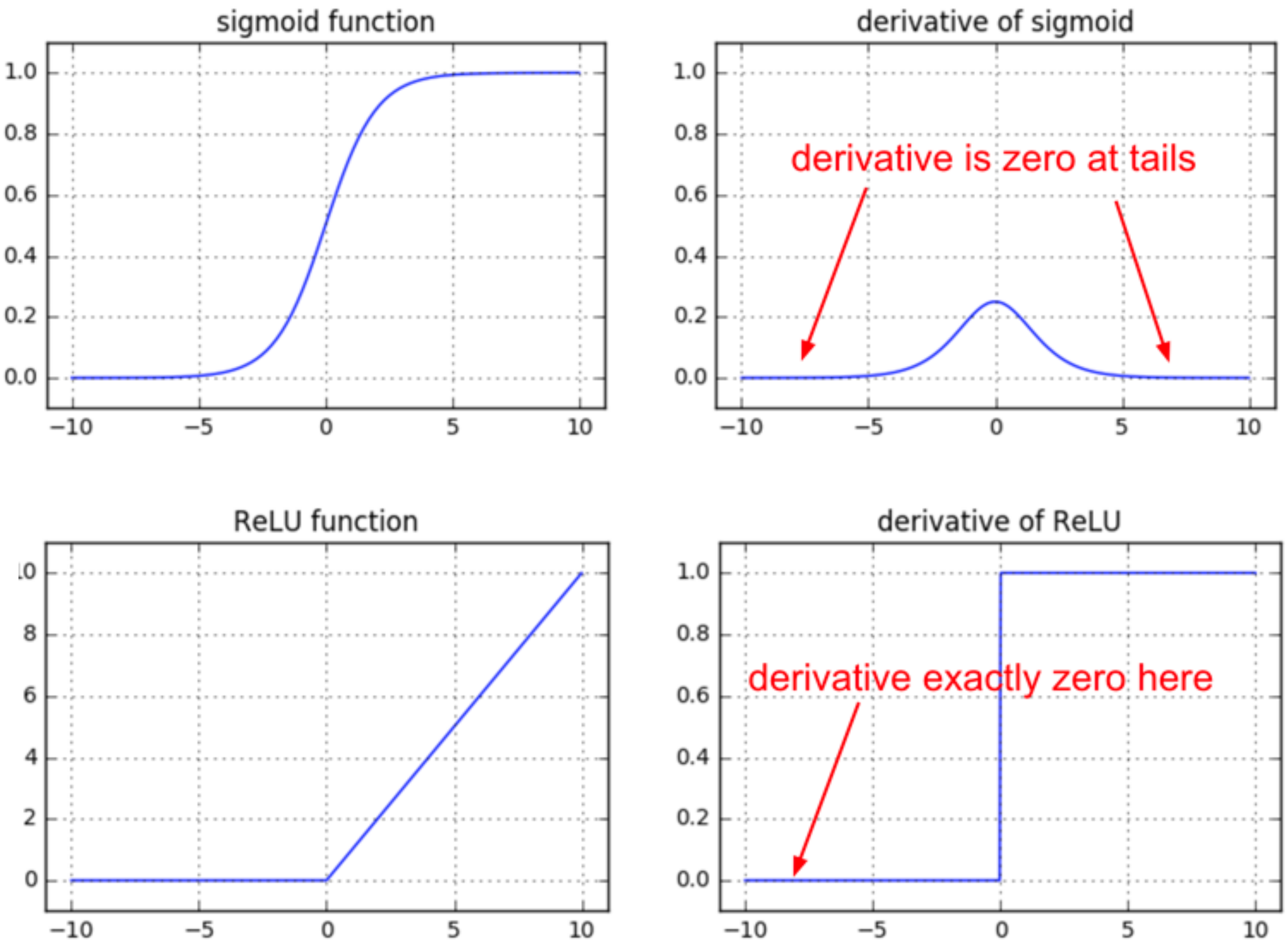
3 Fully-Connected Layers

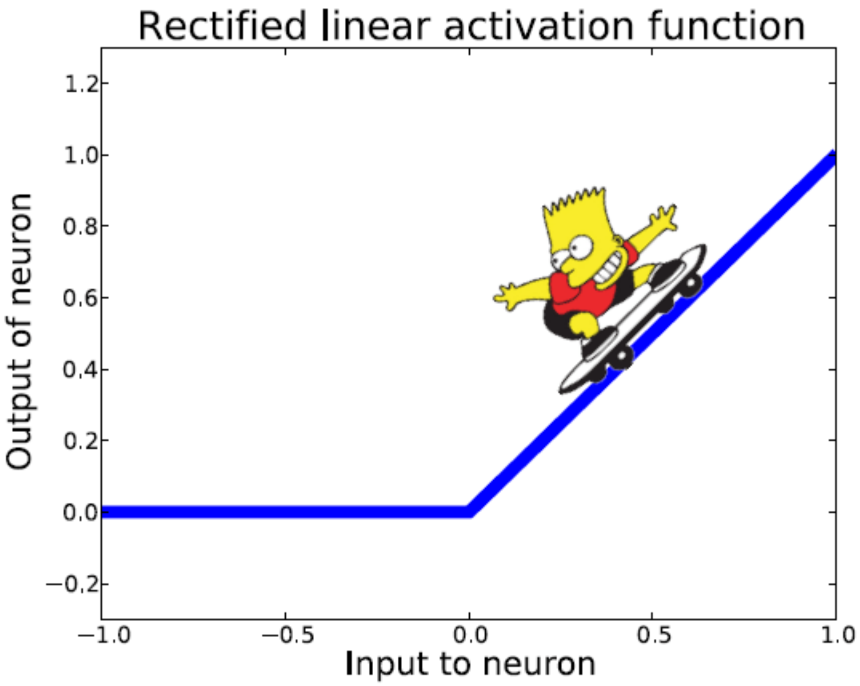
- The **FIRST** winner deep model in computer vision, and one of the most classical choices for domain experts to adapt for their applications
- 5 convolutional layers + 3 fully-connected layers + softmax classifier
- Three Key Design Features: **ReLU, dropout, data augmentation**

Recap: “Chain Rule”



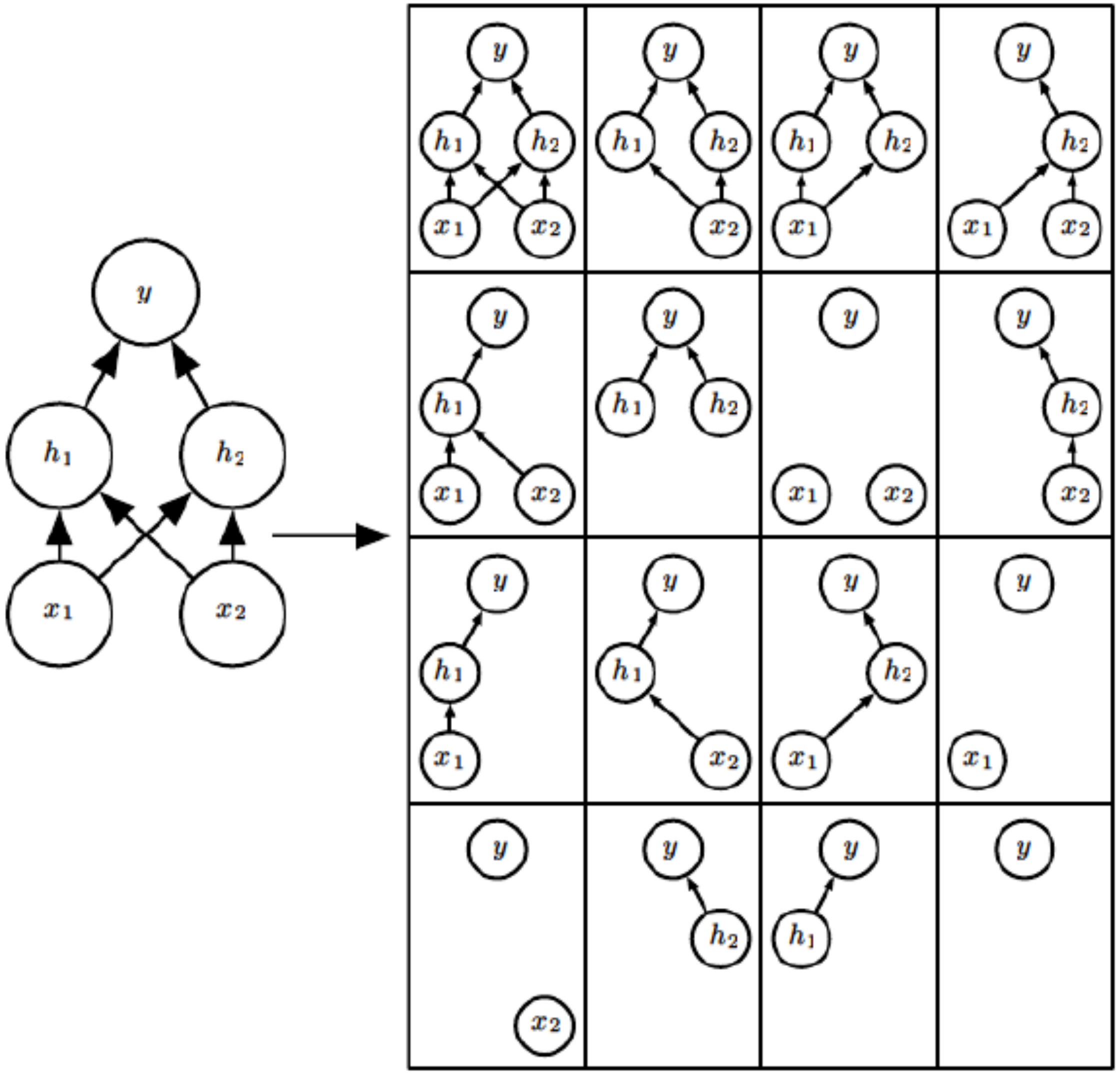
From Sigmoid to ReLU





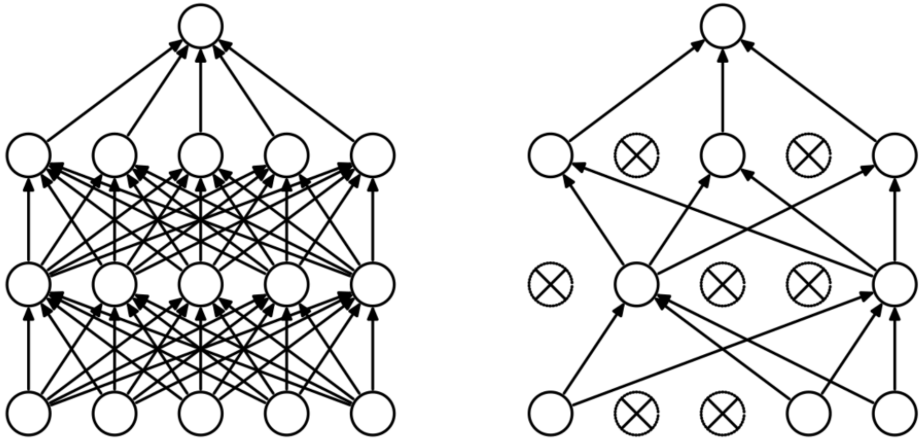
Dropout

- Randomly select weights to update
  - In each update step, randomly sample a different binary mask to all the input and hidden units
  - Multiple the mask bits with the units and do the update as usual
  - Typical dropout probability: 0.2 for input and 0.5 for hidden units
  - Very useful for FC layers, less for conv layers, not useful in RNNs



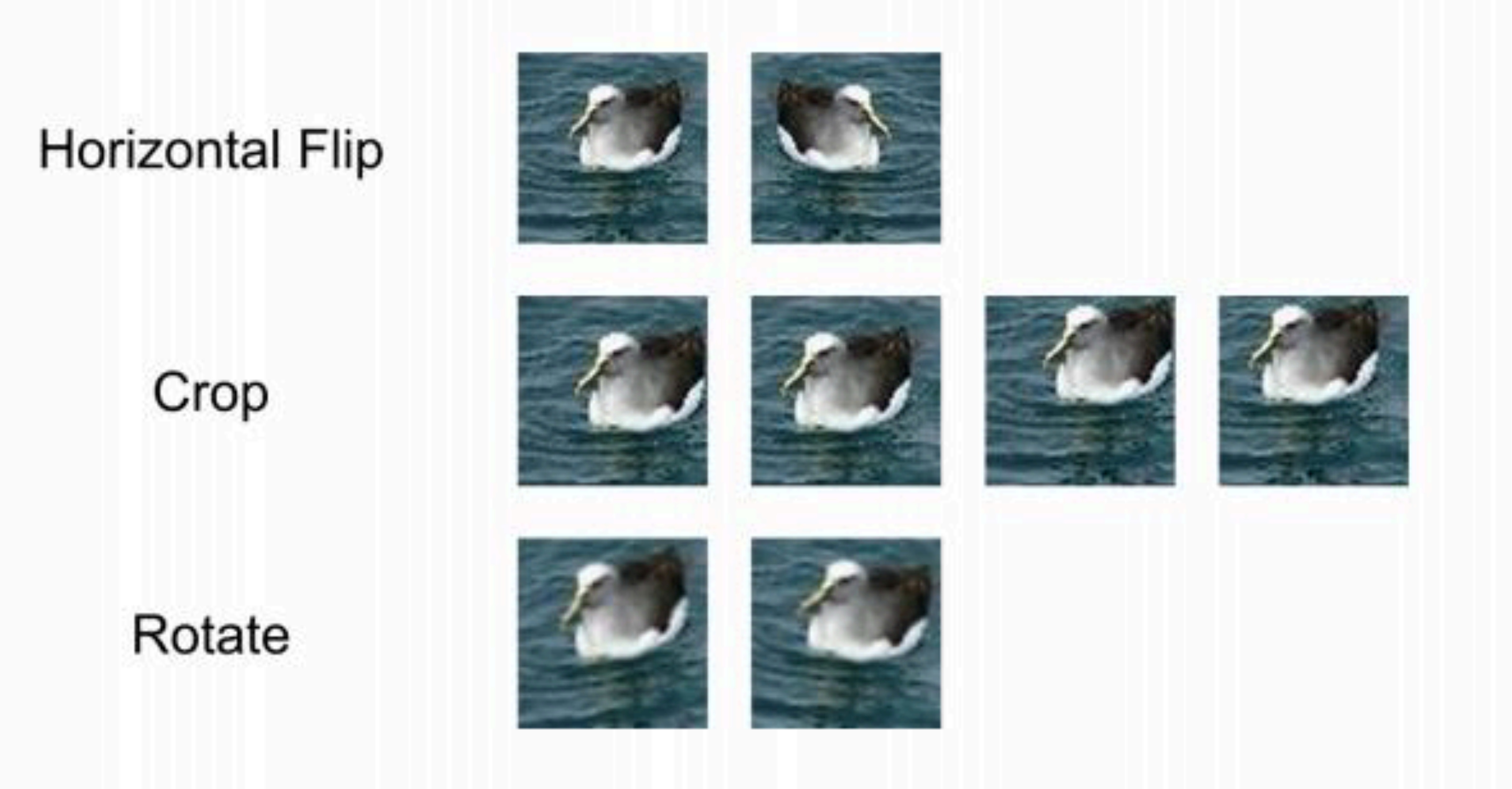
Dropout: A Stochastic Ensemble

- ***Dropout***: a feature-based bagging
  - Resamples input as well as ***latent*** features
  - With ***parameter sharing*** among voters



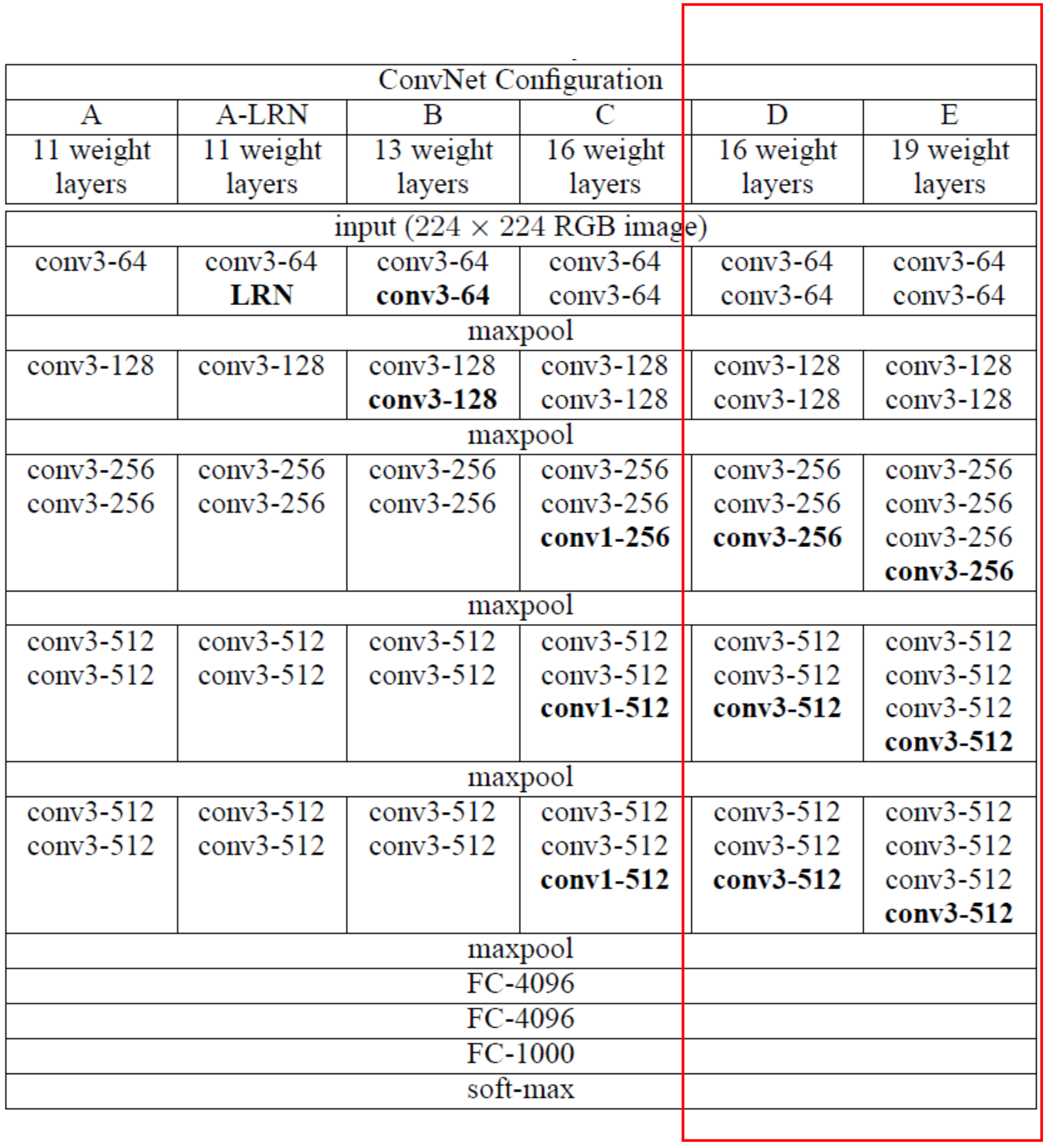
- SGD training: each time loading a minibatch, randomly sample a binary mask to apply to all input and hidden units
  - Each unit has probability α to be included (a hyperparameter)
  - Typically, 0.8 for input units and 0.5 for hidden units
- Different minibatches are used to train different parts of the NN
  - Similar to bagging, but much more efficient
  - No need to retrain unmasked units
  - Exponential number of voters

Data Augmentation



- Adding noise to the input: a special kind of augmentation
- Be careful about the transformation applied -> label preserving
    - **Example:** classifying ‘b’ and ‘d’; ‘6’ and ‘9’

VGG-Net, 2014



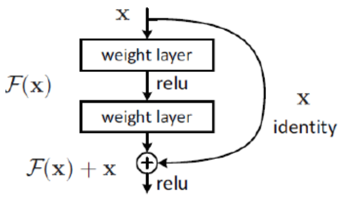
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Key Technical Features:

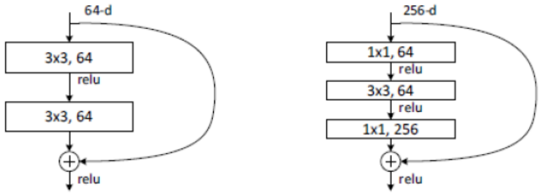
- Increase depth (up to 19)
- Smaller filter size (3)

Configurations D and E are widely used for various tasks, called *VGG-16* and *VGG-19*

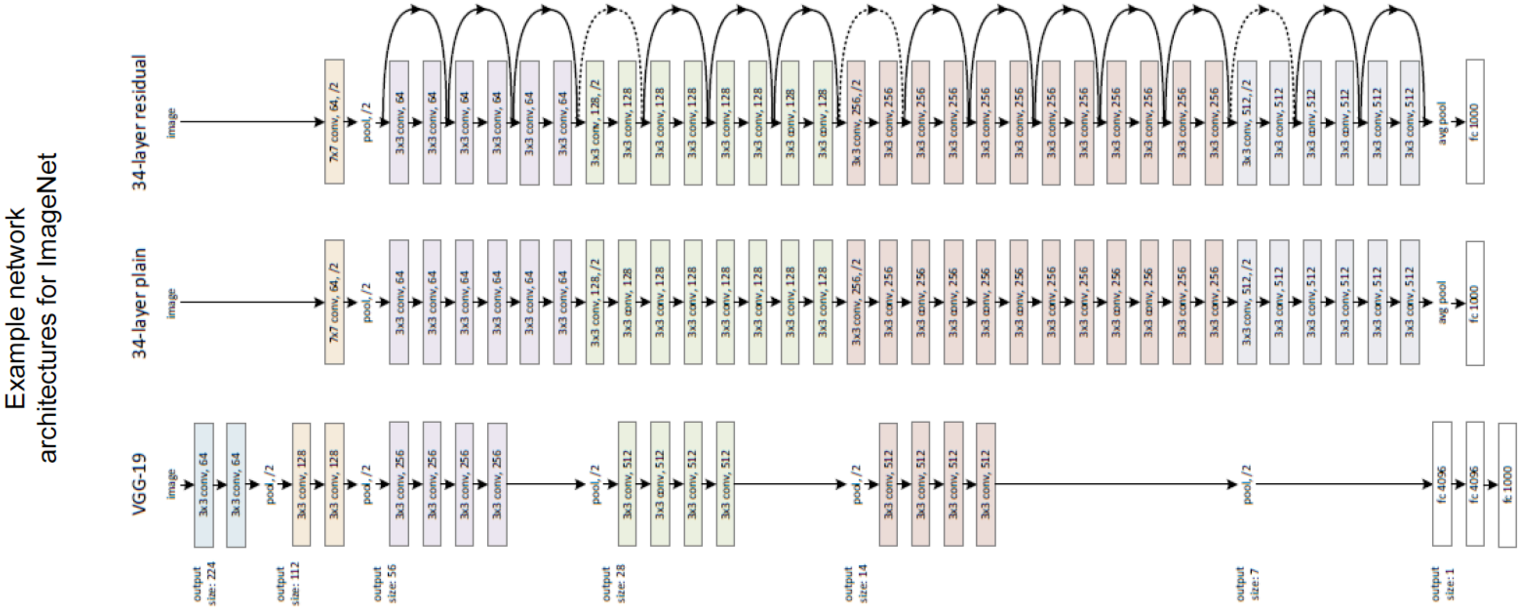
Deep Residual Network (ResNet), 2015



Residual learning: a building block

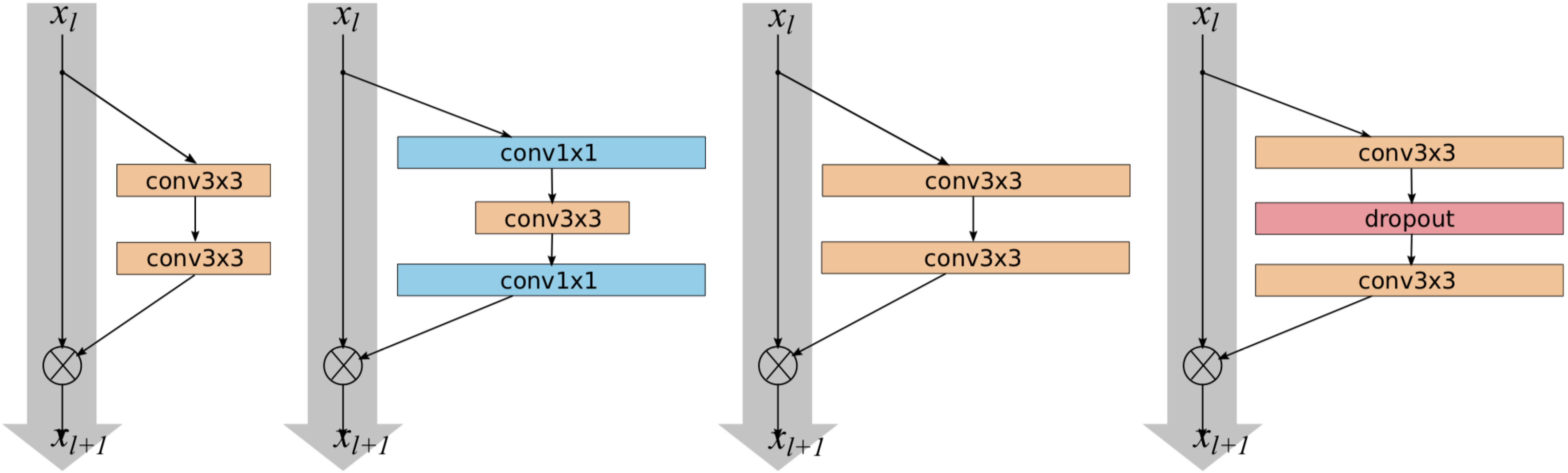


A deeper residual function F for ImageNet



Key Technical Features: skip connections for residual mapping, up to > 1000 layers

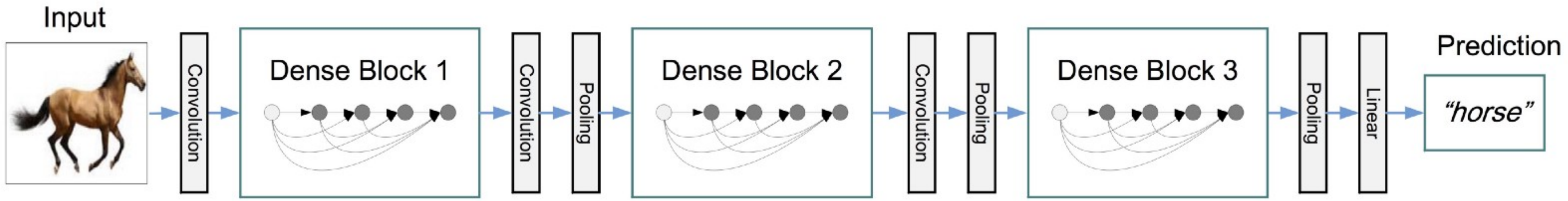
Wide ResNet, 2016

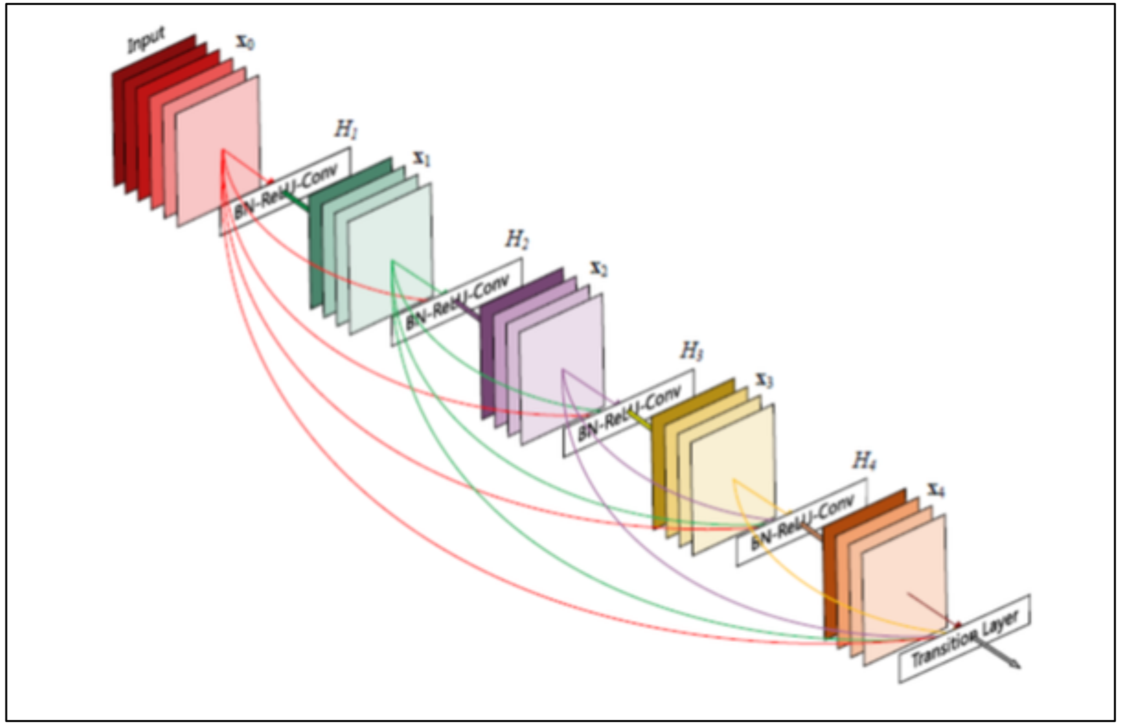


(a) basic (b) bottleneck (c) basic-wide (d) wide-dropout

- Widening of ResNet blocks (if done properly) provides a more effective way of improving performance of residual networks compared to increasing their depth.
- A wide 16-layer deep network has the same accuracy as a 1000-layer thin deep network and a comparable number of parameters, although being several times faster to train.

Densely Connected Convolutional Networks (DenseNet), 2017



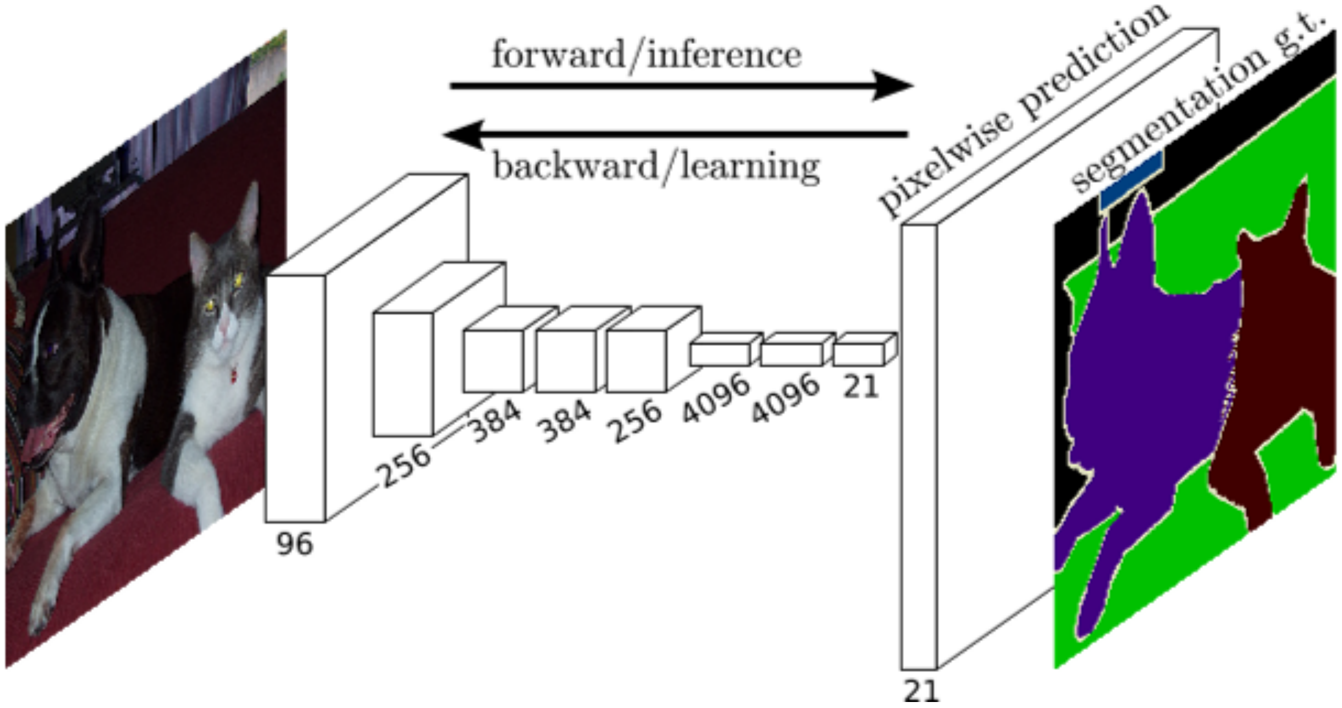


Key Technical Features:

- Finer combination of multi-scale features (or whatever…)

(More) Art of Convolutions

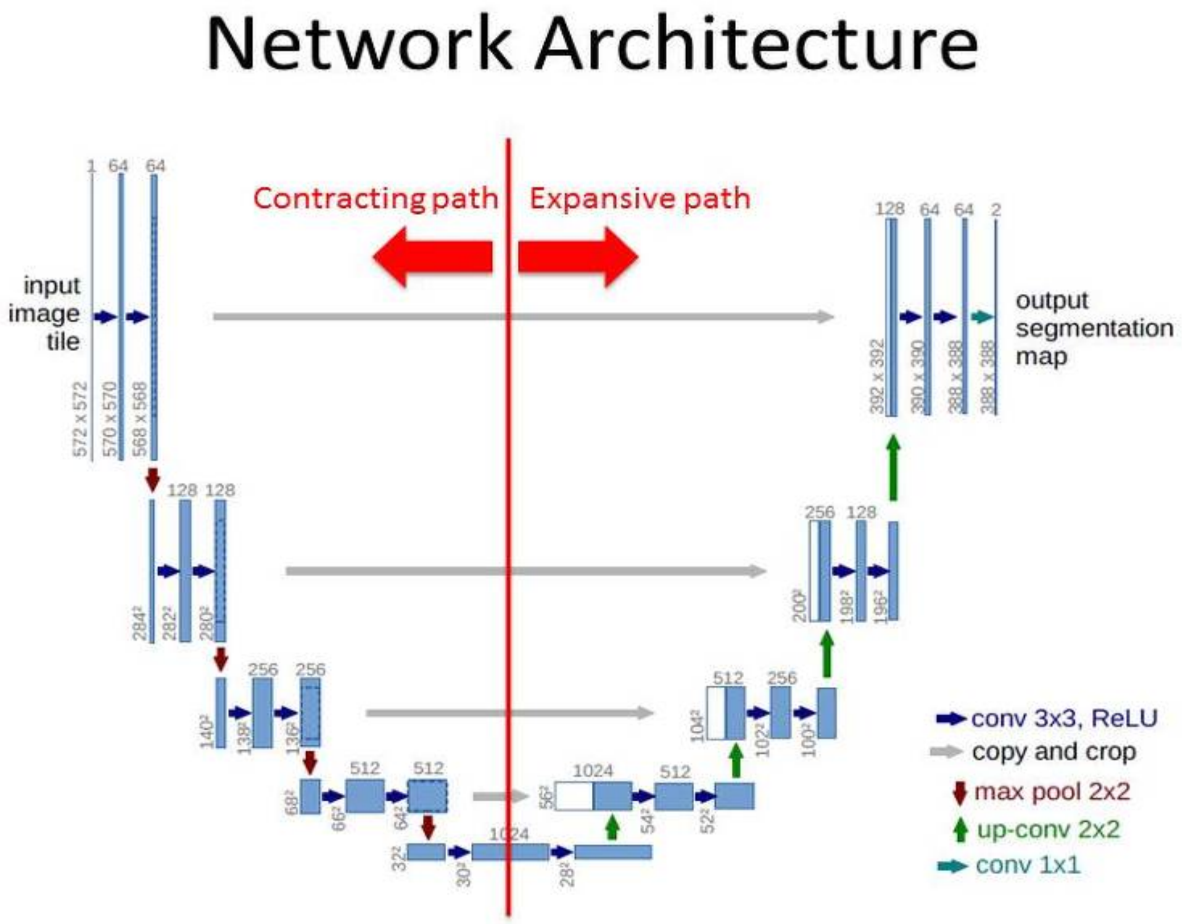
Fully Convolutional Network (FCN), 2014



Key Technical Features:

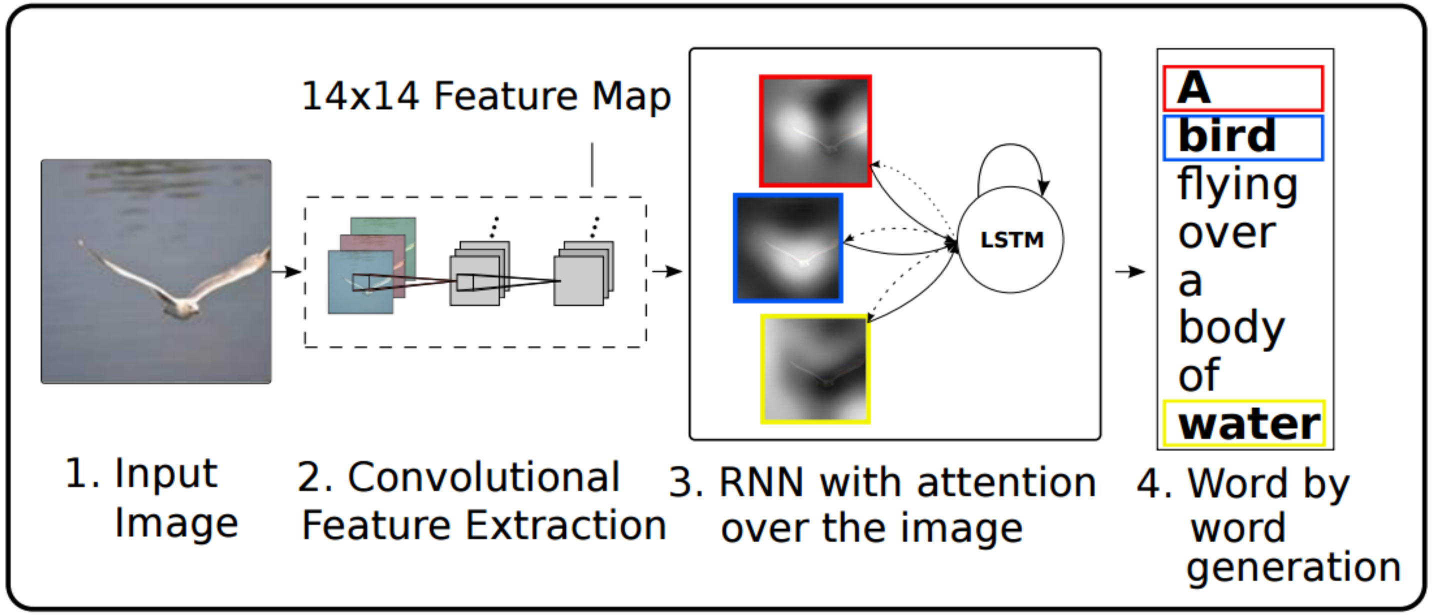
- No fully-connected layer **->** No fixed requirement on input size
- Widely adopted in pixel-to-pixel prediction tasks, e.g., image segmentation

U-Net, 2015



- The architecture consists of a contracting path to capture context
- …and a symmetric expanding path to enable precise localization.
- Also fully convolutional
- Very popular backbone for dense prediction (image segmentation, restoration…)

Attention Mechanism



"Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", 2015

- **Idea is simple:** add a (learned) weighted mask to feature (feature selection)
- Use a feed-forward deep network to extract L feature vectors
- Can use a recurrent network to iteratively update the attention (shown as bright regions) for each output word
- Find meaningful correspondences between words and attentions

Examples of (Input) Visual Attention

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

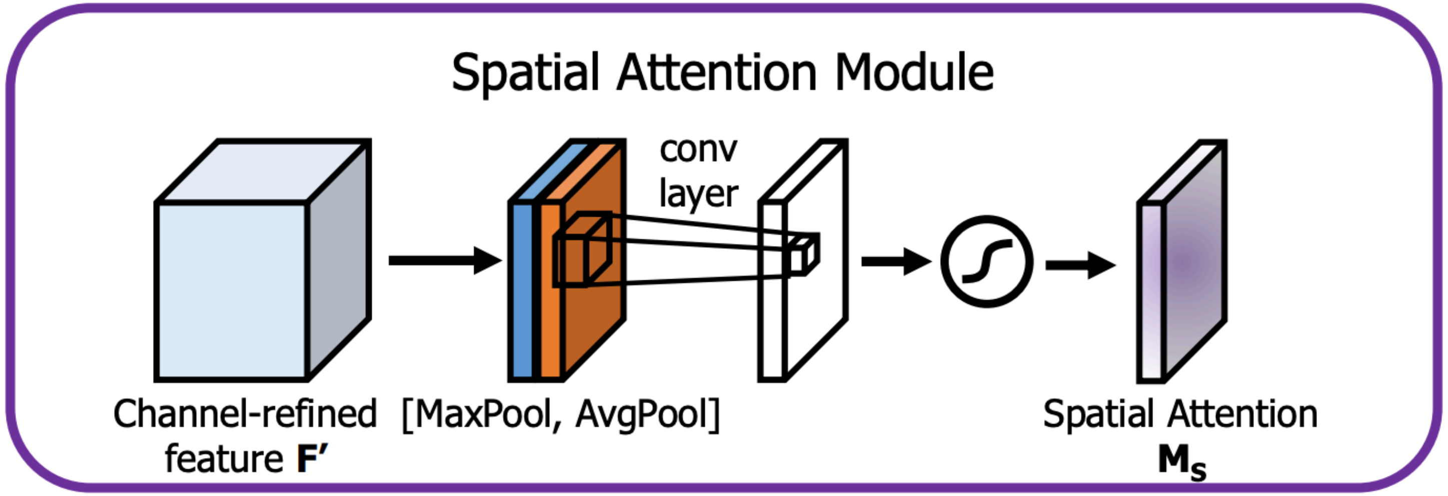
A stop sign is on a road with a mountain in the background.

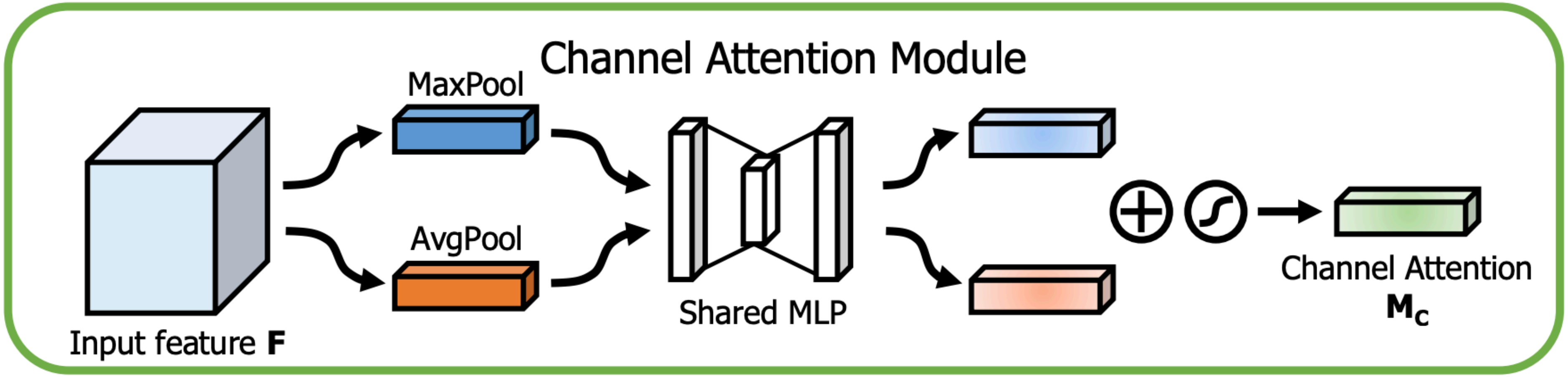
A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

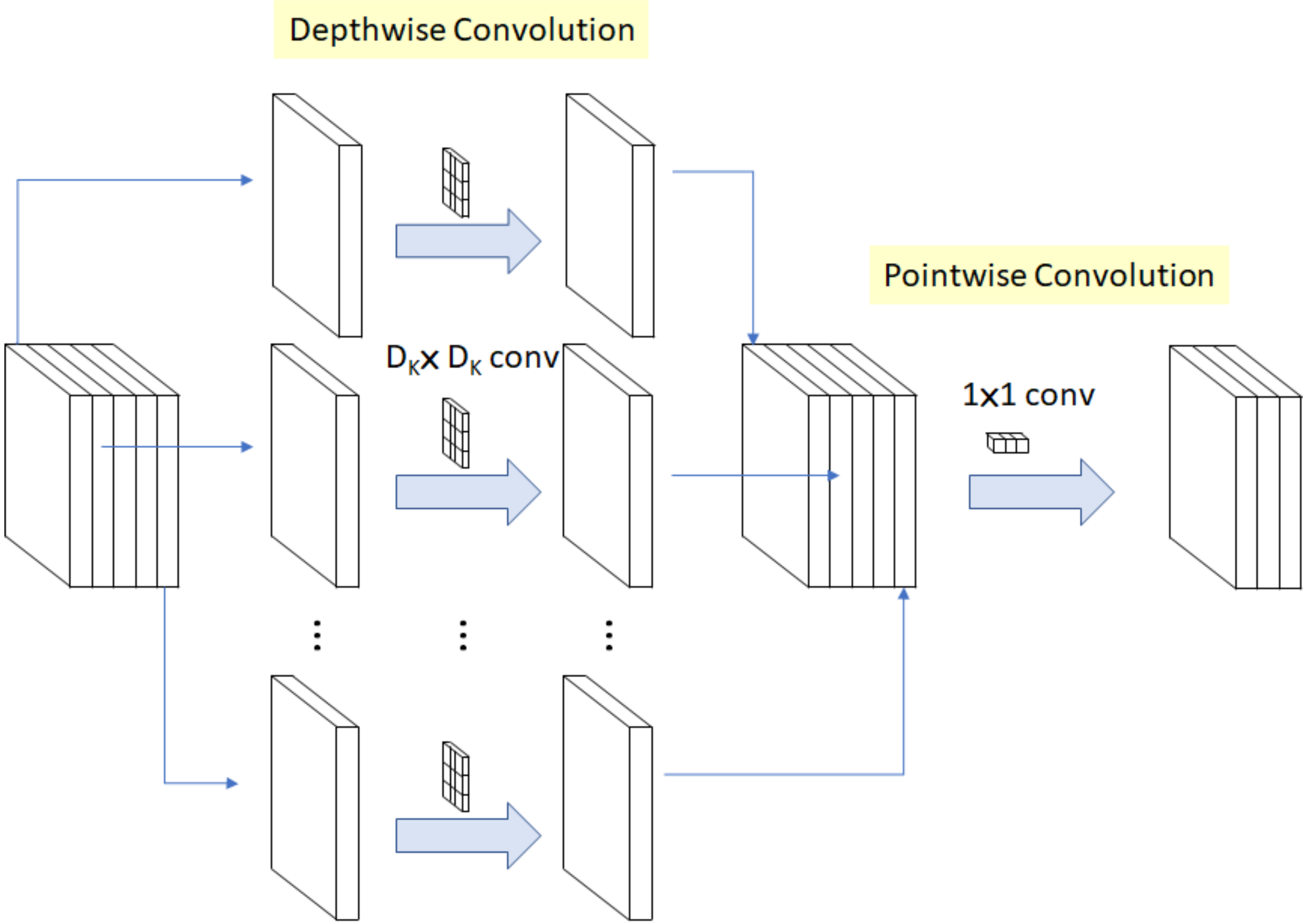
Spatial and Channel Attention





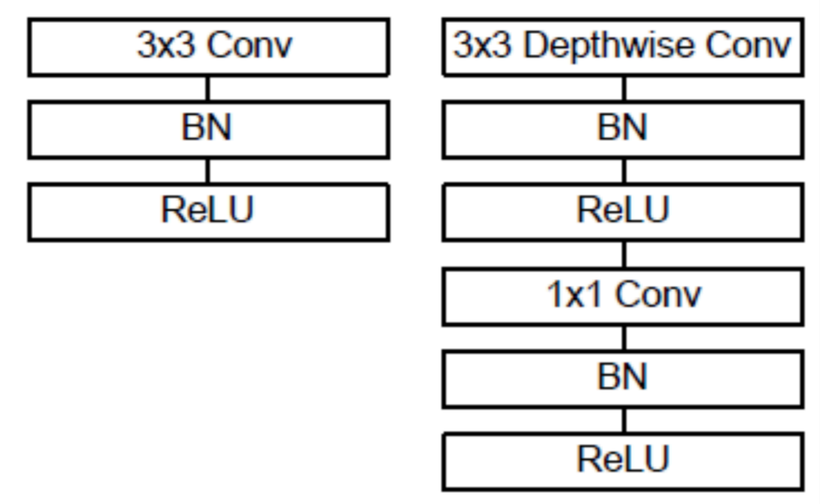
Depth-Wise Convolution

- **Depthwise convolution** is the channel-wise spatial convolution.
- It is often used together with **pointwise convolution,** i.e., 1×1 convolution to change the channel dimension (number of feature maps)



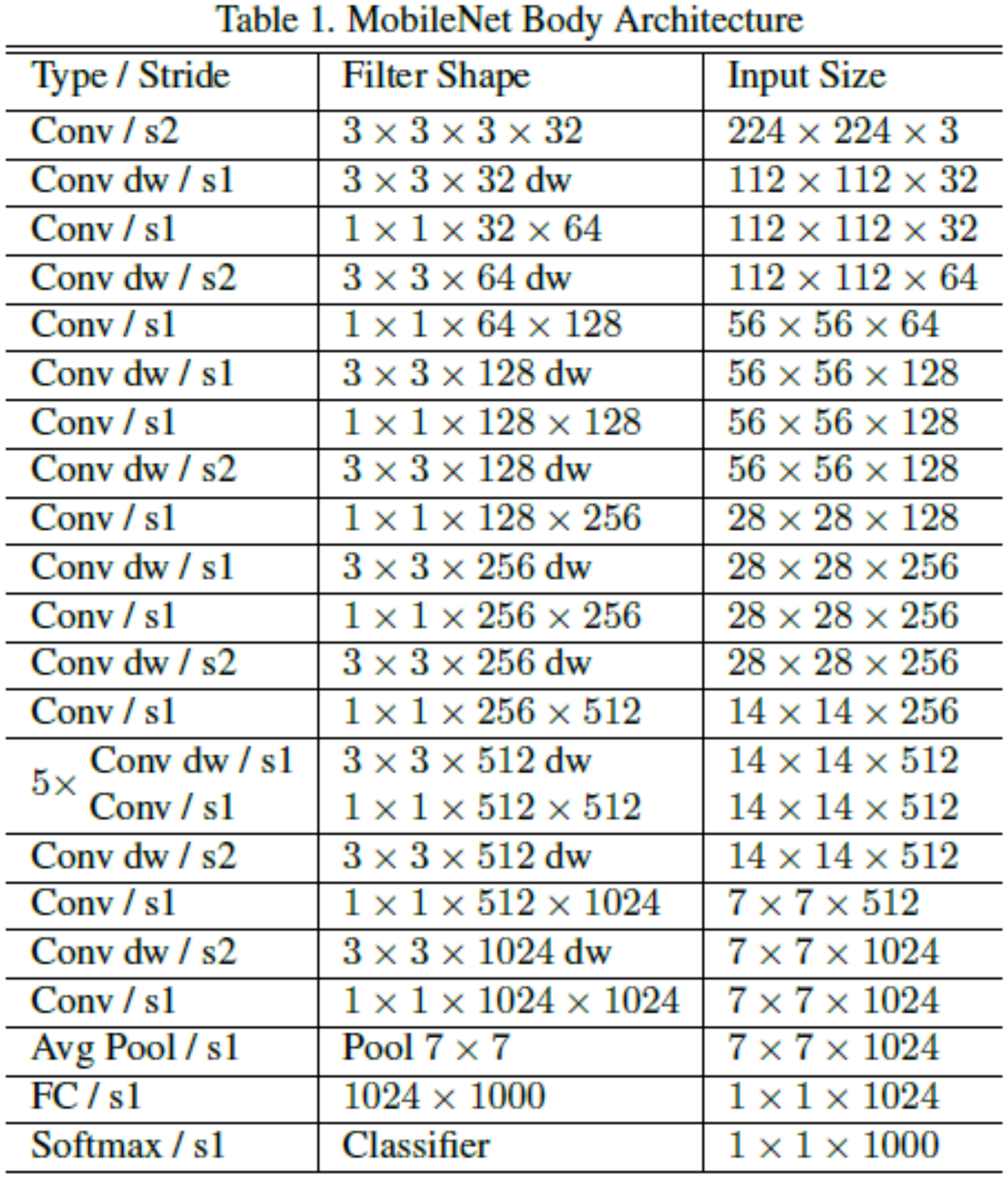
MobileNet (v1)

- Single streamlined, very light-weight architecture
- **Main idea:** Depthwise Separable Convolutions
- **Other ideas:** Width Multiplier α for Thinner Models + Resolution Multiplier ρ for Reduced Representation



Standard Convolution (Left), Depthwise separable convolution (Right) With BN and ReLU

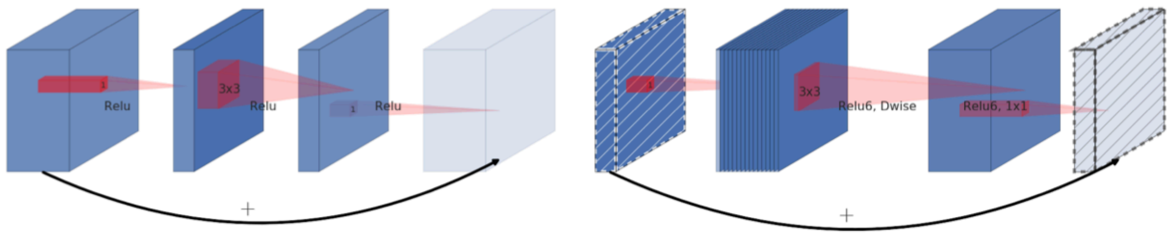
Table 1. MobileNet Body Architecture



Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1 Conv / s1	$3 \times 3 \times 512$ dw $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$ $14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

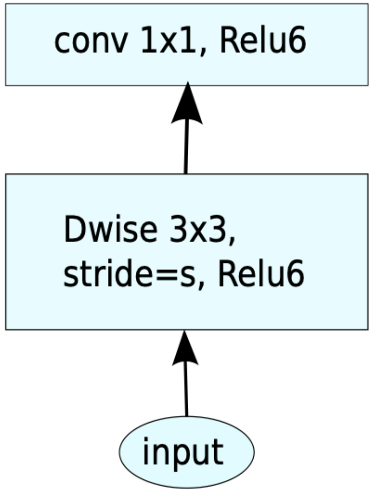
MobileNet (v2)

- **Main idea:** inverted residual structure
    - Adding residual connections between the narrow bottleneck layers (considerably more memory efficient - **Why?**)
    - Non-linearities are removed in narrow layers to maintain representational power
    - The intermediate expansion layer uses lightweight depthwise convolutions to filter features as a source of non-linearity

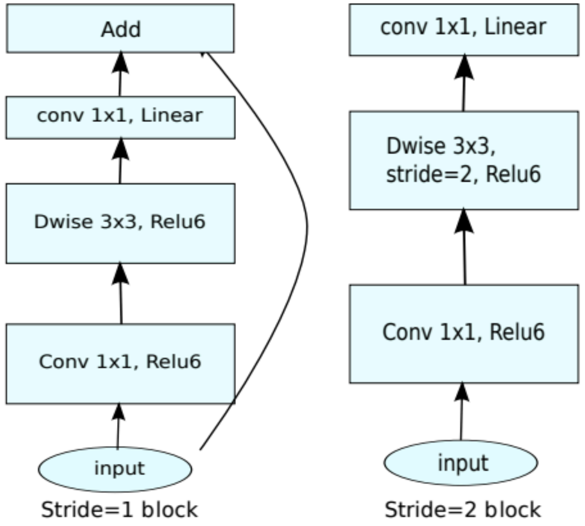


(a) Residual block

(b) Inverted residual block

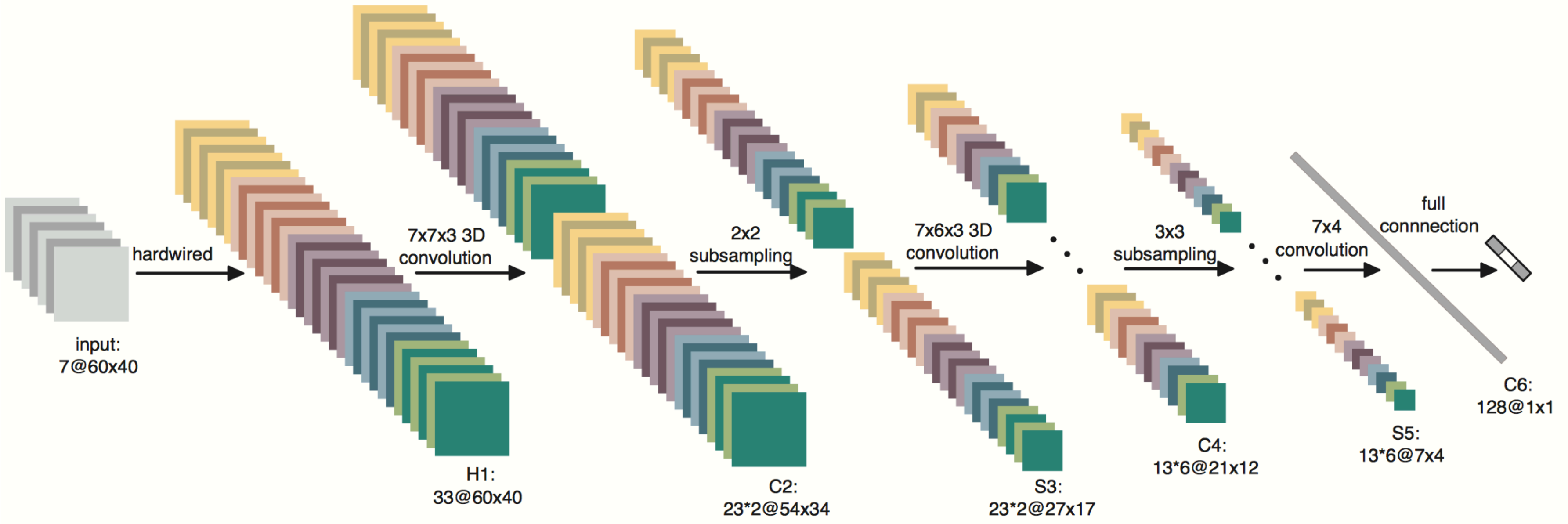


(b) MobileNet[27]



(d) Mobilenet V2

3D Convolutional Network (3D CNN), 2011

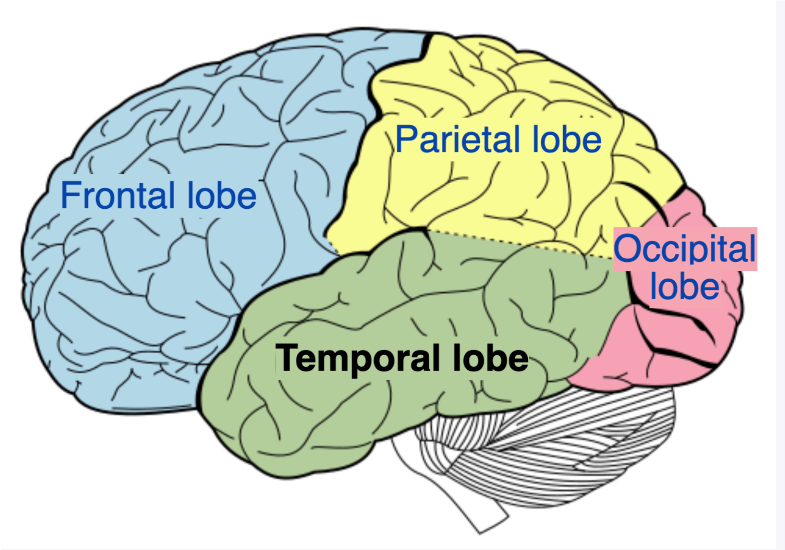


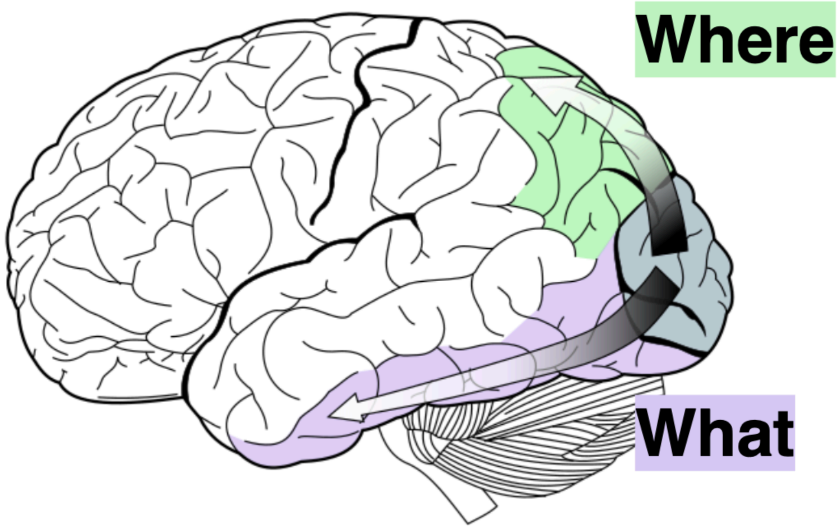
Key Technical Features:

- Going from 2D convolutional filters to 3D filters, to take temporal coherence into consideration

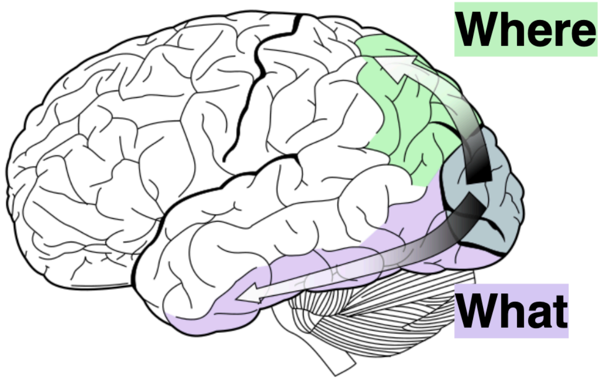
More Efficient Design?

- **“Two-streams hypothesis”** for human vision
  - The **dorsal stream** involves in the guidance of actions and recognizing where objects are in space. It contains a detailed map of the visual field. and detects & analyzes location movements
  - The **ventral stream** is associated with object recognition and form representation. Also described as the “what” stream, it has strong connections to the dorsal stream and other brain regions controlling memory or emotion
- **Long story short:** human brains use two relatively independent systems to recognize objects and to record temporal movements.





Two Stream Network, 2014



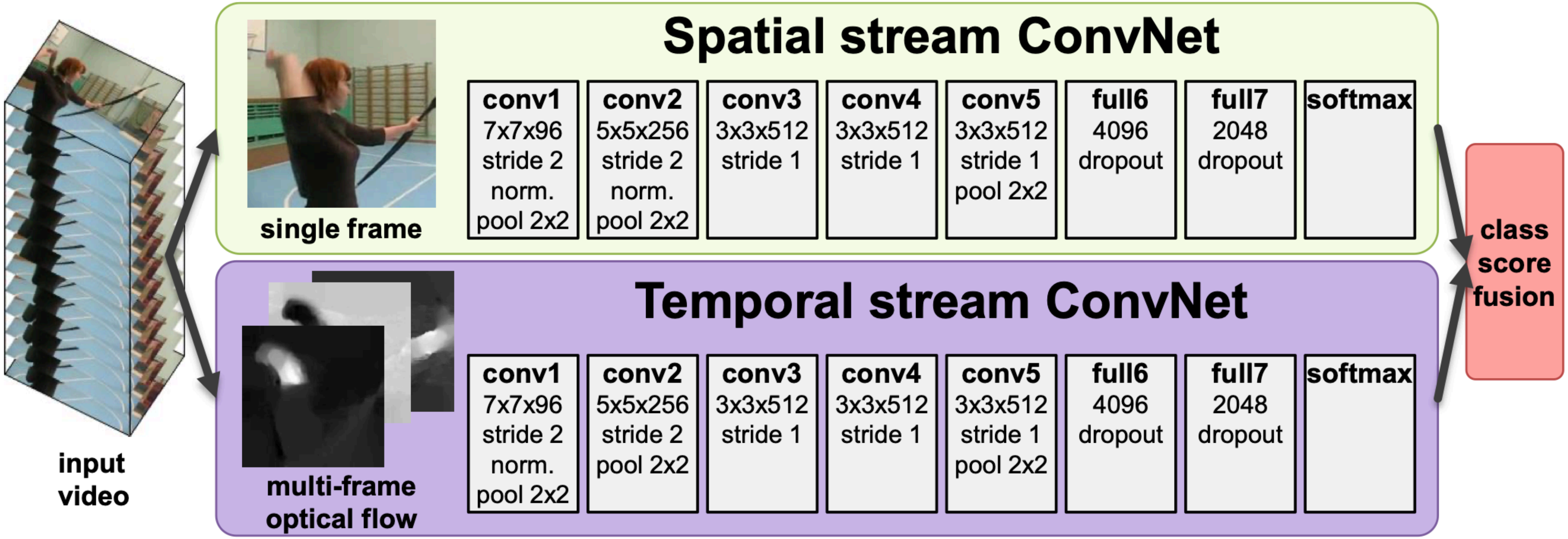
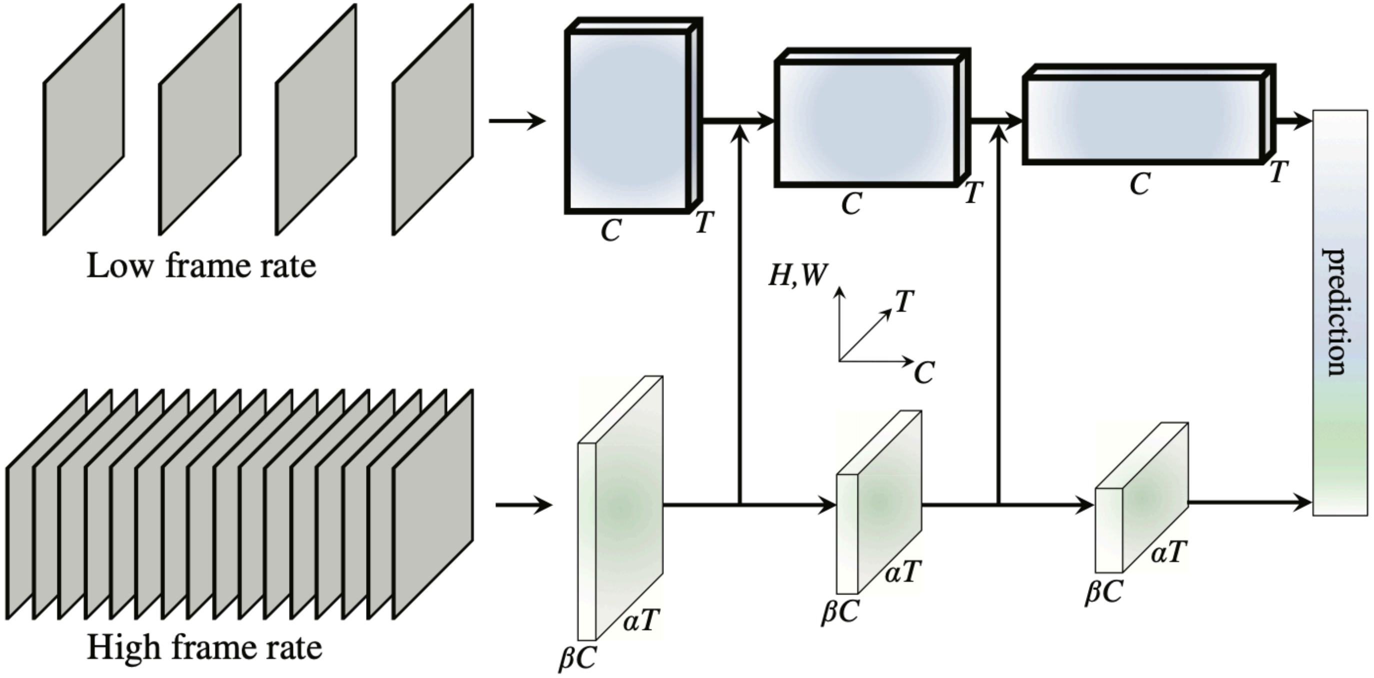


Figure 1: **Two-stream architecture for video classification.**

Slow-Fast Network, 2019

A state-of-the-art two-stream model with

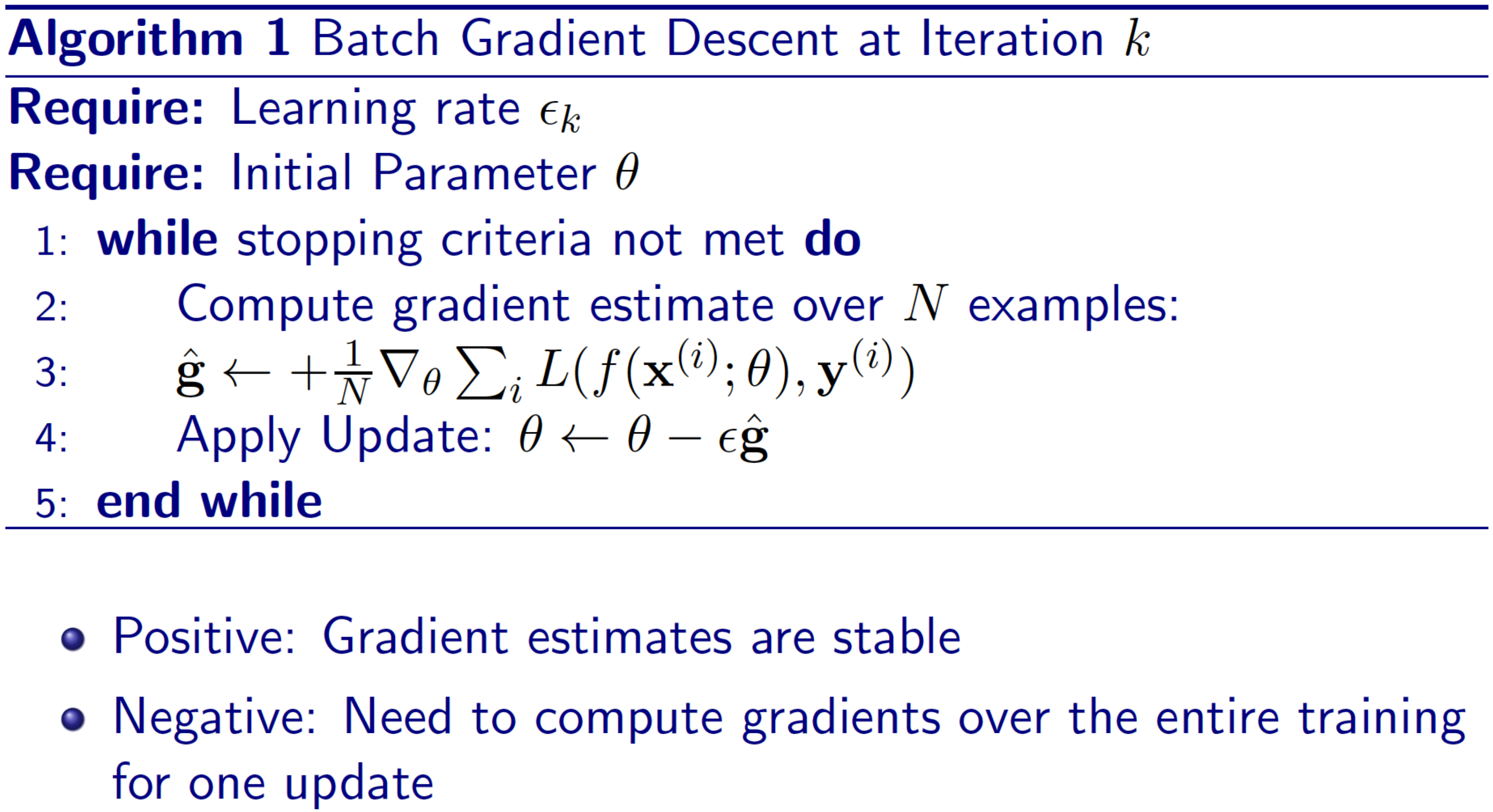
- *(i) a Slow pathway, operating at low frame rate, to capture spatial semantics*
- *(ii) a Fast pathway, operating at high frame rate, to capture motion at fine temporal resolution.*



Optimization Algorithms

Where the magic happens

Gradient Descent (GD)



Algorithm 1 Batch Gradient Descent at Iteration k

```
Require: Learning rate $\epsilon_k$
Require: Initial Parameter $\theta$
while stopping criteria not met do
    Compute gradient estimate over $N$ examples:
    $\hat{\mathbf{g}} \leftarrow +\frac{1}{N}\nabla_{\theta}\sum_i L(f(\mathbf{x}^{(i)};\theta),\mathbf{y}^{(i)})$
    Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
end while
```

- Positive: Gradient estimates are stable
- Negative: Need to compute gradients over the entire training for one update

Stochastic Gradient Descent (SGD)

Algorithm 2 Stochastic Gradient Descent at Iteration k

```
Require: Learning rate $\epsilon_k$
Require: Initial Parameter $\theta$
while stopping criteria not met do
    Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
    Compute gradient estimate:
    $\hat{\mathbf{g}} \leftarrow +\nabla_{\theta} L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
    Apply Update: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$
end while
```

- ϵ_k is learning rate at step k
- Sufficient condition to guarantee convergence:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty \text{ and } \sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

GD versus SGD

- Batch Gradient Descent:

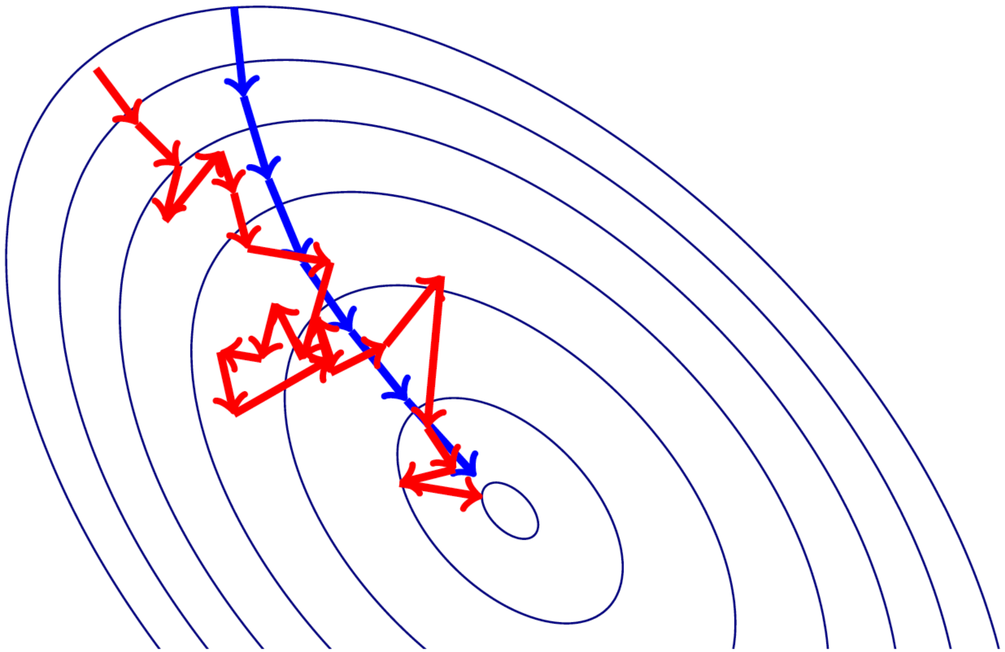
$$\hat{\mathbf{g}} \leftarrow +\frac{1}{N}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$$

$$\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$$

- SGD:

$$\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$$

$$\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$$



Minibatch

- Potential Problem: Gradient estimates can be very noisy
- Obvious Solution: Use larger mini-batches (In theory, growingly larger)

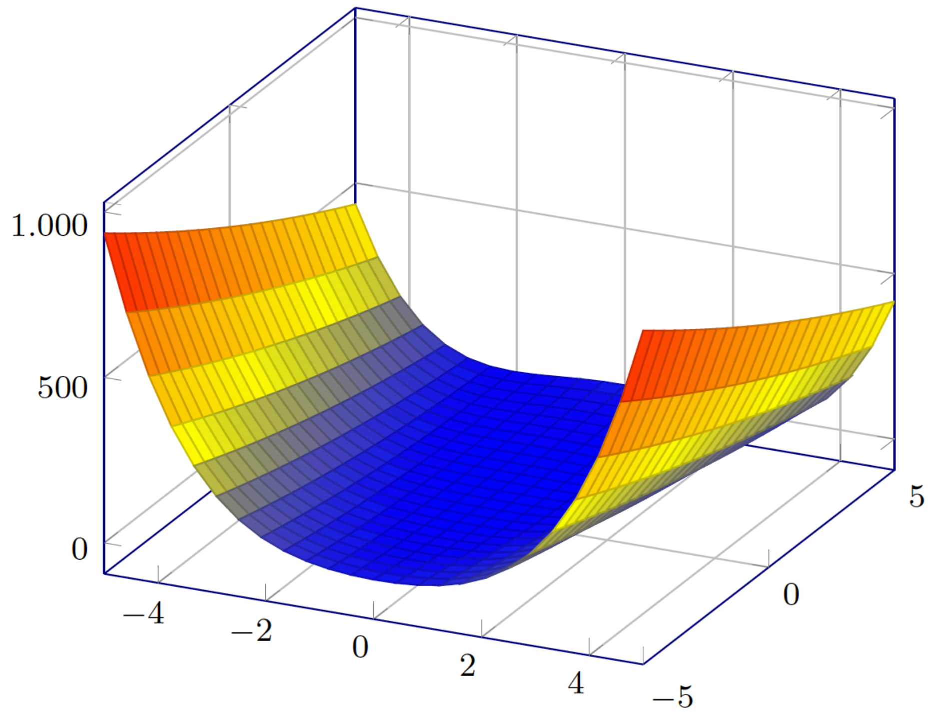
- Advantage: Computation time per update does not depend on number of training examples.
- This allows convergence on extremely large datasets

- **The larger MB size the better (only if you can)!!**

“Large Scale Learning with Stochastic Gradient Descent”, Leon Bottou.

Momentum

- The Momentum method is a method to accelerate learning using SGD
- In particular SGD suffers in the following scenarios:
    - Error surface has high curvature
    - Small but consistent gradients
    - Noisy gradients



- Gradient Descent would move quickly down the walls, but very slowly through the valley floor

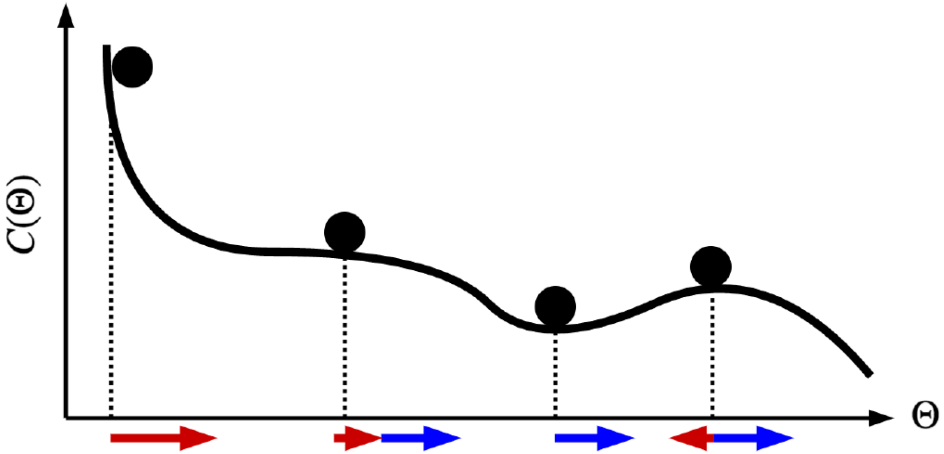
Momentum

- Update rule in SGD:

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \eta \boldsymbol{g}^{(t)}$$

where $\boldsymbol{g}^{(t)} = \nabla_{\Theta} C(\Theta^{(t)})$

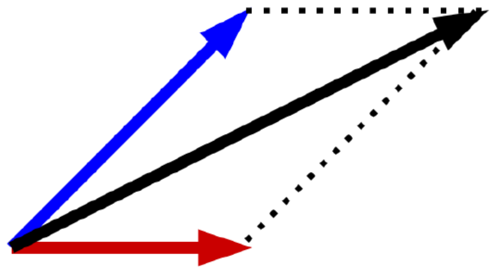
  - Gets stuck in local minima or saddle points



- Momentum: make the same movement $\boldsymbol{v}^{(t)}$ in the last iteration, corrected by negative gradient:

$$\boldsymbol{v}^{(t+1)} \leftarrow \lambda \boldsymbol{v}^{(t)} - (1 - \lambda) \boldsymbol{g}^{(t)}$$

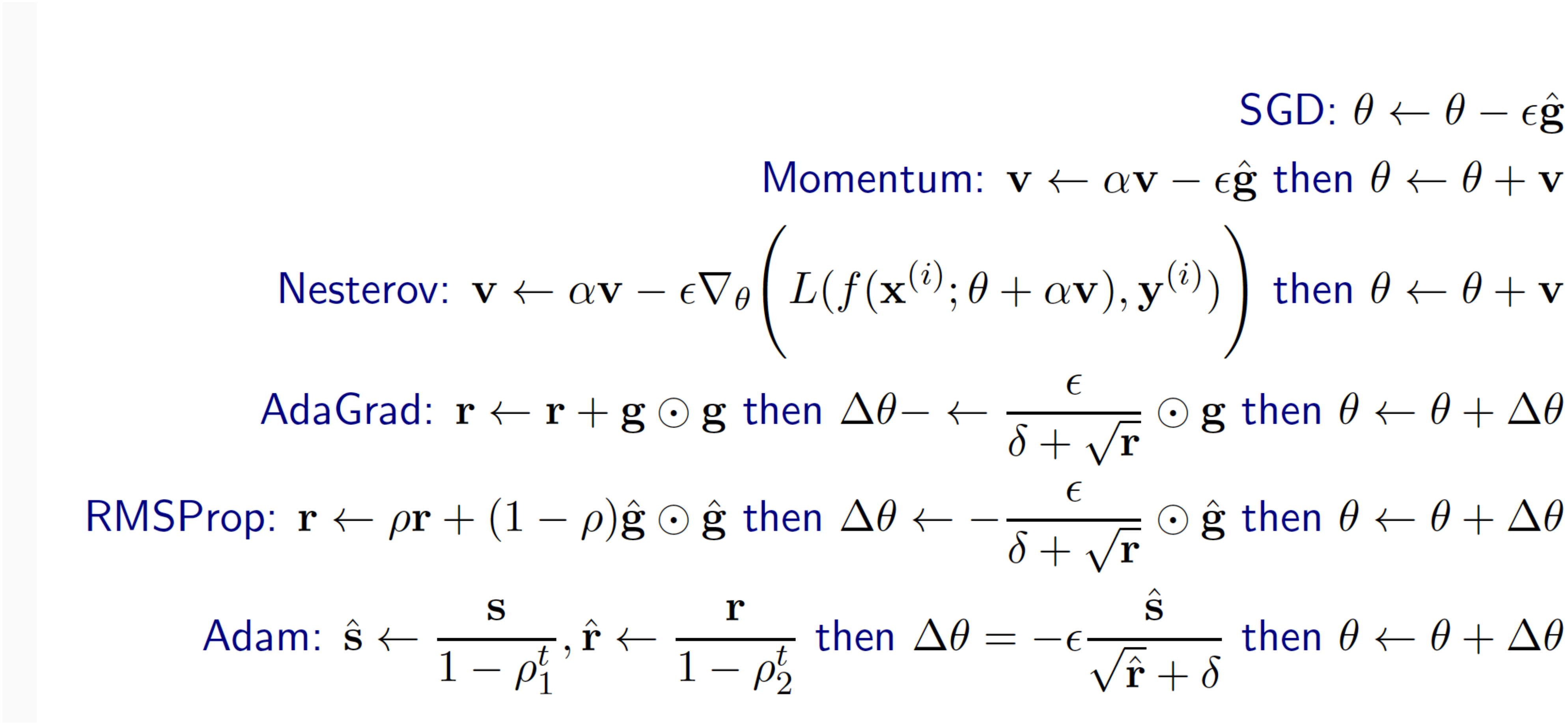
$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} + \eta \boldsymbol{v}^{(t+1)}$$



- $\boldsymbol{v}^{(t)}$ is a moving average of $-\boldsymbol{g}^{(t)}$

Adaptive Learning Rate Optimization

- Popular Solver Examples: AdGrad, RMSProp, Adam



$$\text{SGD: } \theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$$

$$\text{Momentum: } \mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \hat{\mathbf{g}} \text{ then } \theta \leftarrow \theta + \mathbf{v}$$

$$\text{Nesterov: } \mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \left(L(f(\mathbf{x}^{(i)}; \theta + \alpha \mathbf{v}), \mathbf{y}^{(i)}) \right) \text{ then } \theta \leftarrow \theta + \mathbf{v}$$

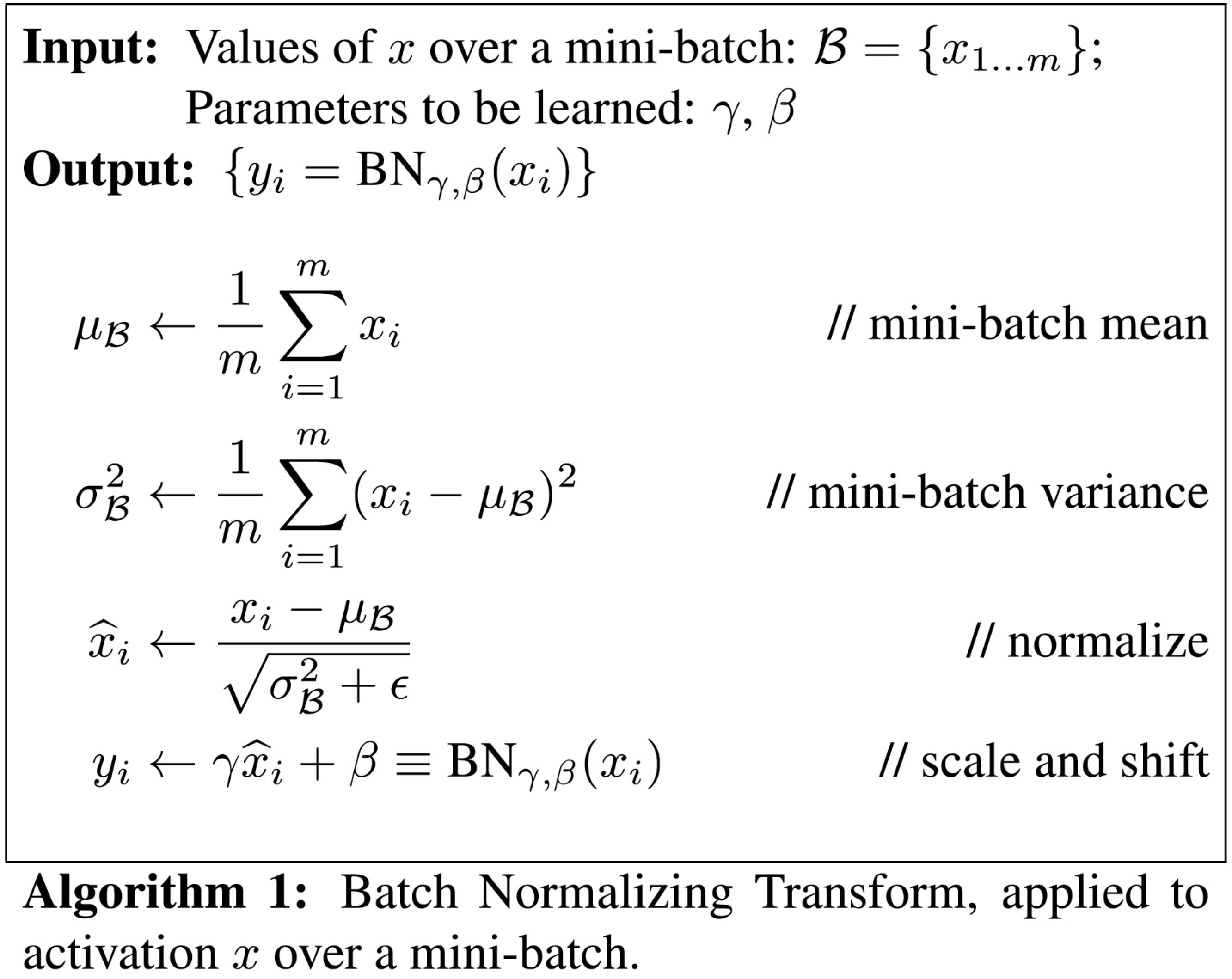
$$\text{AdaGrad: } \mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g} \text{ then } \Delta\theta - \leftarrow \frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g} \text{ then } \theta \leftarrow \theta + \Delta\theta$$

$$\text{RMSProp: } \mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \hat{\mathbf{g}} \odot \hat{\mathbf{g}} \text{ then } \Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}} \text{ then } \theta \leftarrow \theta + \Delta\theta$$

$$\text{Adam: } \hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \rho_1^t}, \hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - \rho_2^t} \text{ then } \Delta\theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta} \text{ then } \theta \leftarrow \theta + \Delta\theta$$

Batch Normalization

- In ML, we assume future data will be drawn from same probability distribution as training data
- For a hidden layer, after training, the earlier layers have new weights and hence may generate a new distribution for the next hidden layer
- We want to reduce this internal covariate shift for the benefit of later layers



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Batch Normalization

- First three steps are just like standardization of input data, but with respect to only the data in mini-batch.

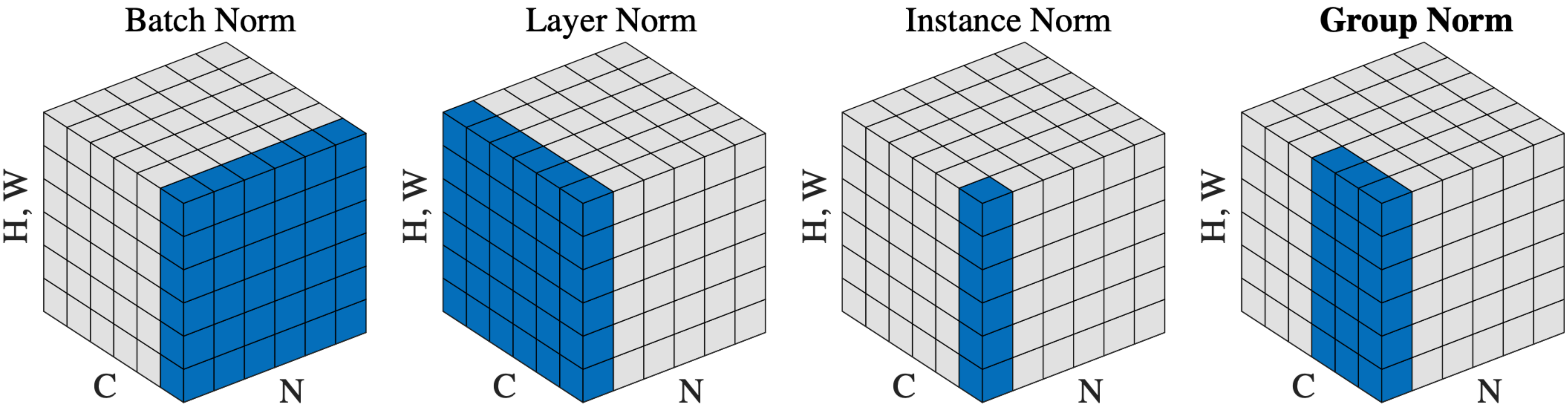
- We can take derivative and incorporate the learning of last step parameters into backpropagation.

- Note last step can completely un-do previous 3 steps

- But even if so, this un-doing is driven by the later layers, not the earlier layers; later layers get to "choose" whether they want standard normal inputs or not

- In fact, the **true reason** why BN works remains to be a mystery …

Many Normalization Schemes…



Comparing Popular Normalization methods. Each subplot shows a feature map tensor, with *N* as the batch axis, *C* as the channel axis, and (*H, W*) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Weight Initialization

- All Zero Initialization: Terribly Wrong!
  - If every neuron in the network computes the same output, then they will also all compute the same gradients during back-propagation and undergo the exact same parameter updates.
  - Need “break the symmetry”

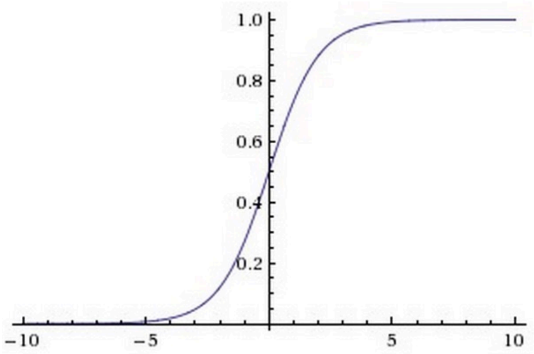
- Small Random Initialization is the standard practice
- Current recommendation for initializing CNNs with RELU: **Why?**

`w = np.random.randn(n) * sqrt(2.0/n)`

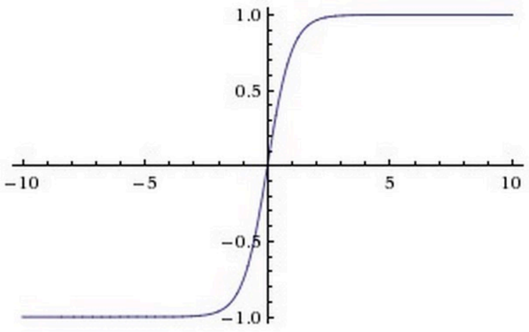
- “randn”: Gaussian; “n”: the number of inputs for current layer.

- For general NNs, layer-wise pre-training is safe.
- Even safer: start from a pre-trained model

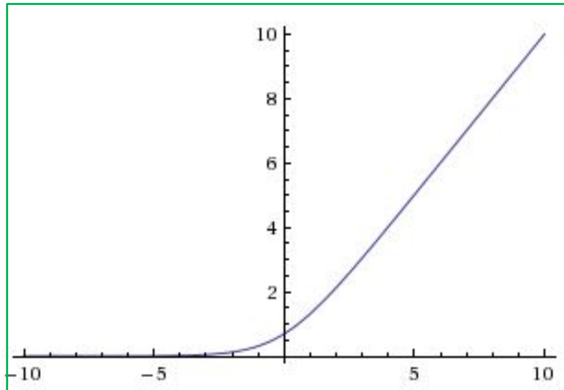
Choice of Activation Functions



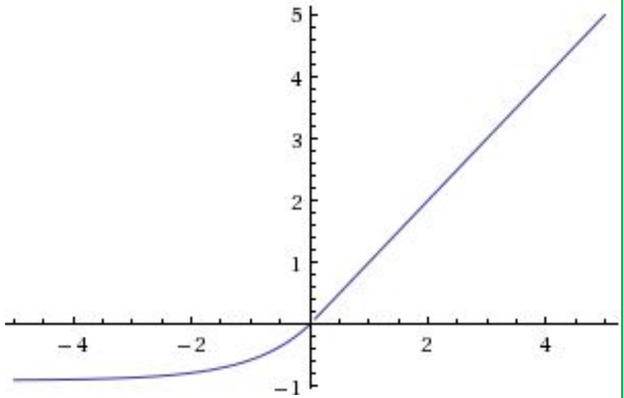
Sigmoid



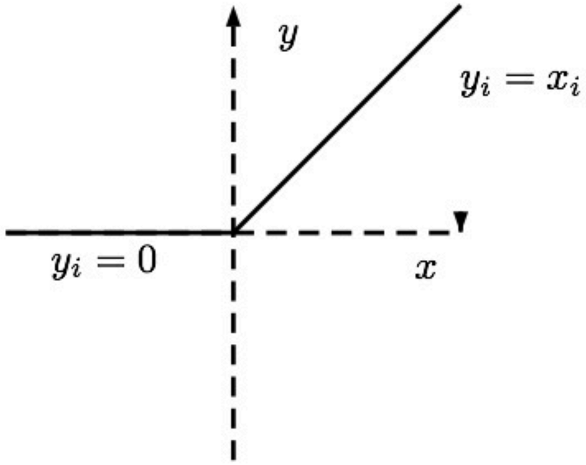
Tanh



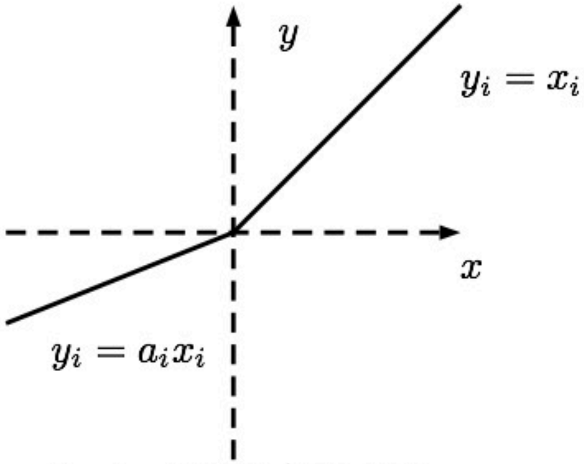
Softplus



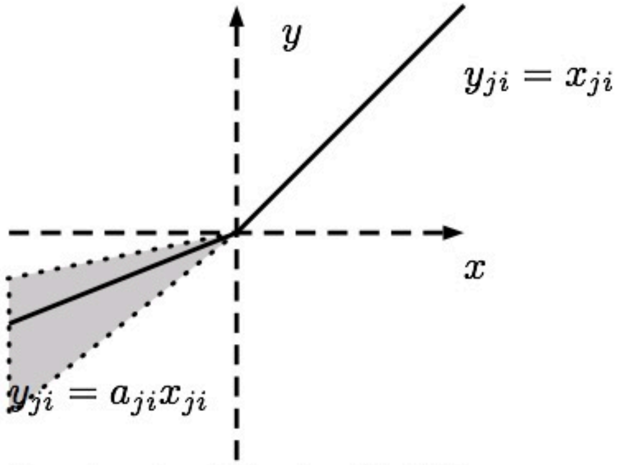
ELU



ReLU



Leaky ReLU/PReLU



Randomized Leaky ReLU

Monitor Your Training Curve

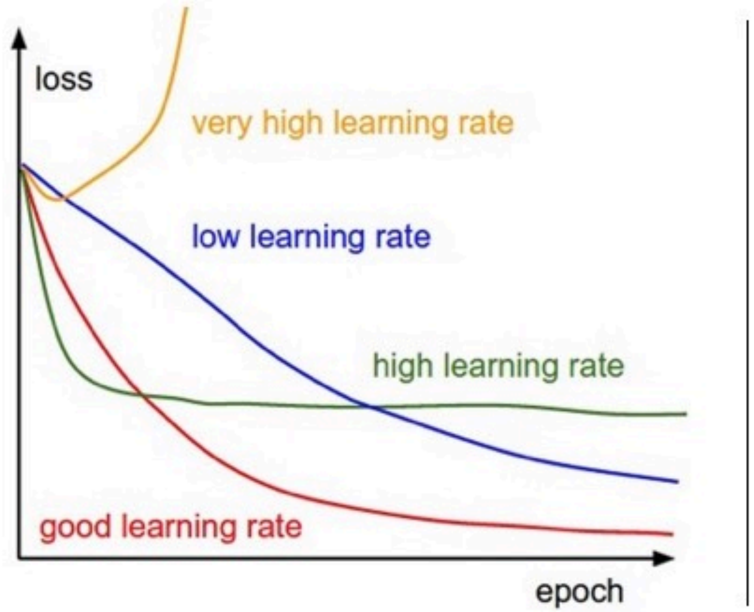
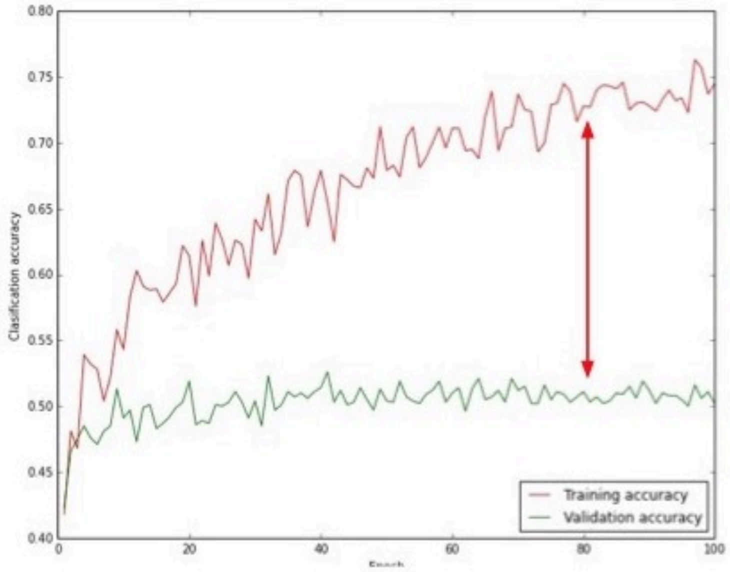


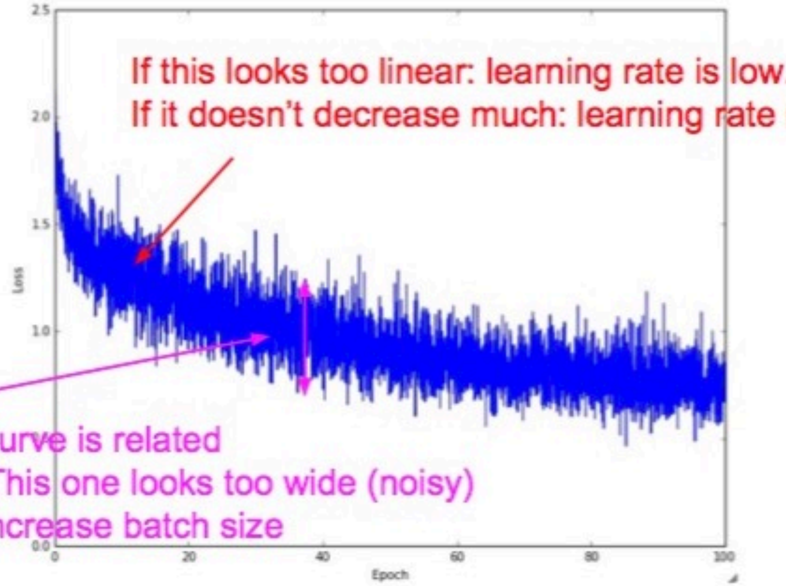
Figure 1



big gap = overfitting
=> increase regularization strength

no gap
=> increase model capacity

Figure 3



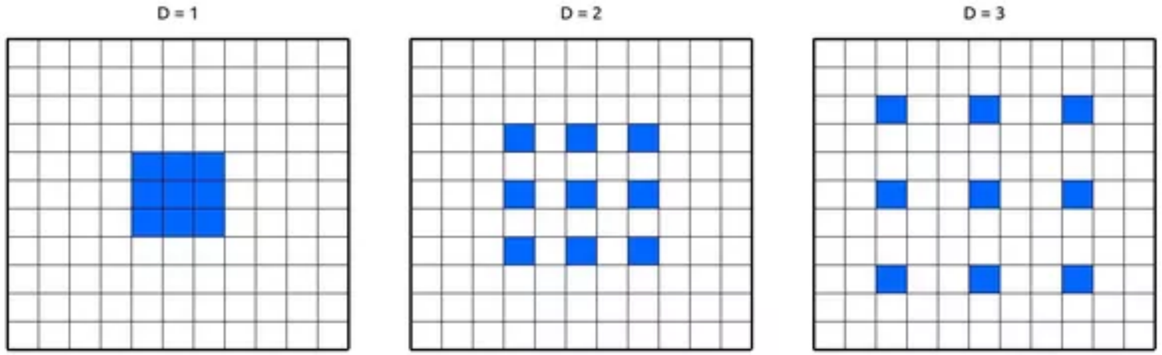
If this looks too linear: learning rate is low.
If it doesn't decrease much: learning rate might be too high

the "width" of the curve is related
to the batch size. This one looks too wide (noisy)
=> might want to increase batch size

Figure 2

The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

Dilated Convolutions, 2015



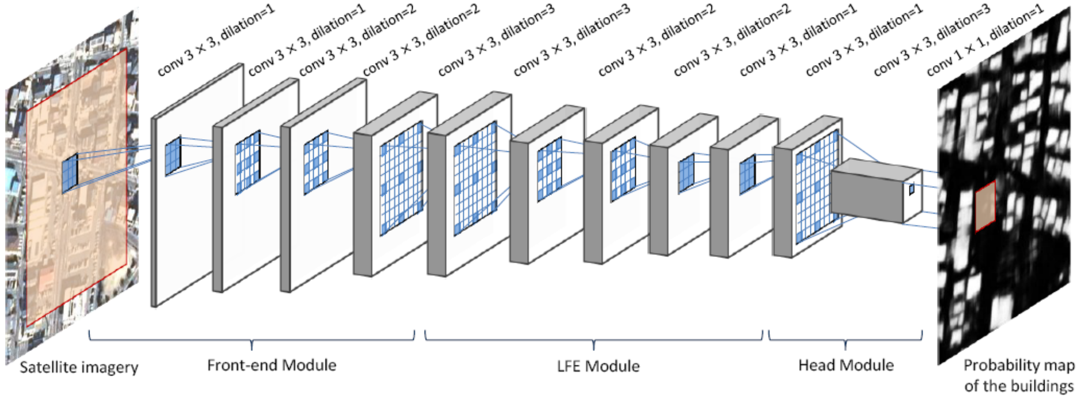
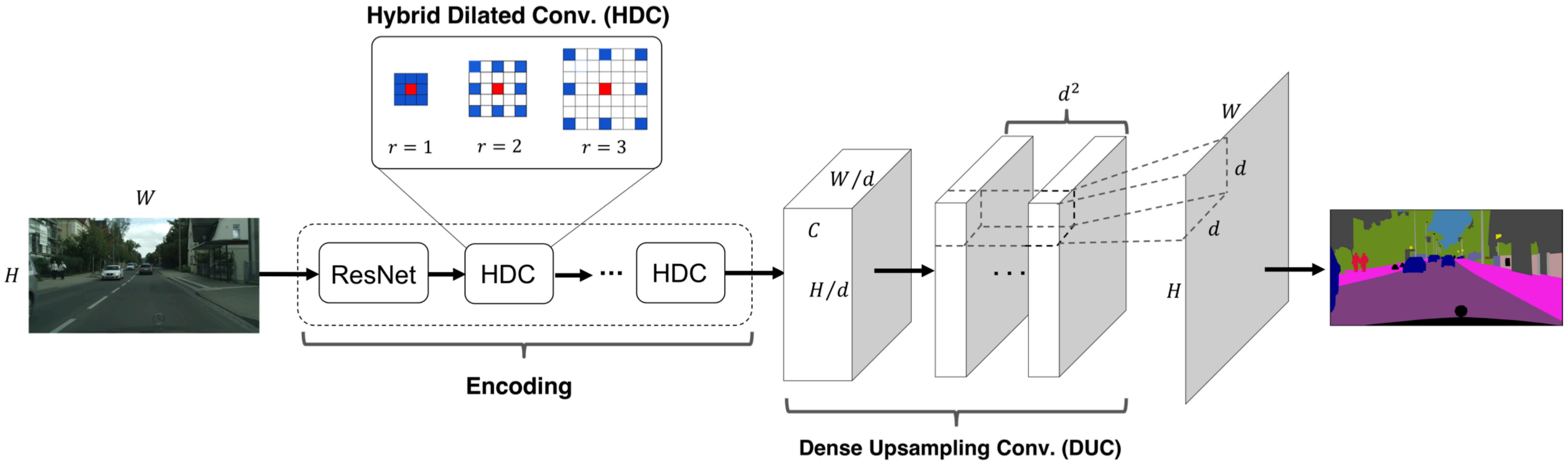
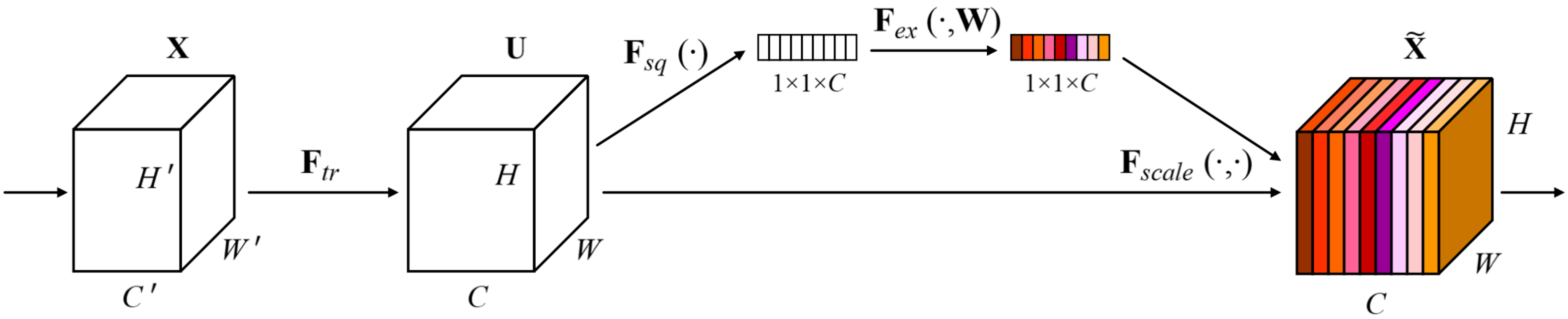


Figure 2. Overview of the proposed network architecture. In each layer, filter kernels are depicted in square with grid pattern. The blue cells in dilated kernel represent valid weights and blank cells represent invalid region. As shown, in case of kernels with dilation factor of 2, valid weights align by interval of 1 and in case of 3, they align by interval of 3.

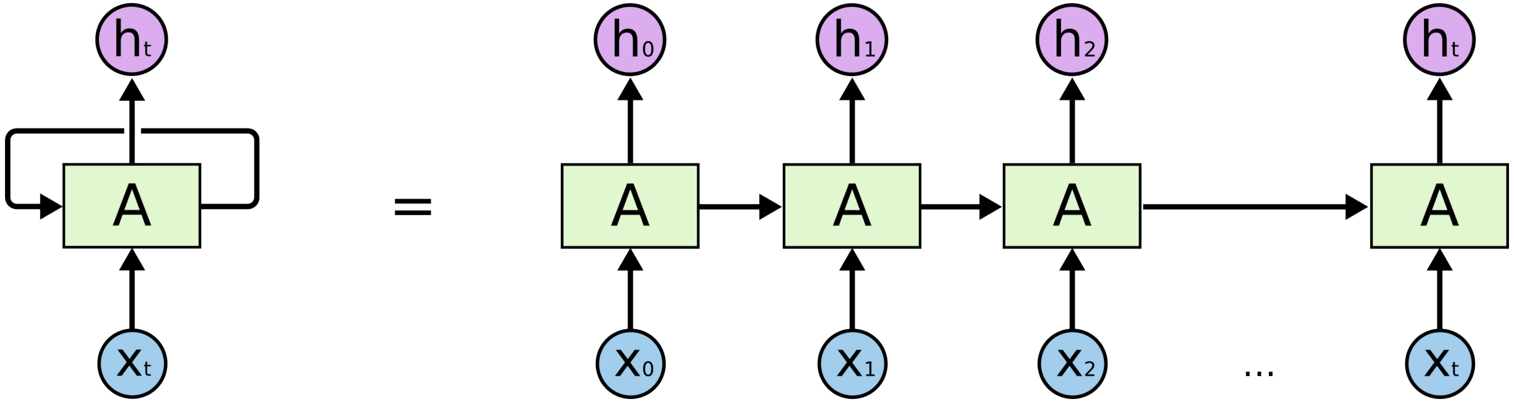


Squeeze and Excitement

A special "channel attention": to adaptively recalibrates channel-wise feature responses by explicitly modelling inter-channel dependencies



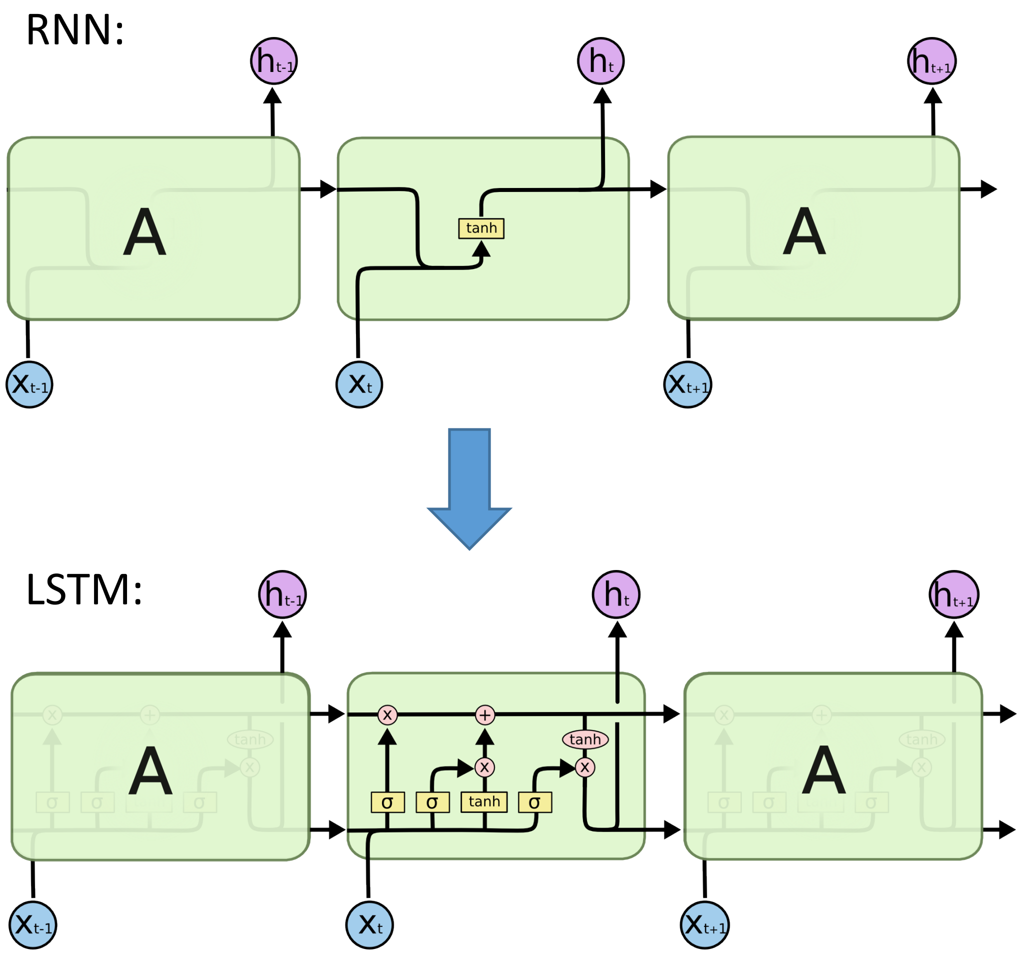
RNN and LSTM



- A RNN is **unfolded** its forward and backward computations.
- Backpropagation Through Time (**BPTT**): Because the parameters are shared by all time steps in the network, the gradient at each output depends not only on the calculations of the current time step, but also the previous time steps
- **Vanishing/Exploding Gradients**: Difficulty in learning long-term dependency

An intro article for RNN/LSTM: <u>"Understanding LSTM Networks":</u>
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

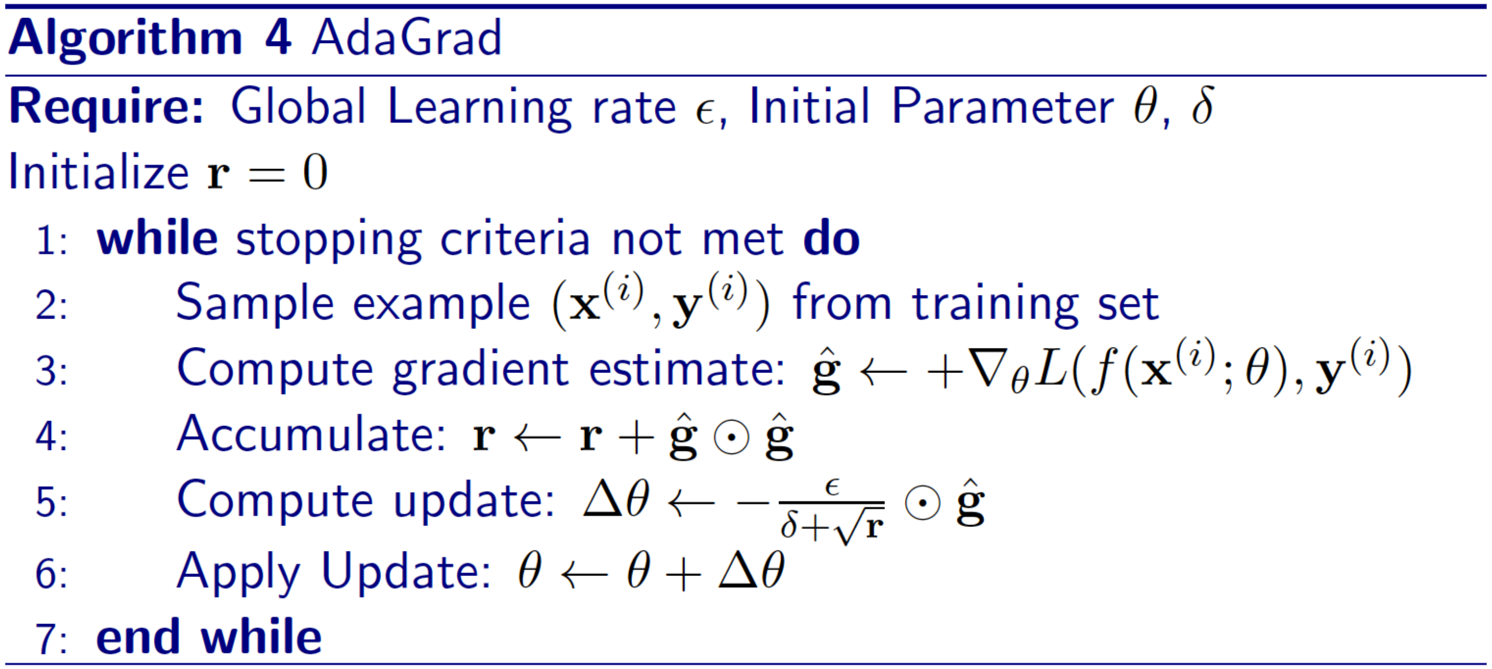
RNN and LSTM



- A Long Short Term Memory (**LSTM**) combats vanishing gradients through a **gating** mechanism, thus capturing **long-term dependency** better.
  - Similar “shortcut” idea to ResNet
- An LSTM does the exact same thing as a RNN, just in a different way!
- **Key Idea:** the gating functions are learned together with weights, and determine how much information we would like keep from last state and current computation, etc.

AdaGrad

- **Idea:** Downscale a model parameter by square-root of sum of squares of all its historical values
- Parameters that have large partial derivative of the loss -> learning rates for them are rapidly declined
- Some interesting theoretical properties

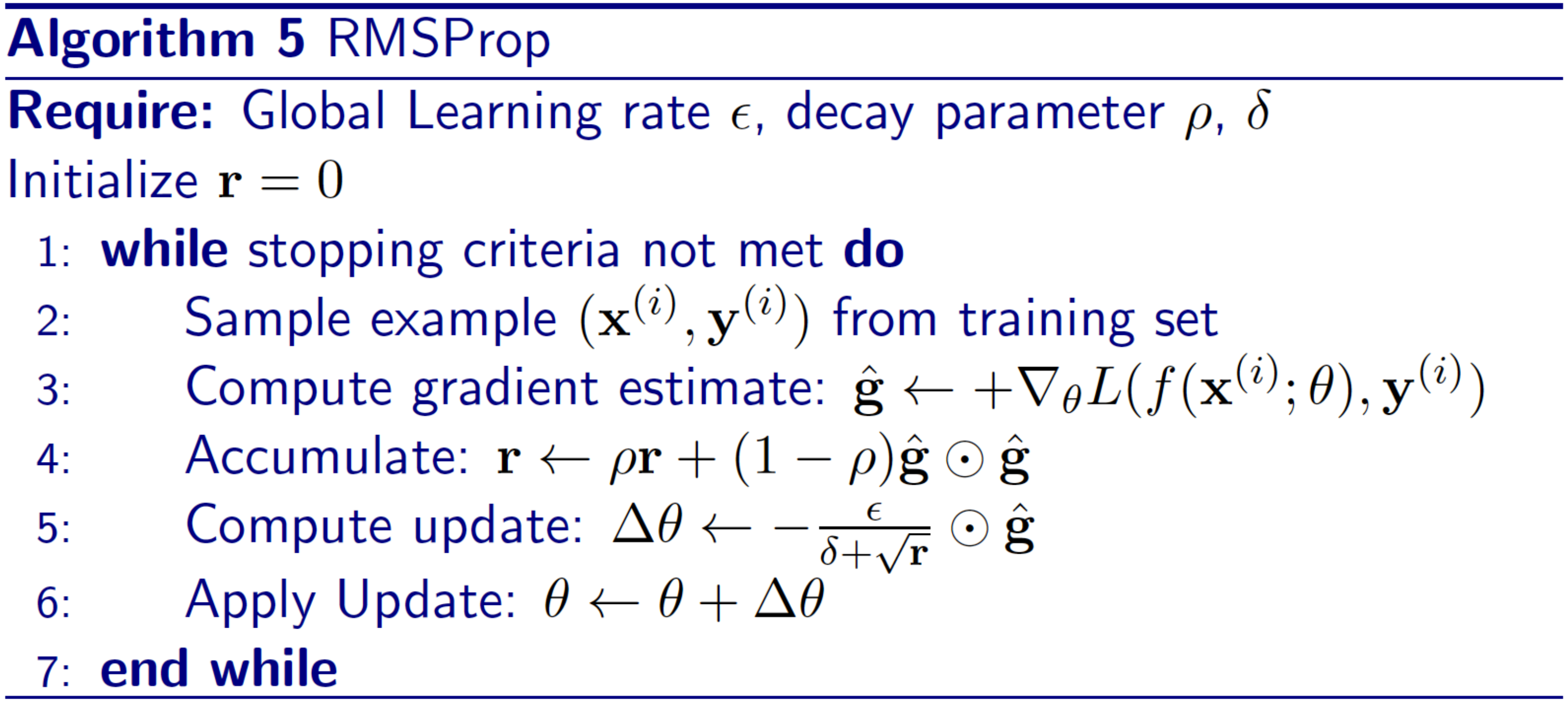


Algorithm 4 AdaGrad

```
Require: Global Learning rate $\epsilon$, Initial Parameter $\theta$, $\delta$
Initialize $\mathbf{r} = 0$
while stopping criteria not met do
    Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
    Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
    Accumulate: $\mathbf{r} \leftarrow \mathbf{r} + \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
    Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$
    Apply Update: $\theta \leftarrow \theta + \Delta\theta$
end while
```

RMSProp

- AdaGrad might shrink the learning rate too aggressively, we can adapt it to perform better by accumulating an exponentially decaying average of the gradient

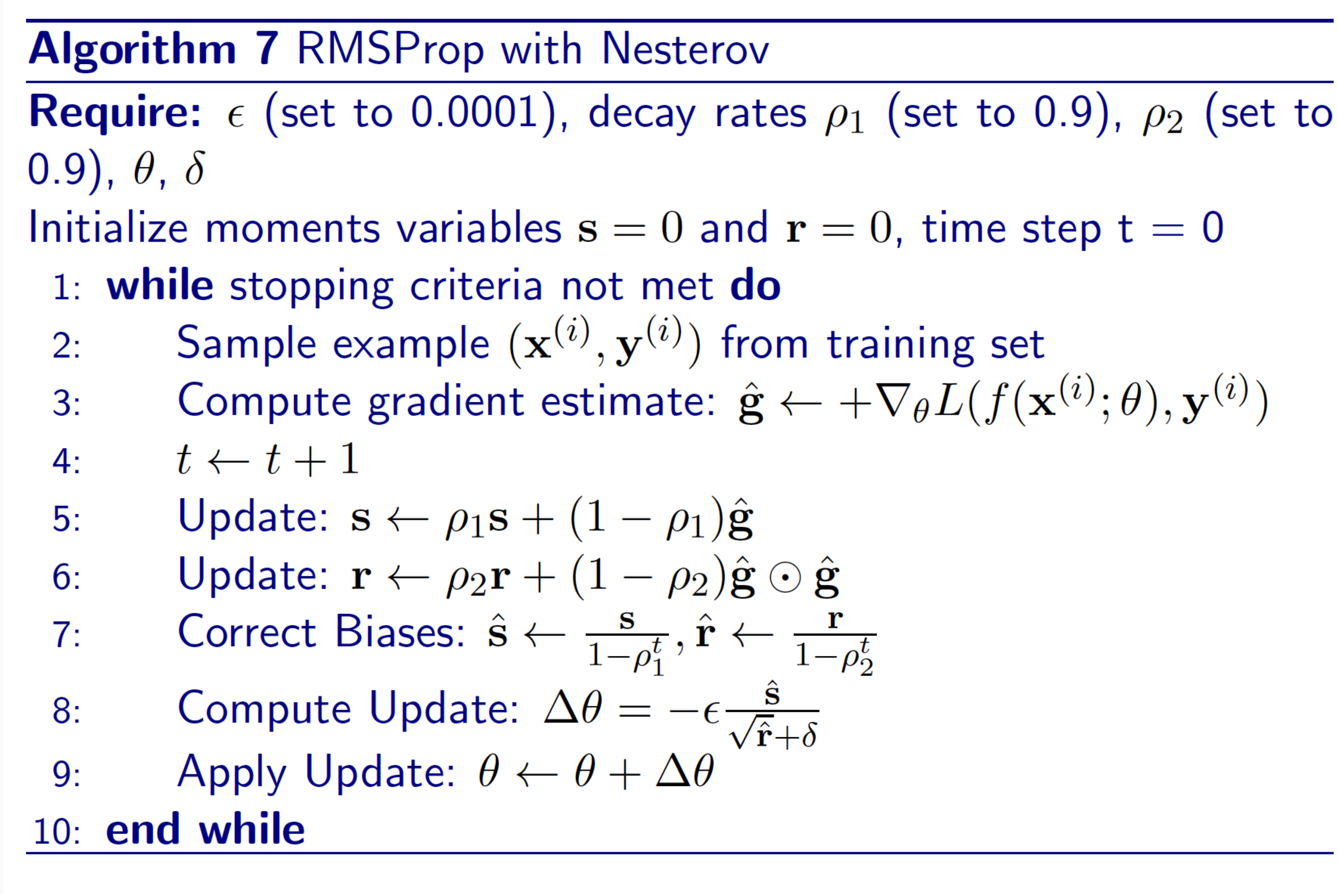


Algorithm 5 RMSProp

```
Require: Global Learning rate $\epsilon$, decay parameter $\rho$, $\delta$
Initialize $\mathbf{r} = 0$
while stopping criteria not met do
    Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
    Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
    Accumulate: $\mathbf{r} \leftarrow \rho\mathbf{r} + (1 - \rho)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
    Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$
    Apply Update: $\theta \leftarrow \theta + \Delta\theta$
end while
```

Adam

- Adam is like RMSProp with Momentum but with bias correction terms for the first and second moments



```
Algorithm 7 RMSProp with Nesterov
Require: $\epsilon$ (set to 0.0001), decay rates $\rho_1$ (set to 0.9), $\rho_2$ (set to 0.9), $\theta$, $\delta$
Initialize moments variables $\mathbf{s} = 0$ and $\mathbf{r} = 0$, time step t $= 0$
while stopping criteria not met do
    Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
    Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$
    $t \leftarrow t + 1$
    Update: $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1)\hat{\mathbf{g}}$
    Update: $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
    Correct Biases: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\rho_1^t}, \hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1-\rho_2^t}$
    Compute Update: $\Delta\theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}}+\delta}$
    Apply Update: $\theta \leftarrow \theta + \Delta\theta$
end while
```

Hyperparameter Choice

- **Mini-batch size:** preferably to be power of 2
- **Filter size, pooling size, padding**
  - **Recommended** (but no universal best setting exists: always try some different!): small filter (e.g., 3×3, 1×1), small stride (e.g., 1), small max pooling (e.g., 2×2), zero-padding
  - Important to preserve the spatial size of feature maps when going deep
- **Learning rate**
  - **Popular Learning rate scheduling:** if you see that you stopped making progress on the validation set, divide the LR by 2 (or by 5), and keep going, which might give you a surprise.
  - More research: SGD re-start, cyclical learning rate, etc.