JOHNS HOPKINS
WHITING SCHOOL *of* ENGINEERING

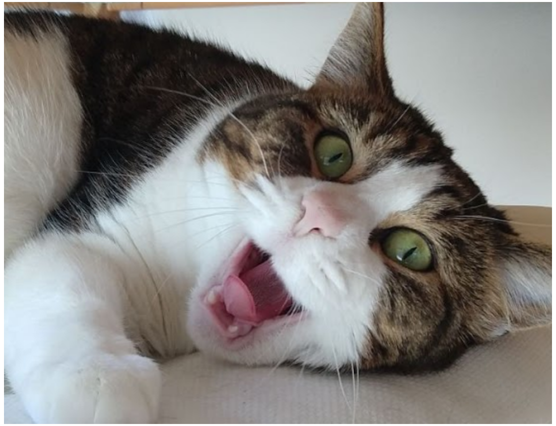
EN.601.482/682 Machine Learning: Deep Learning

Recap

Silvio Amir
samir@jhu.edu

Department of Computer Science
Johns Hopkins University

Image Classification



5.1	cat	1
2.3	car	0
-0.7	frog	0

One-hot encoding

Image Classification

Input image

$\mathbf{x}$

$f(\mathbf{x}, \mathbf{W})$

10 numbers for class scores

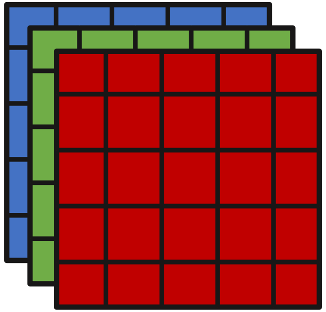
$\mathbf{W}$

Parameters or weights

Image Classification

Input image

x



$f(\mathbf{x}, \mathbf{W})$

W

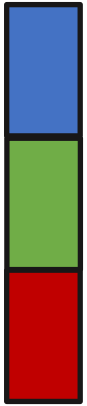
Parameters or weights

10 numbers for class scores

Image Classification

Input image

x



Array of 32x32x3 numbers
(3072 numbers total)

$f(\mathbf{x}, \mathbf{W})$

W

Parameters or weights

10 numbers for class scores

Linear Classifier

Input image
x

Array of 32x32x3 numbers
(3072 numbers total)

3072 x 1

$$\boxed{f(x,W)} = \boxed{W}\boxed{x} + \boxed{b}$$

10 x 1

10 x 3072

10 x 1

Bias term: constant!
→ Input independent preference

$$f(\mathbf{x},\mathbf{W})$$

W
Parameters or weights

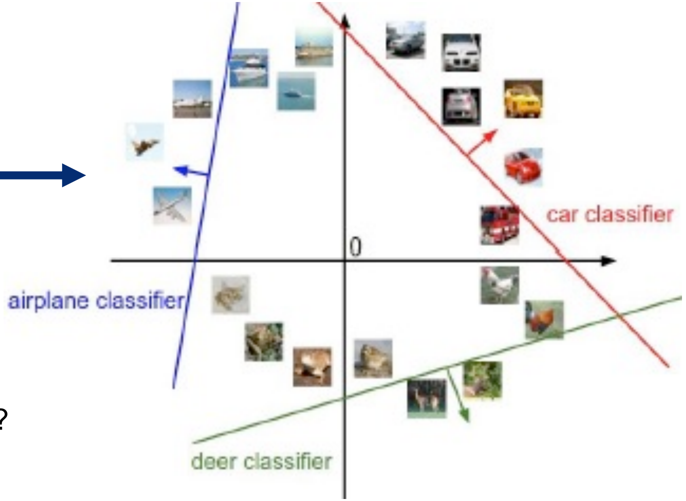
10 numbers for class scores

Training

Scores: $s = f(x_i, W) = Wx_i + b$

f(x,W)

W
How to estimate?



Loss Function

Loss function: quantifies classifier performance

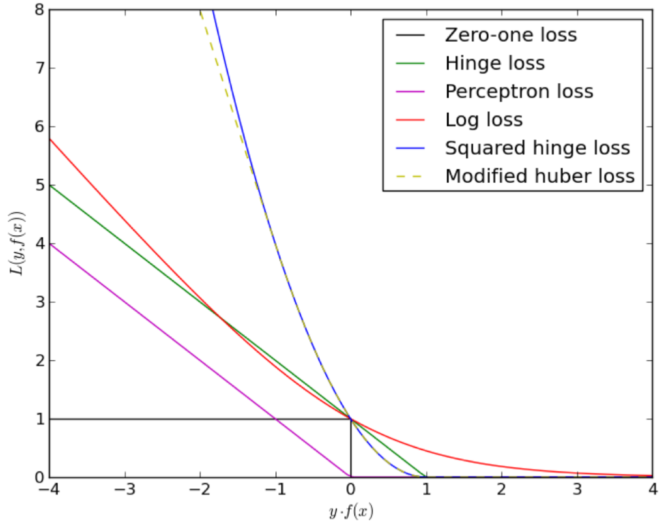
Given a dataset $\{(x_i, y_i)\}_{i=1}^N$ where

x_i is the image (or feature representation)

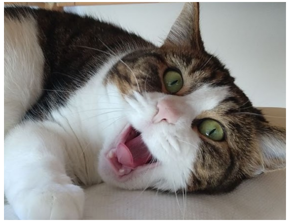
y_i is the corresponding label (integer)

Loss over the dataset: sum over all examples

$$L = \frac{1}{N} \sum_i L_i\left(f(x_i, W), y_i\right)$$



The Mean Squared Error



Loss: $L_i = (y_i - s_i)^2$

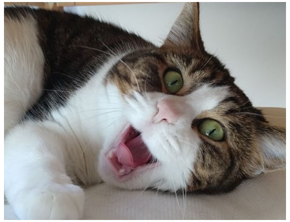
True label

compare

3.2 → ← **5**

$L_i = (5 - 3.2)^2 = 3.24$

Negative Log Likelihood

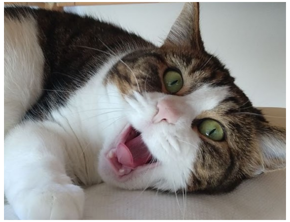


True label for cat
(one-hot encoding)

	Scores	True label
Cat	**3.2**	**1.00**
Car	5.1	0.00
Bird	-1.7	0.00

Negative Log Likelihood

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

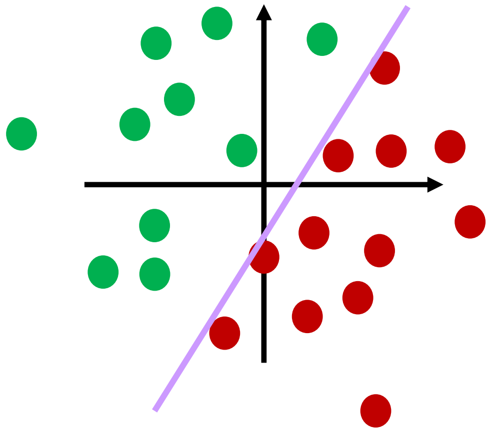


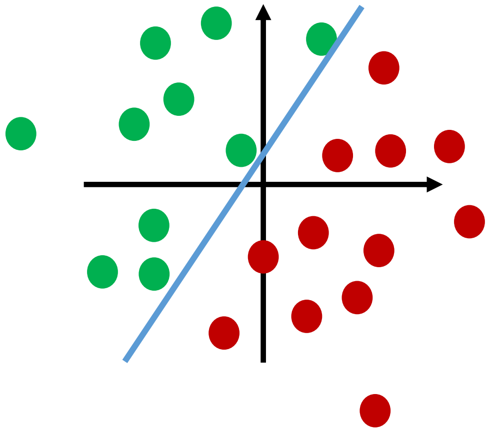
Loss: $L_i = -\log P(Y = y_i|X = x_i)$ → Log-likelihood of true class

		exp()		normalize	
Cat	**3.2**		**24.4**		**0.13**
Car	5.1		164.0		0.87
Bird	-1.7		0.18		0.00
			Probabilities are ≥ 0		Probabilities must sum to 1

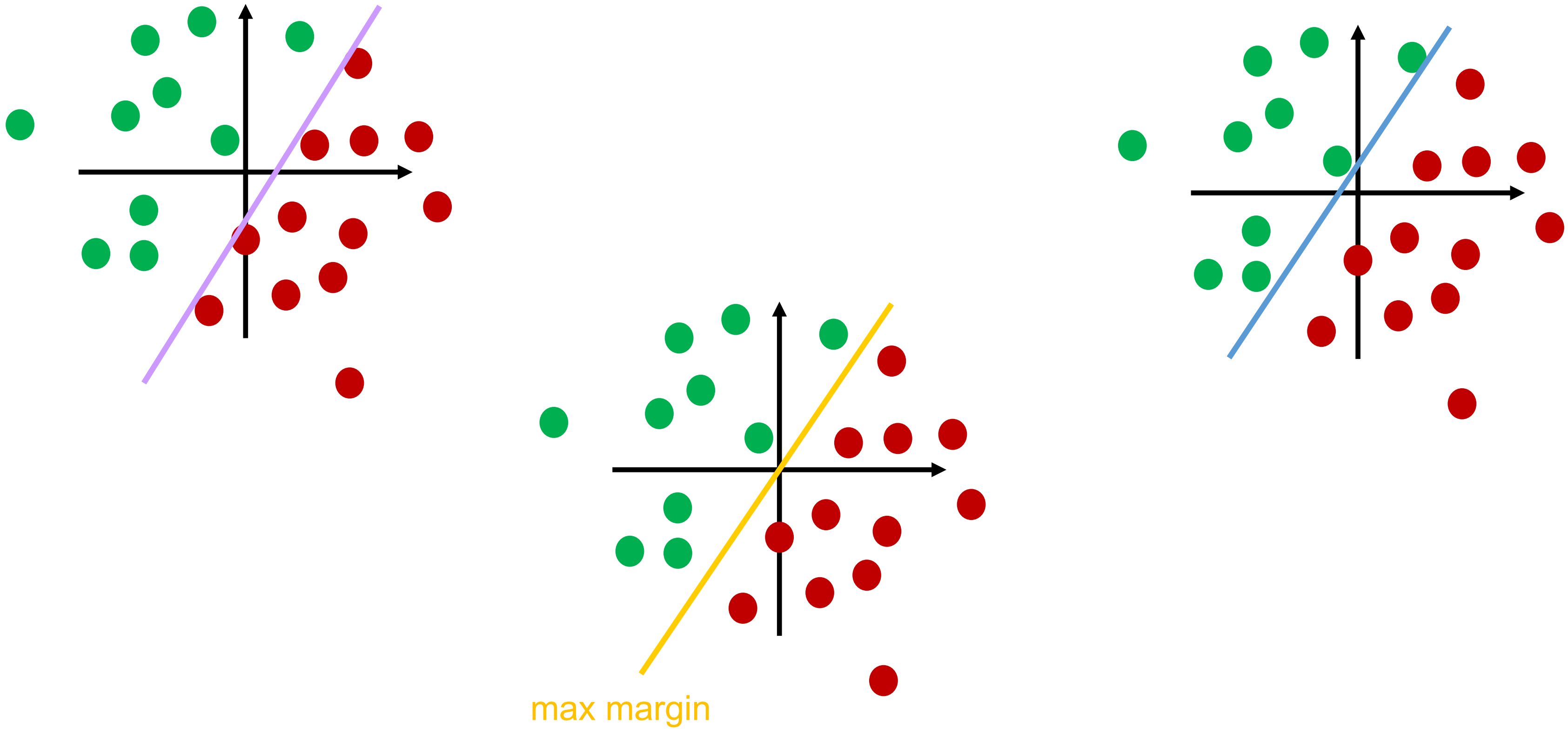
→ L_i = -log(0.13)
= **0.89** or **2.04**

Which W to choose?

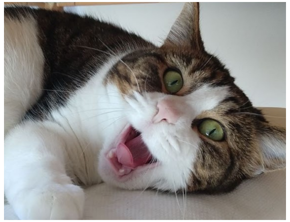




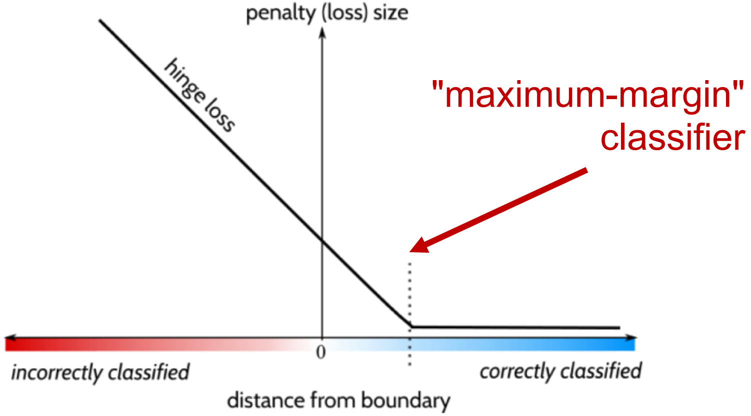
Which W to choose?



The Hinge (SVM) Loss



Cat	**3.2**	1.3	2.2
Car	5.1	**4.9**	2.5
Bird	-1.7	2.0	**-3.1**

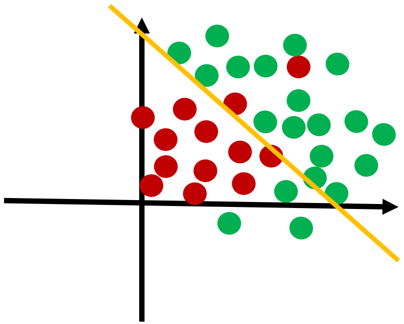


Loss:

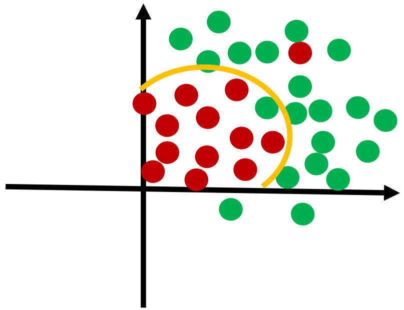
$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

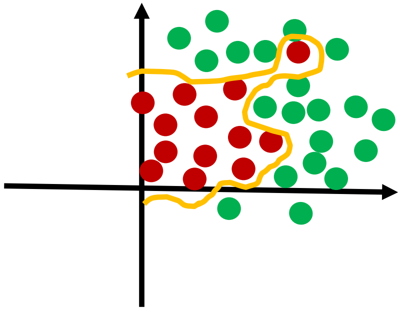
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Bias and Variance



High bias





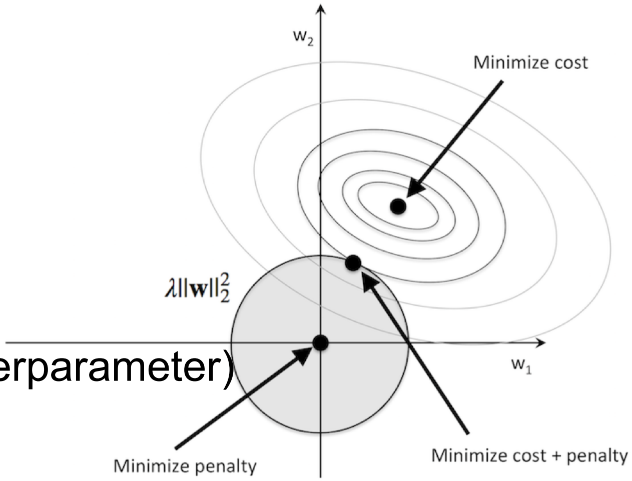
High Variance

Norm Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_i L_i\left(f(x_i, W), y_i\right)}_{\text{Data term: Predictions must match annotations}} + \underbrace{\lambda R(W)}_{\text{Regularization: } \lambda \text{ strength (hyperparameter)}}$$

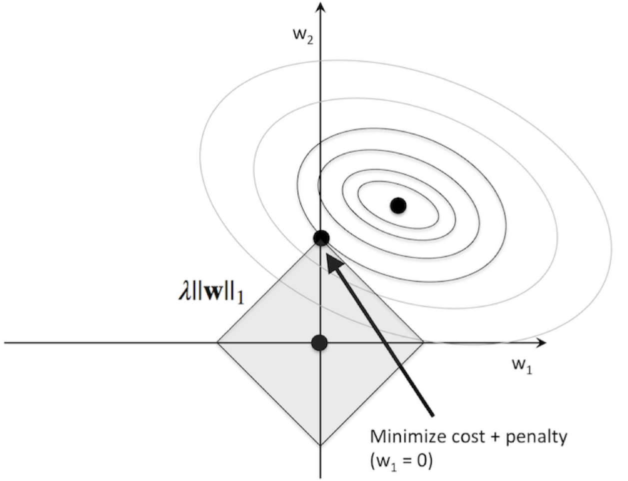
Data term: Predictions must match annotations

Regularization
λ strength (hyperparameter)



Simple regularizers

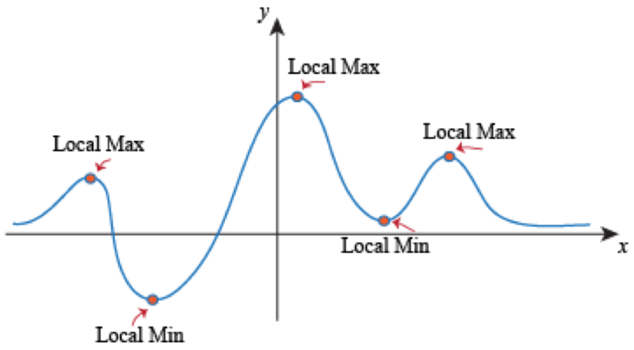
- L2 (magnitude): $R(W) = \sum_{k,l} W_{k,l}^2$
- L1 (sparsity): $R(W) = \sum_{k,l} |W_{k,l}|$
- Versions, e.g. Elastic net, Huber,…



Images from rasbt.github.io.

Gradient Descent

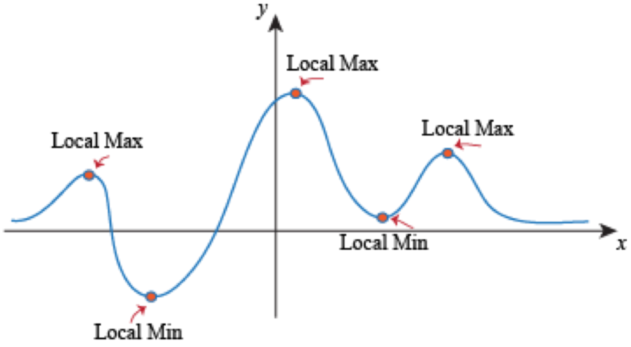
Minimize the loss function with gradient descent



Gradient Descent

Minimize the loss function with gradient descent

```
while not_converged:
    gradient = eval_gradient(loss,data,weights)
    weights += - l_rate * gradient
```



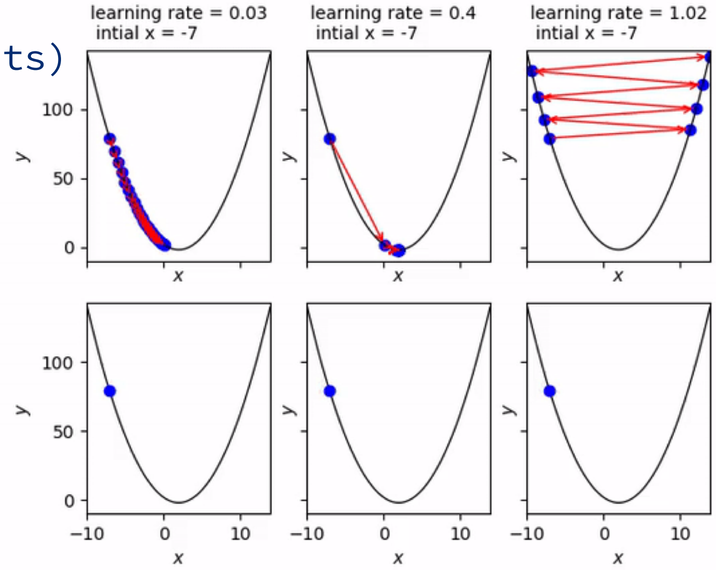
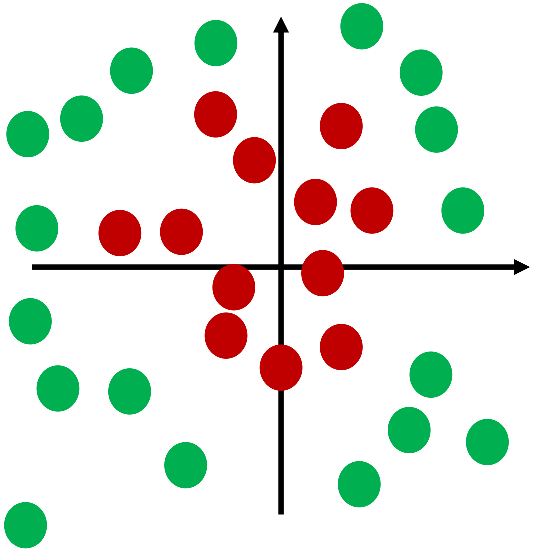


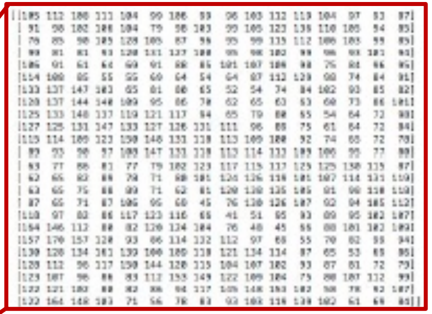
Image from jed-ai.github.io. Too large step sizes can cause divergence!

Real World Data

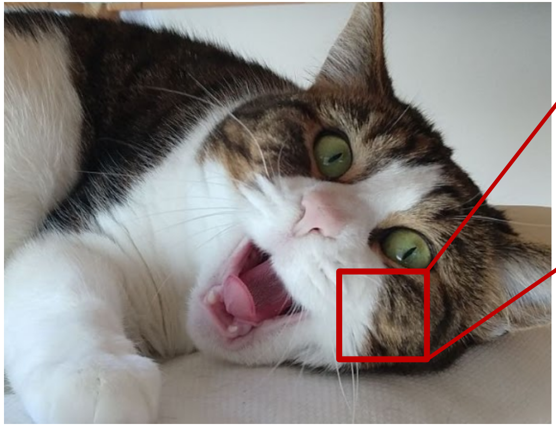


Challenges of Image Data

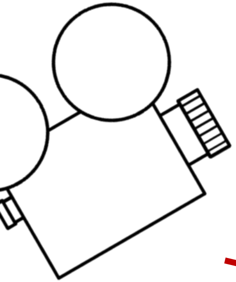
Why is this challenging?

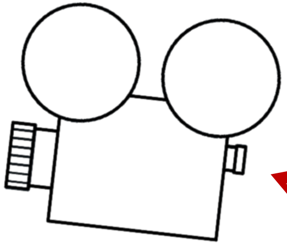


What the computer sees



What the human sees

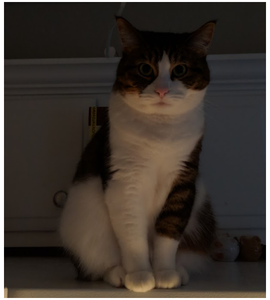
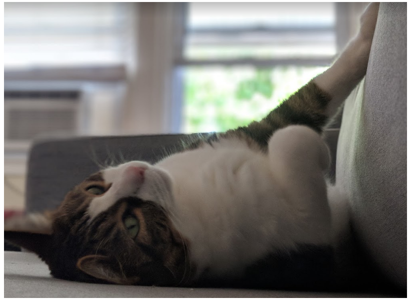




When camera moves,
all pixel values change!

Challenges of Image Data

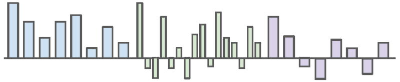
Why is this challenging?

Features

Higher-level representation via feature extraction

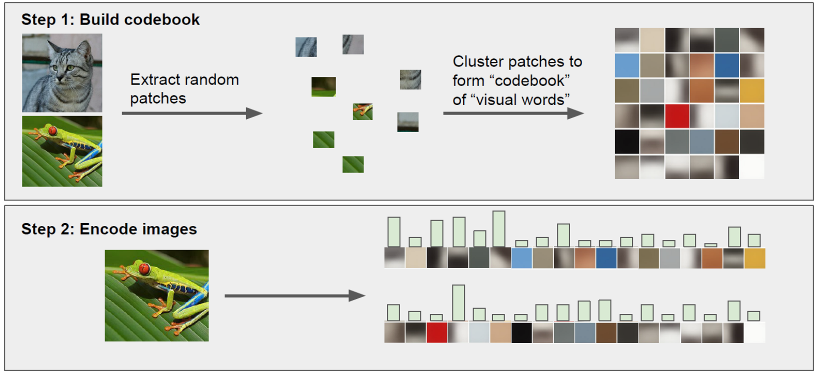
Feature extractor



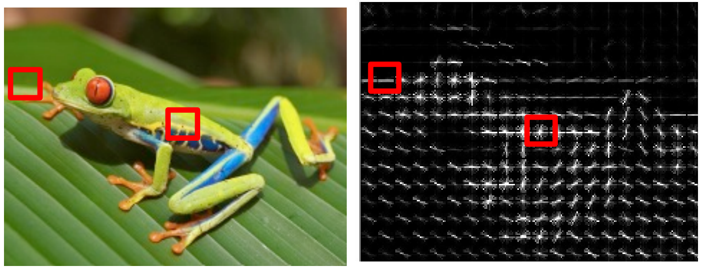
class scores

$$f(\mathbf{x},\mathbf{W})$$

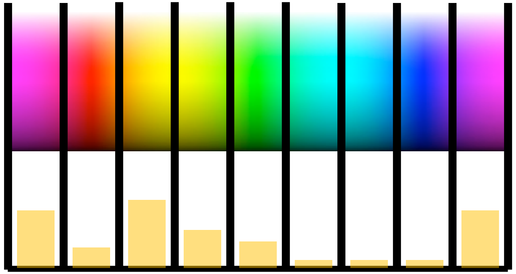
Features for Image Classification



Bag of visual words



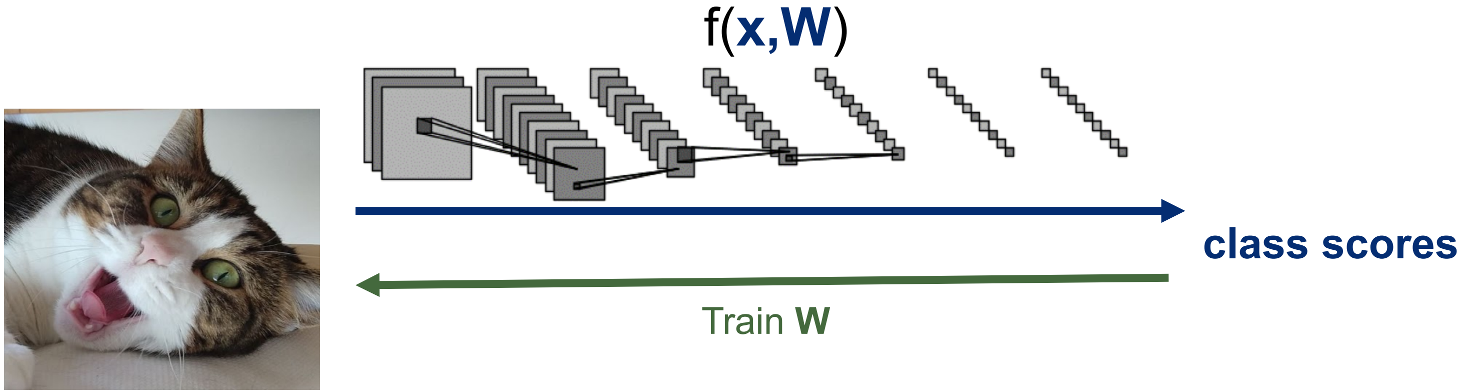
Oriented gradients



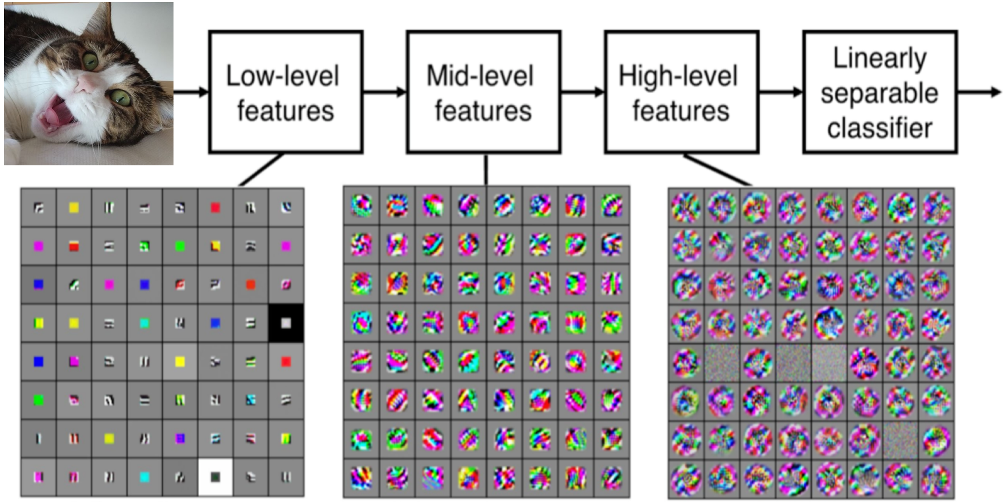
Color histogram

Images from Stanford cs231n.

Learning Feature Extractors



Learning Feature Extractors



Recap

Neural Networks

Our Deep Learning Toolbox

Components

Linear classifiers $f(\mathbf{x},\mathbf{W}) = \mathbf{W}\ \mathbf{x} + \mathbf{b}$

Loss Functions

Mean Squared Error
Negative Log Likelihood
Hinge Loss
Norm regularization

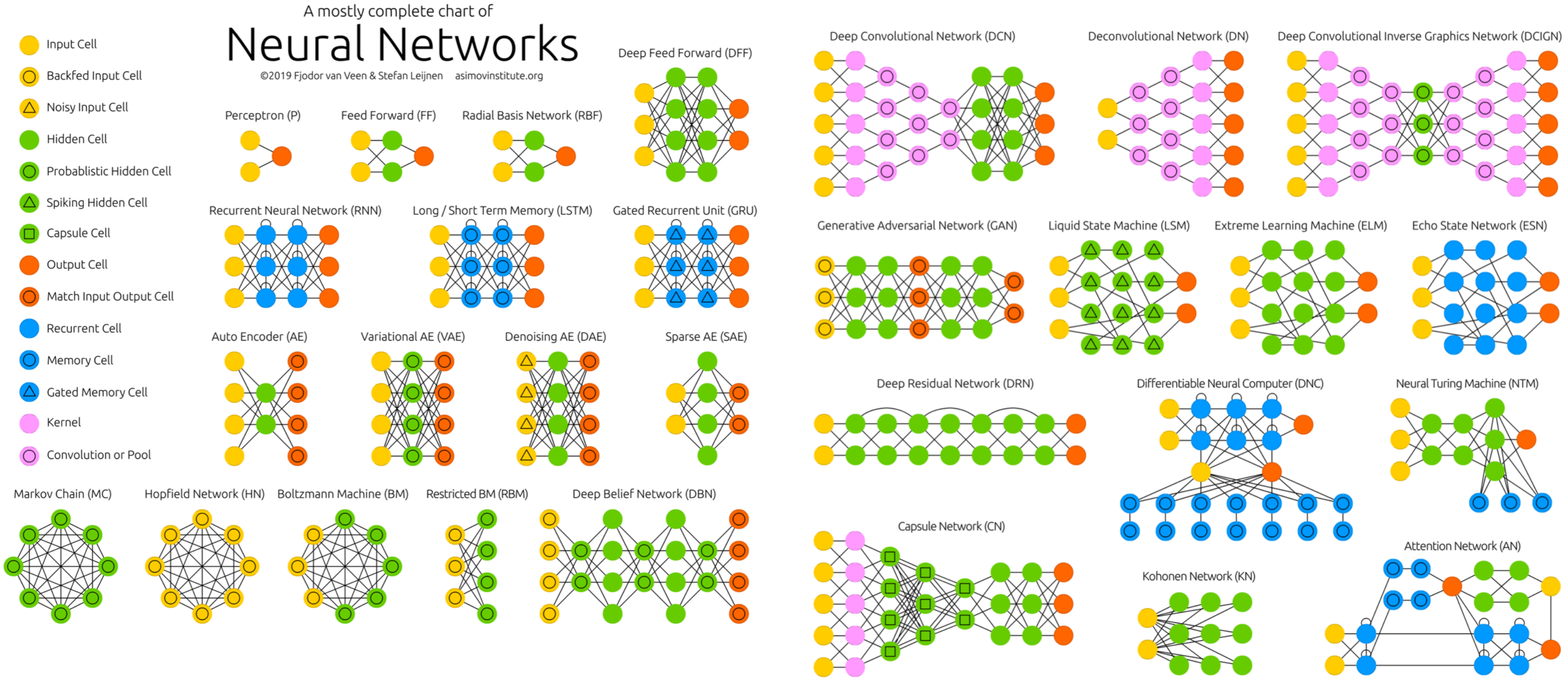
Activations

Softmax
Sigmoid

Optimization

Gradient Descent

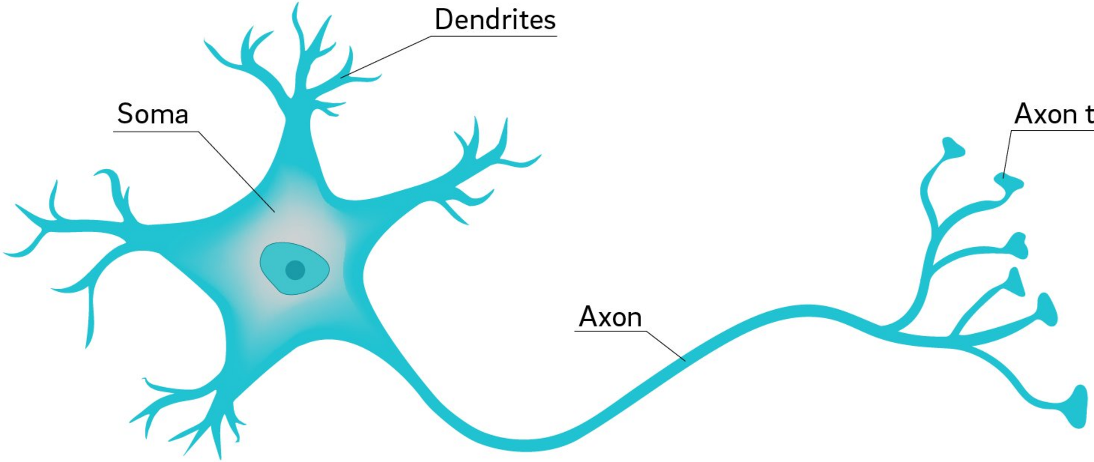
Neural Network Zoo (2016)



https://www.asimovinstitute.org/neural-network-zoo/

Neuron

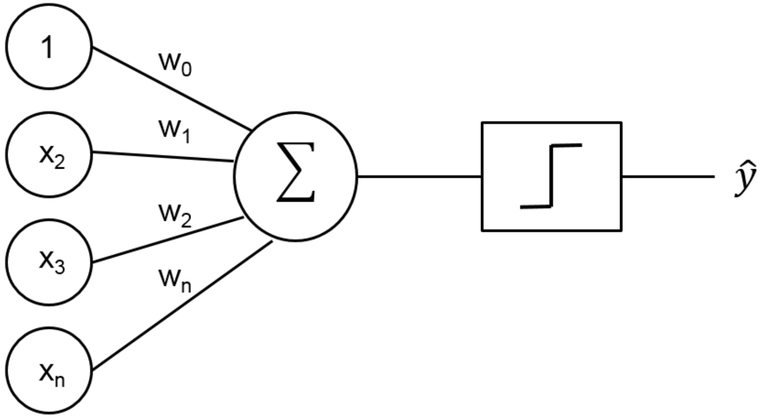
- Neural networks imply connections to neurons



- Signal comes in through dendrites into soma
- Then connects via the axon to other neurons
- Fires if input exceeds a certain threshold

Neuron

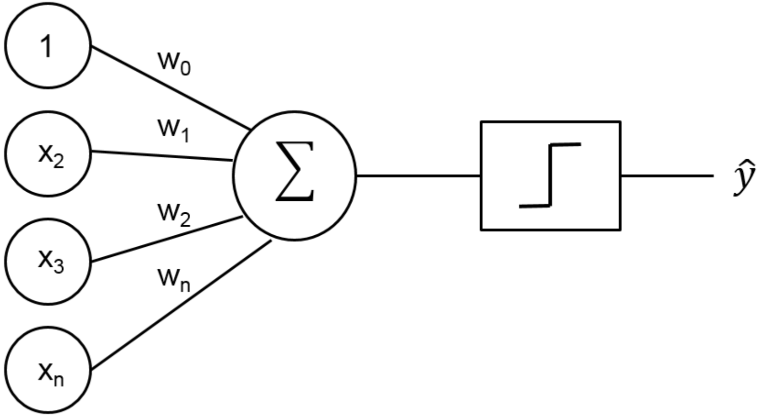
- Neural networks imply connections to neurons

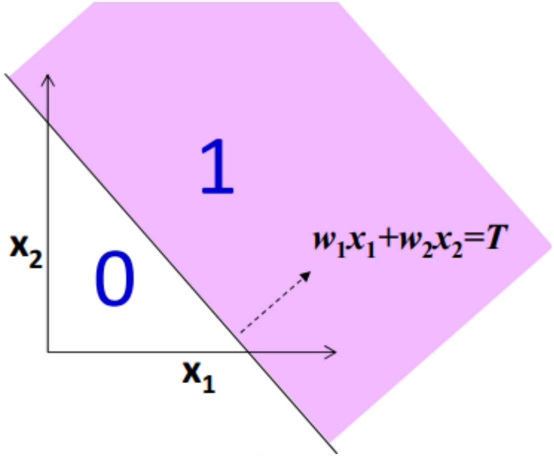


- Signal comes in through dendrites into soma
- Then connects via the axon to other neurons
- Fires if input exceeds a certain threshold

Perceptron

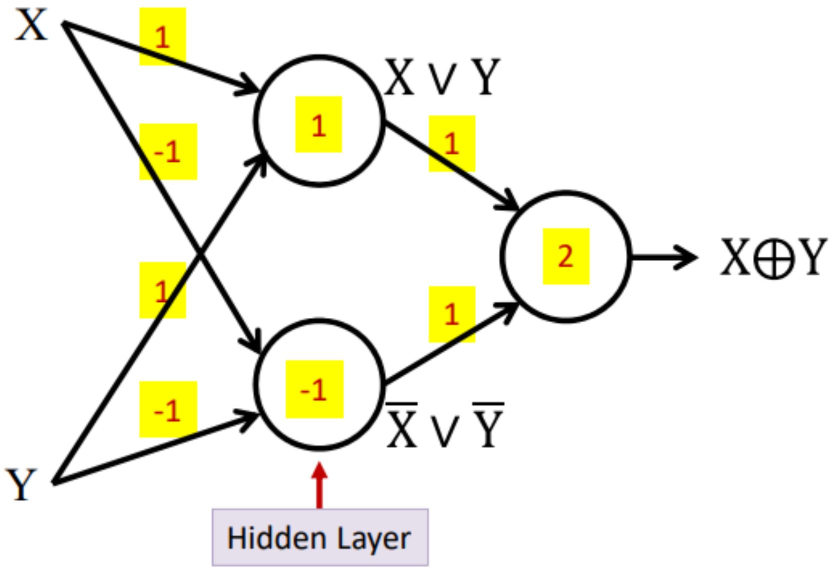
The perceptron can model almost all Boolean gates but in the end it is still a linear classifier





A Single Perceptron is Not Enough

Individual perceptrons are weak → Networked elements required!

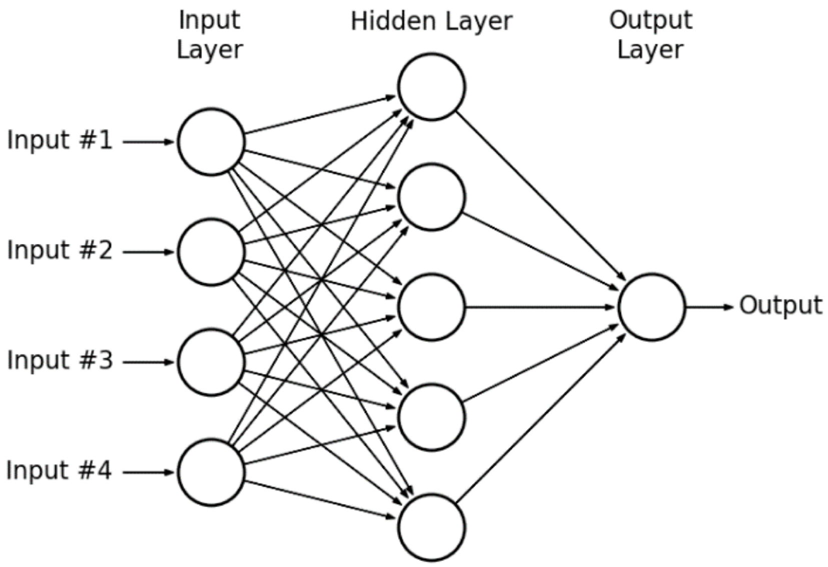


(X or Y) and (Not X or Not Y) → XOR

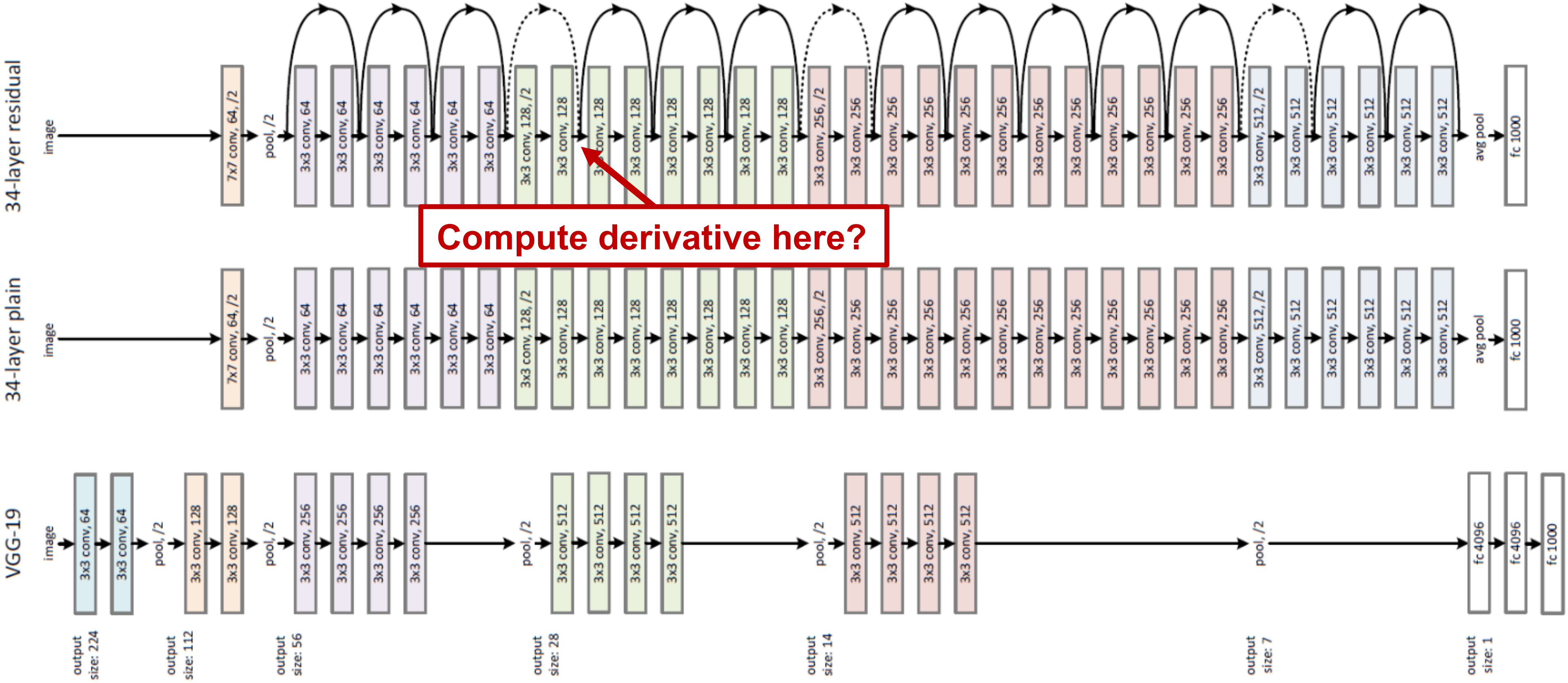
Minsky, M., & Papert, S. (1969). Perceptron Expanded Edition.

Multi-layer Perceptron

Multi-layer perceptrons can compose arbitrarily complex functions.



Computational Graphs

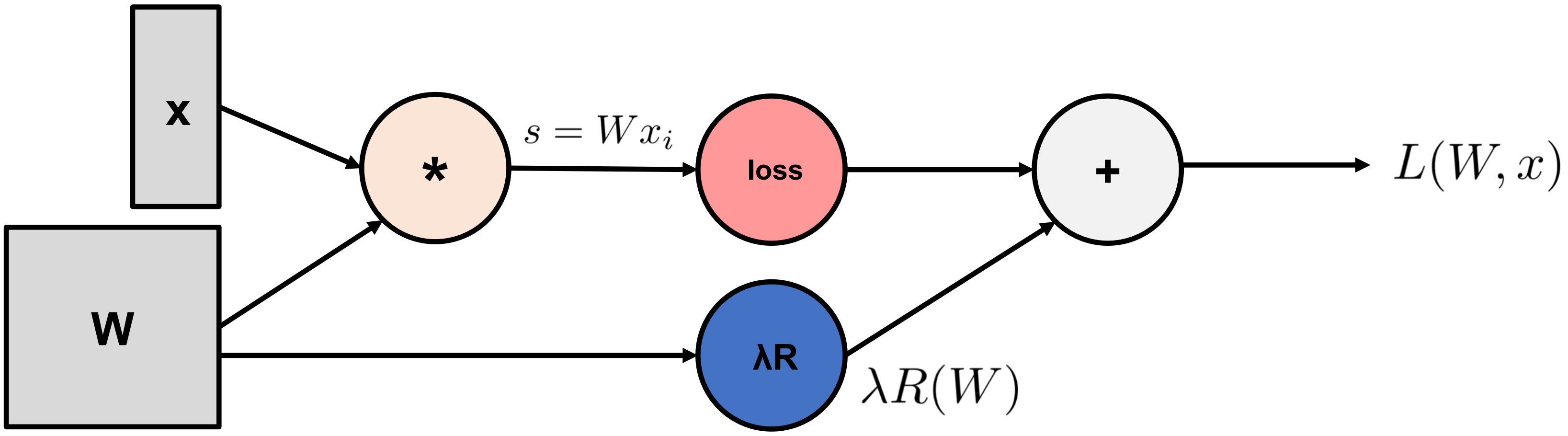


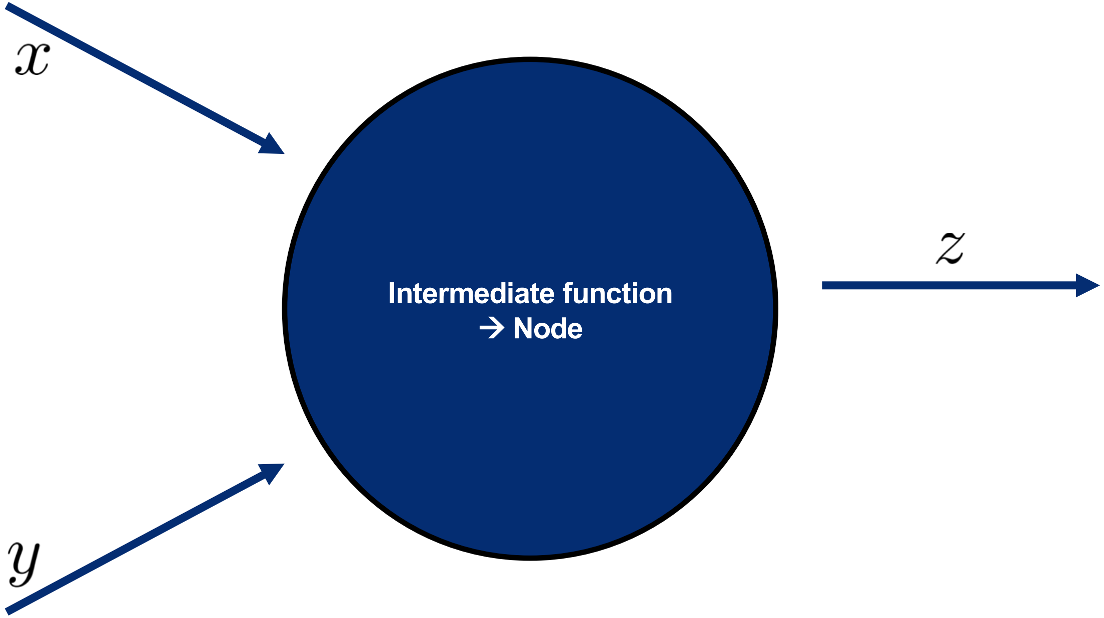
Computational Graphs

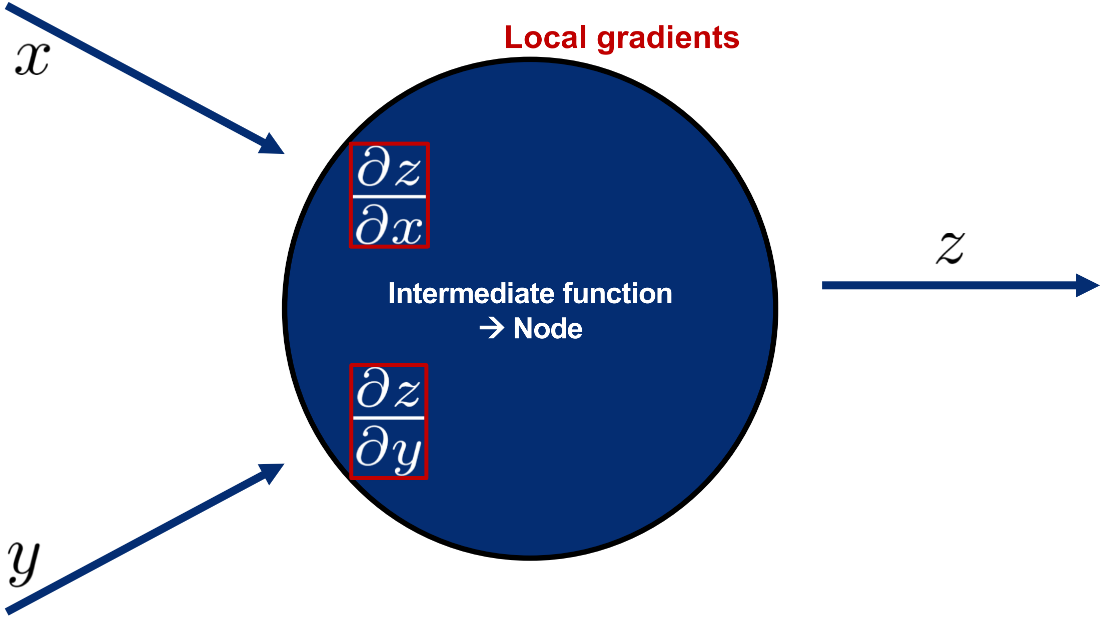
Score function: $s = f(x_i, W) = Wx_i$

Loss function: $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

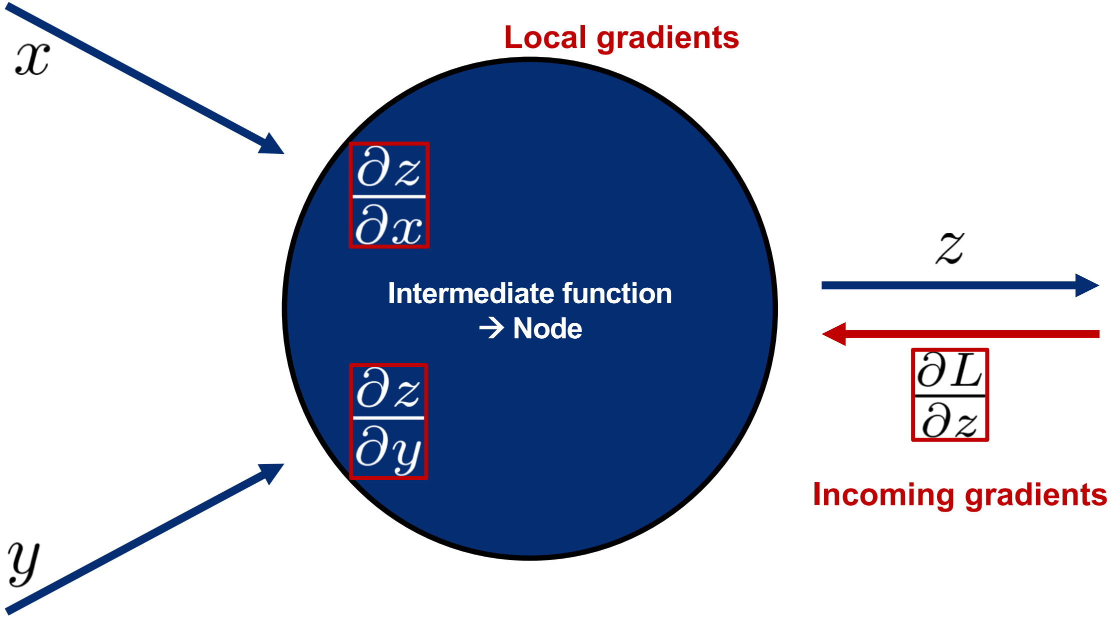
$$L(W, x) = \frac{1}{N} \sum_i L_i\left(f(x_i, W), y_i\right) + \lambda R(W)$$



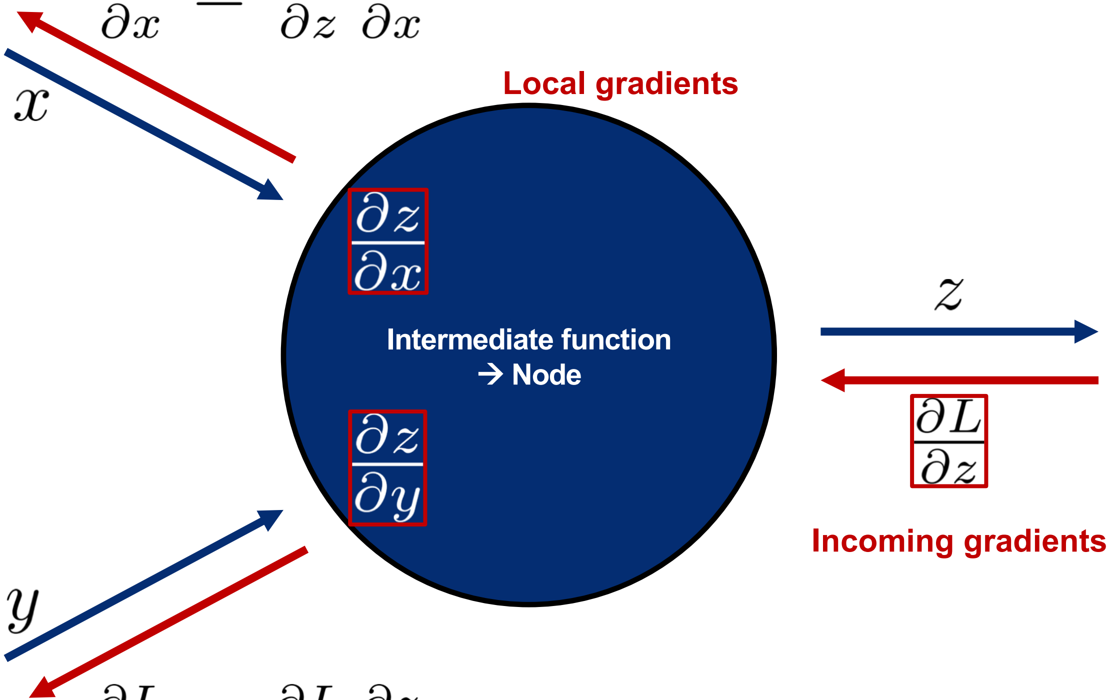

x
y
Intermediate function
→ Node
z



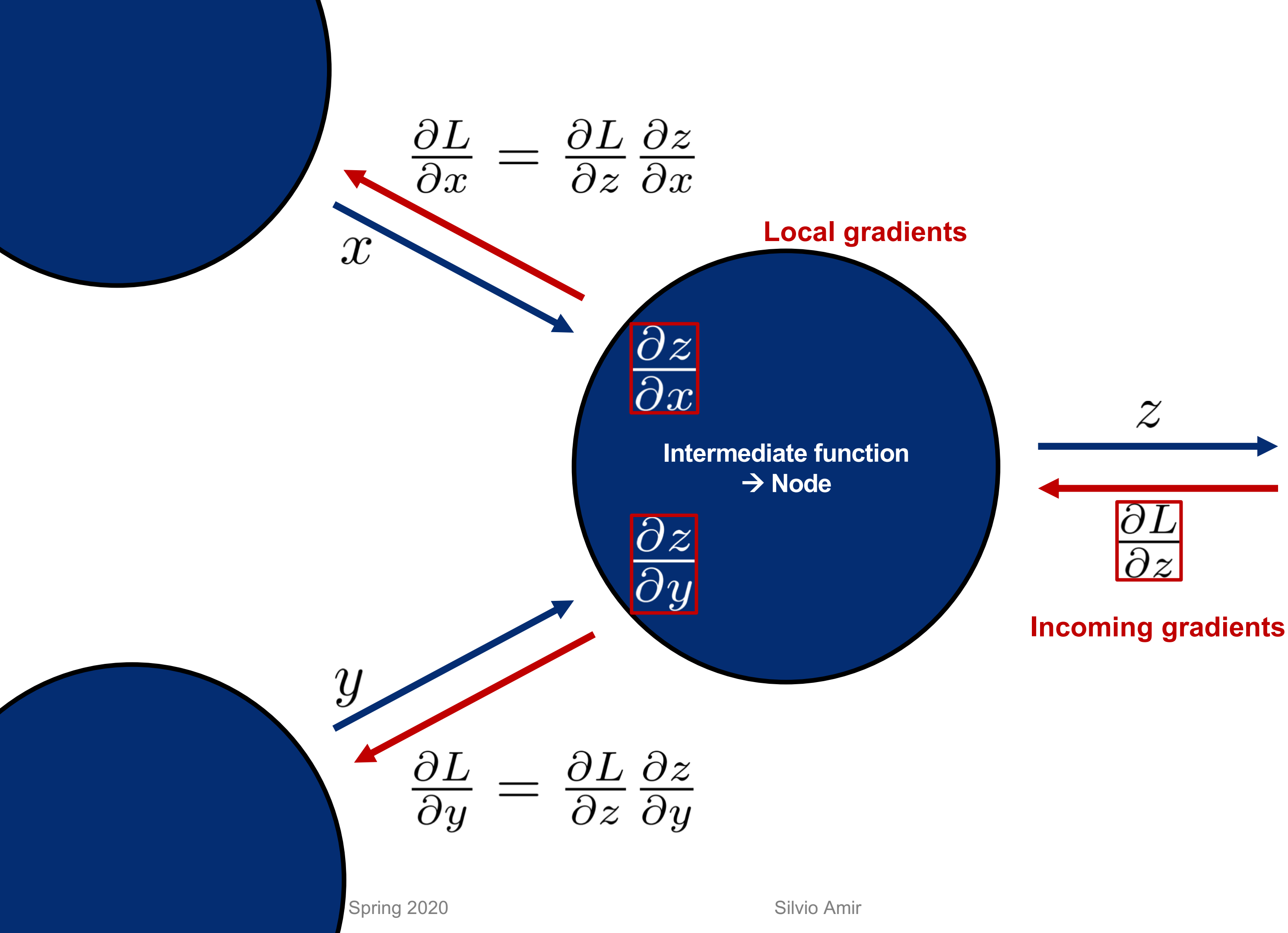
Local gradients
x
$\frac{\partial z}{\partial x}$
Intermediate function
→ Node
z
$\frac{\partial z}{\partial y}$
y


x
Local gradients
$\frac{\partial z}{\partial x}$
Intermediate function
→ Node
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
Incoming gradients
y

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$$



$$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$$



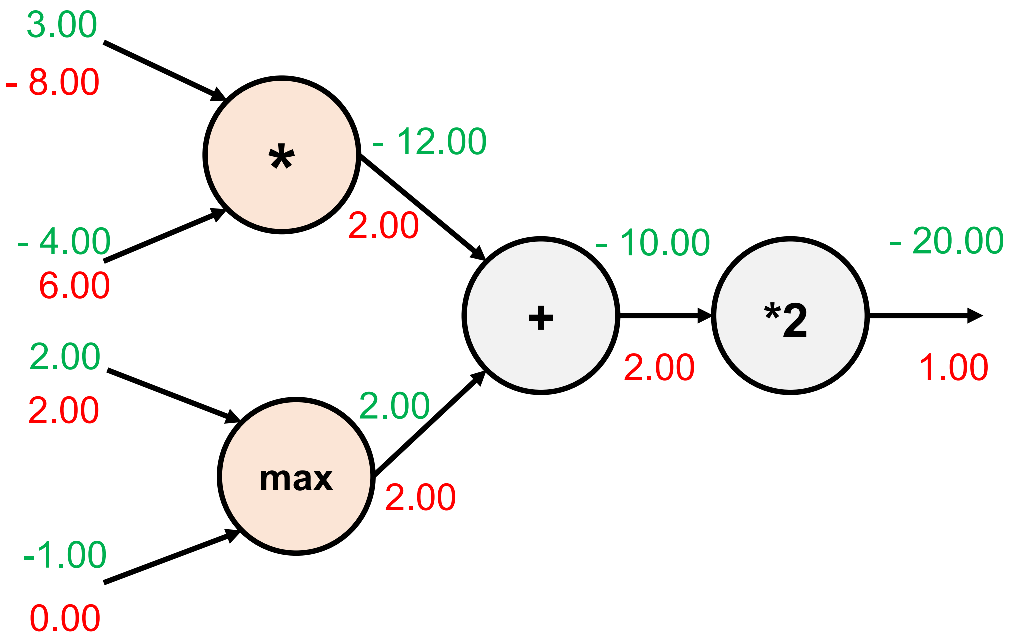
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$$
x
Local gradients
$\boxed{\frac{\partial z}{\partial x}}$
Intermediate function
→ Node
$\boxed{\frac{\partial z}{\partial y}}$
z
$\boxed{\frac{\partial L}{\partial z}}$
Incoming gradients
y
$$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$$

Patterns in Flow

Add node
→ Distributes gradient

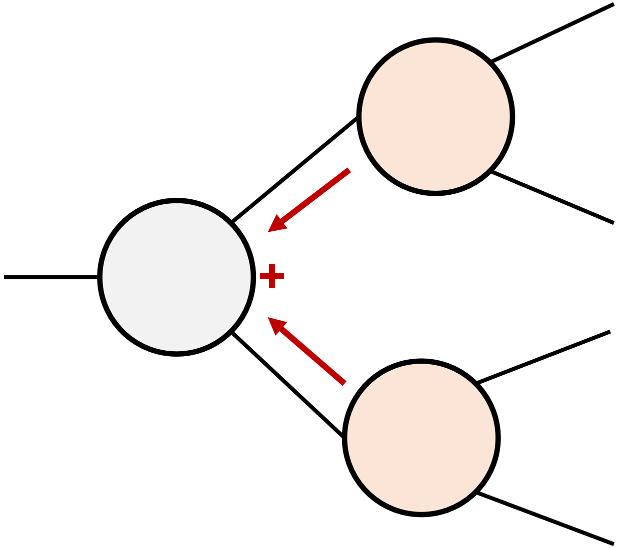
Max node
→ Routes gradient

Mul node
→ Switches gradient



Patterns in Flow

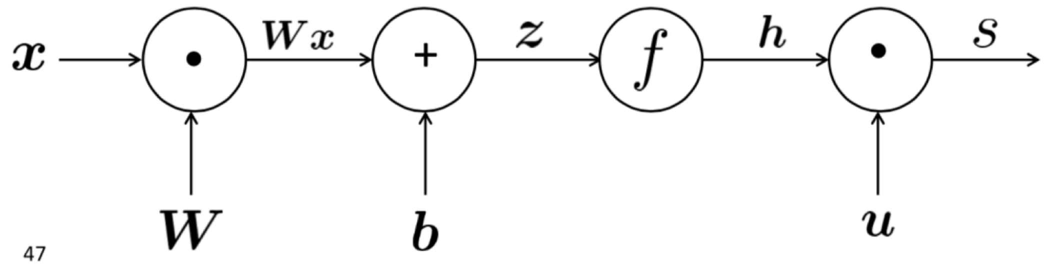
Gradients add at branches

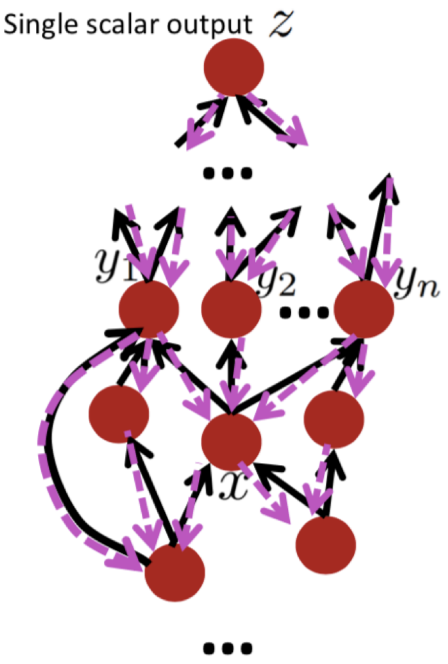


Backpropagation Algorithm

Compute derivatives of the loss wrt to all parameters of the network

- **breakdown the networks computation into small steps**
- define local derivatives of outputs wrt to inputs
- recursively apply the chain rule to compose derivatives
- re-use intermediate values computed in previous steps

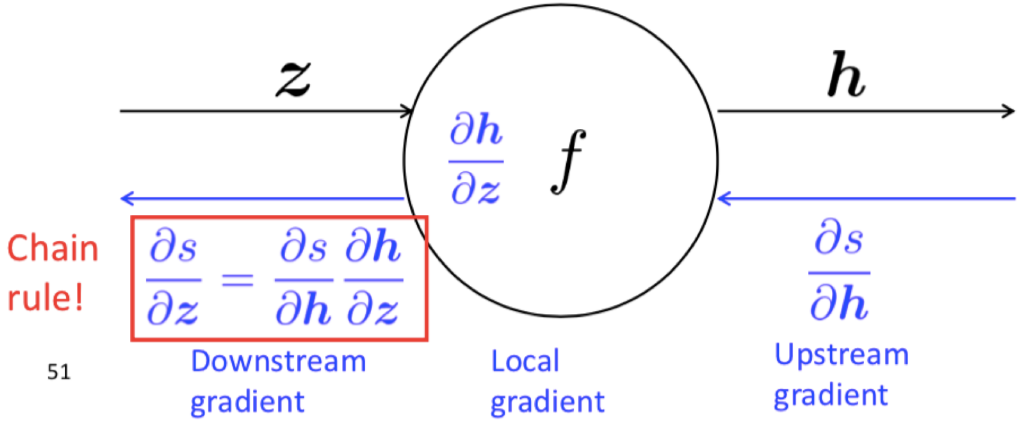




Backpropagation Algorithm

Compute derivatives of the loss wrt to all parameters of the network

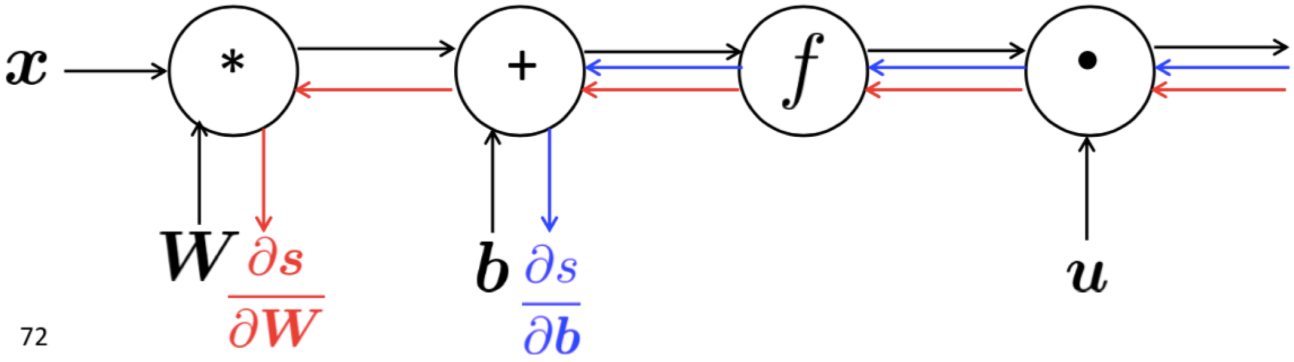
- breakdown the networks computation into small steps
- **define local derivatives of outputs wrt to inputs**
- recursively apply the chain rule to compose derivatives
- re-use intermediate values computed in previous steps



Backpropagation Algorithm

Compute derivatives of the loss wrt to all parameters of the network

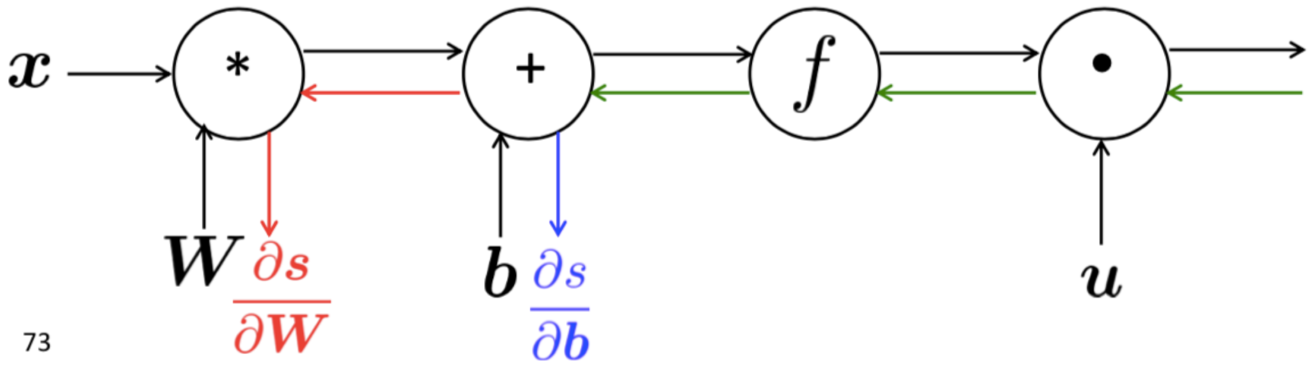
- breakdown the networks computation into small steps
- define local derivatives of outputs wrt to inputs
- **recursively apply the chain rule to compose derivatives**
- re-use intermediate values computed in previous steps



Backpropagation Algorithm

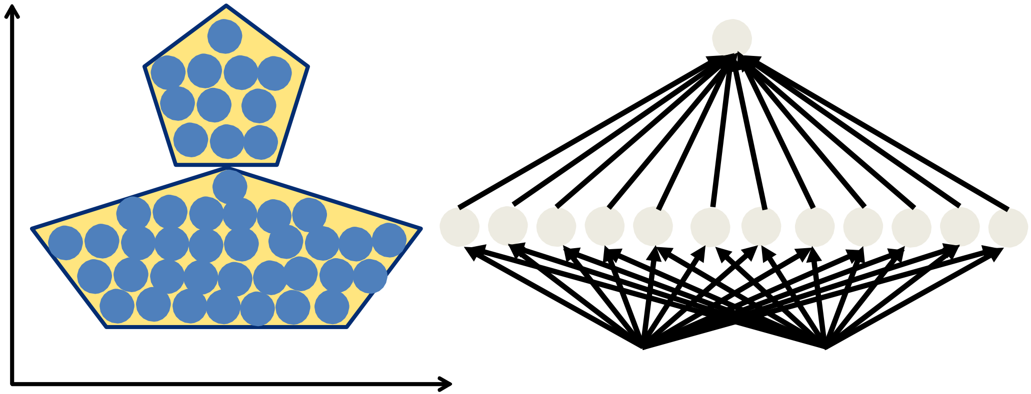
Compute derivatives of the loss wrt to all parameters of the network

- breakdown the networks computation into small steps
- define local derivatives of outputs wrt to inputs
- recursively apply the chain rule to compose derivatives
- **re-use intermediate values computed in previous steps**



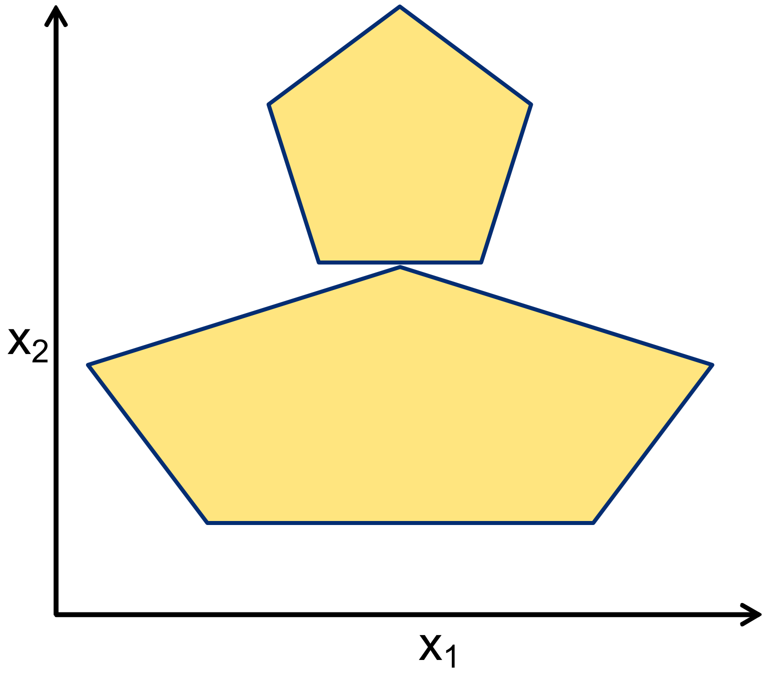
Multilayer Perceptron

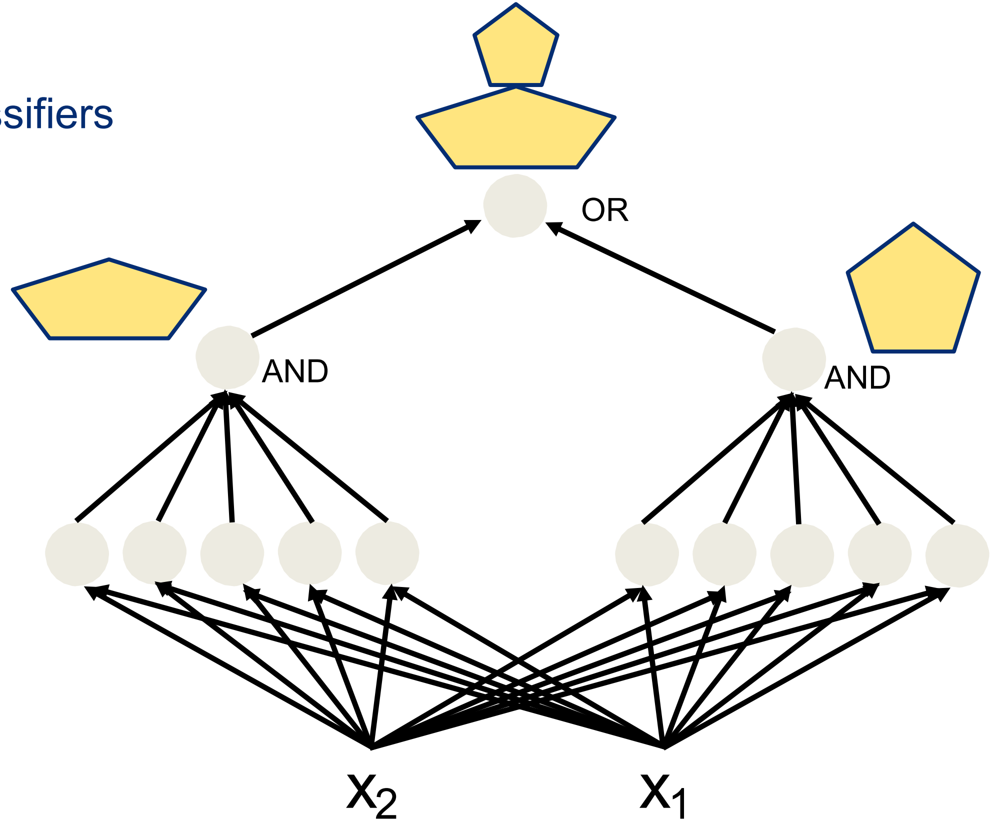
Multilayer Perceptrons are Universal Function Approximators



Multilayer Perceptron

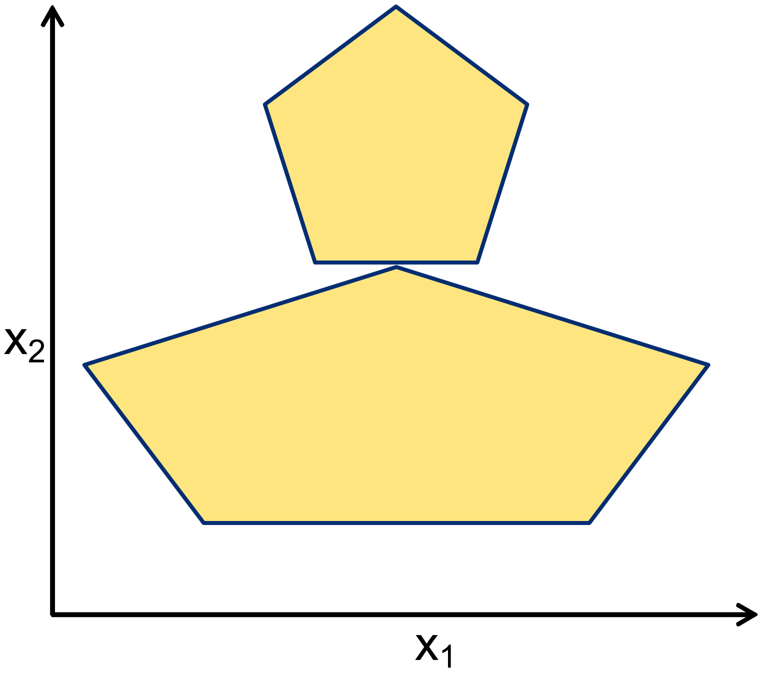
Multilayer Perceptrons are Universal Classifiers

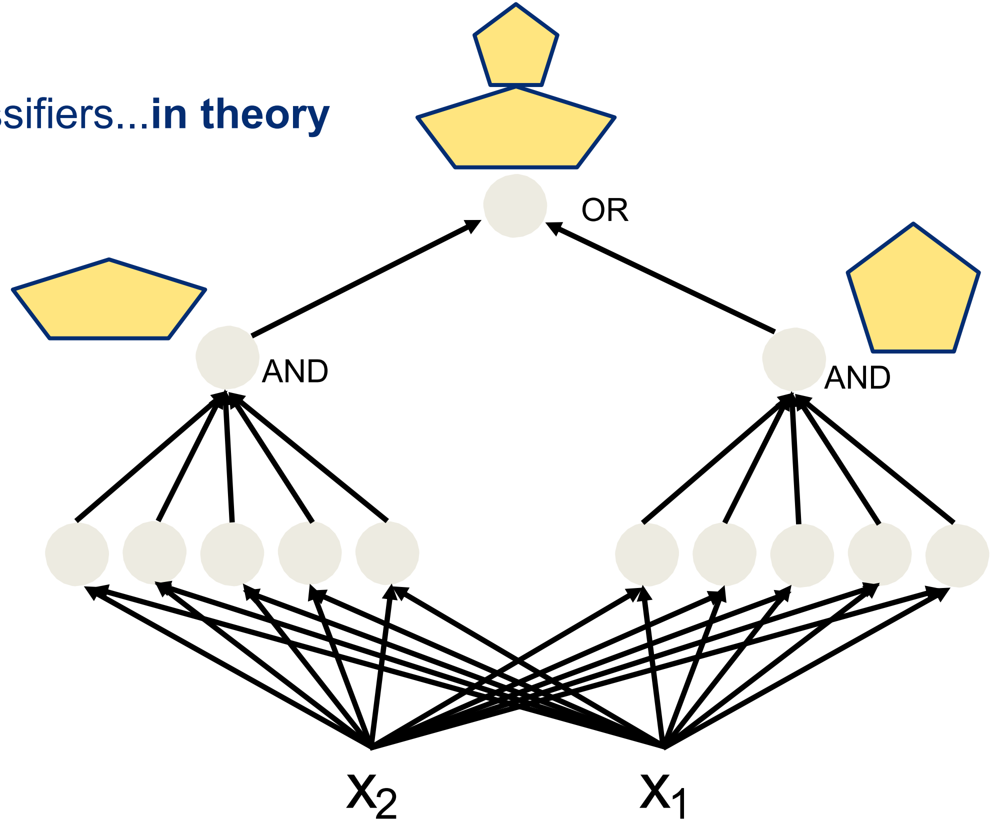




Multilayer Perceptron

Multilayer Perceptrons are Universal Classifiers...**in theory**



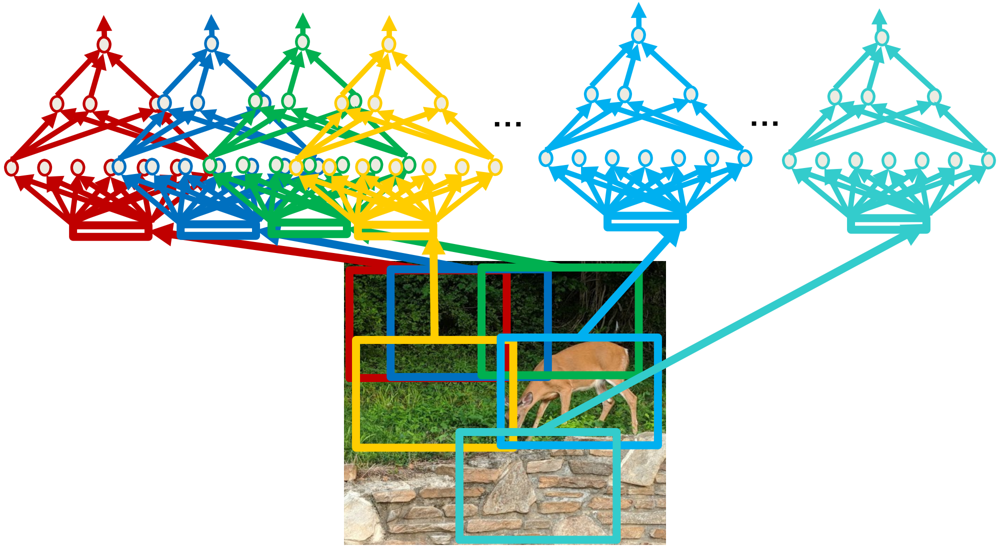


Pattern Recognition

Conventional MLPs are sensitive to location, but often the **location** of a pattern **is not important**!

Scanning with a MLP

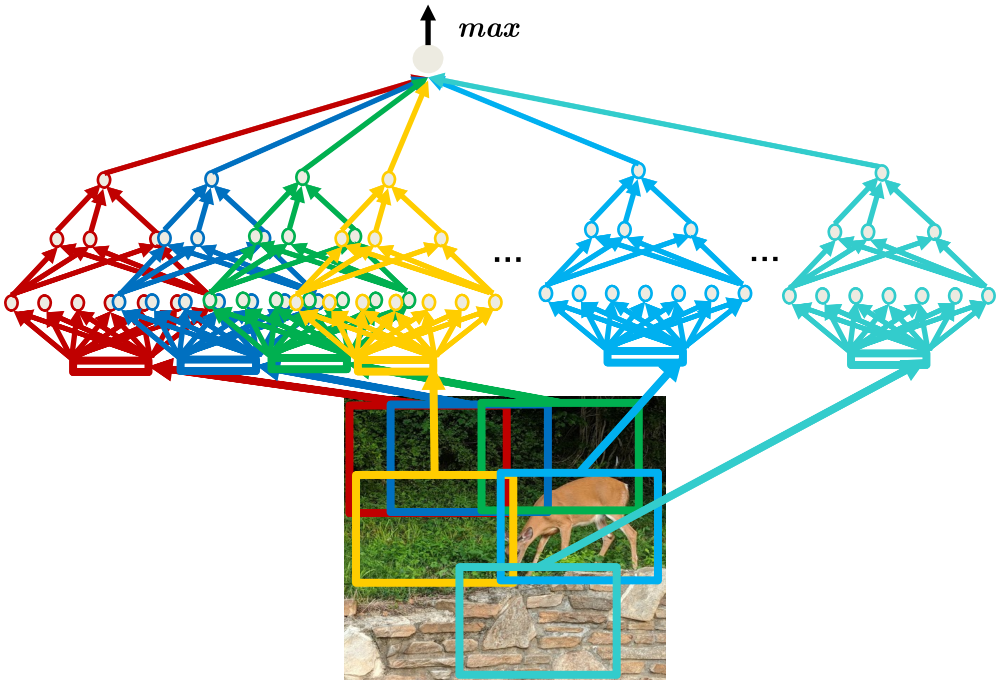
Shift invariance can be achieved by scanning with a MLP



Scanning with a MLP

Shift invariance can be achieved by scanning with a MLP

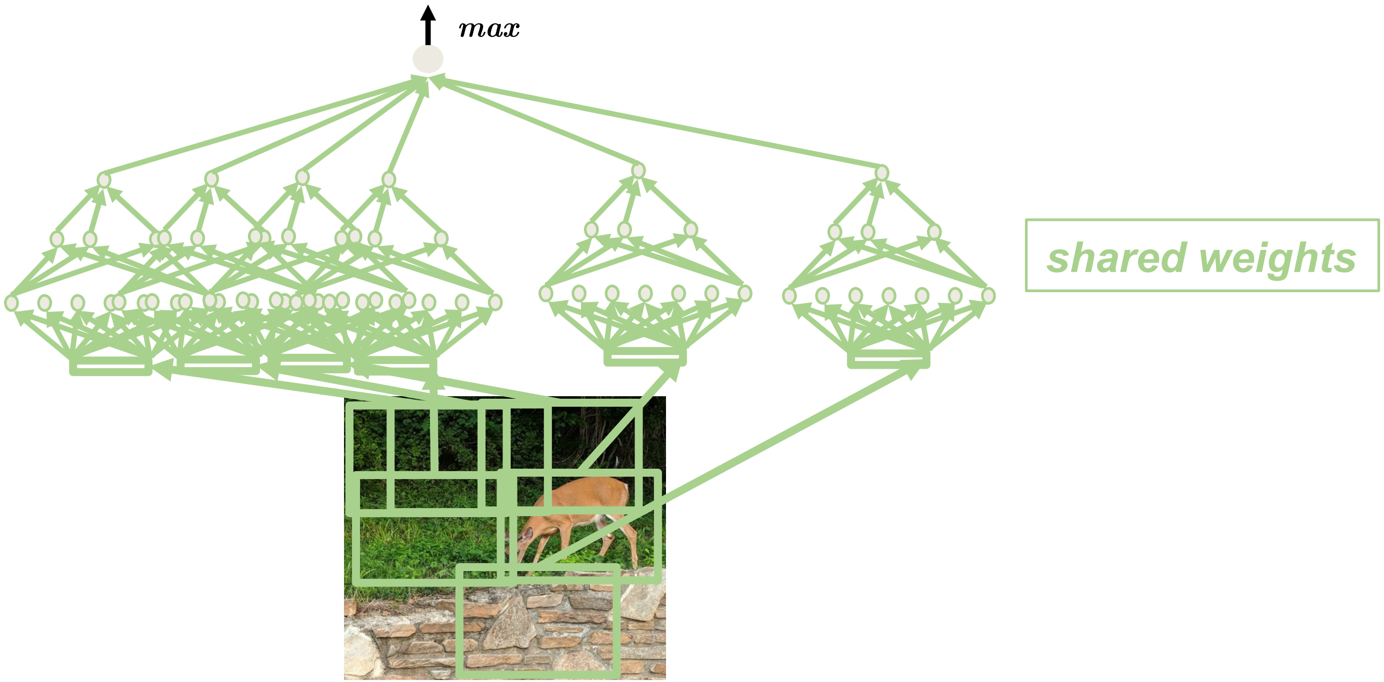
Combine multiple local predictions



Scanning with a MLP

Shift invariance can be achieved by scanning with a MLP

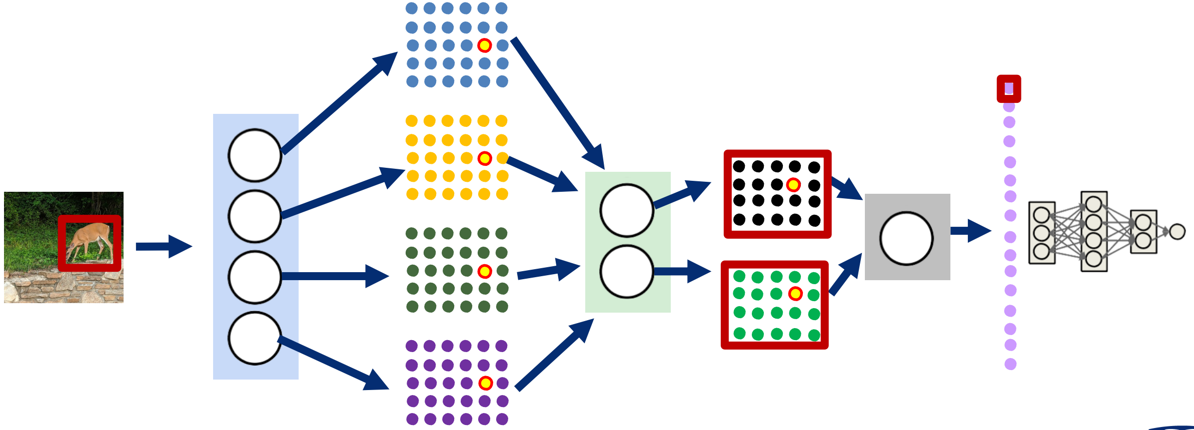
Combine multiple local predictions



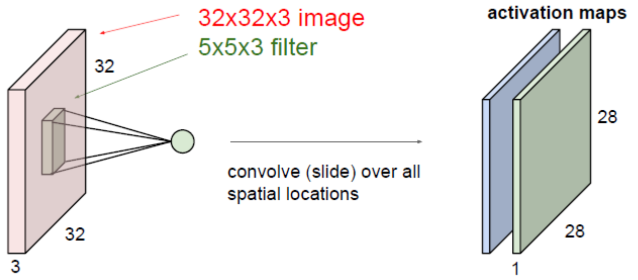
Scanning with a MLP

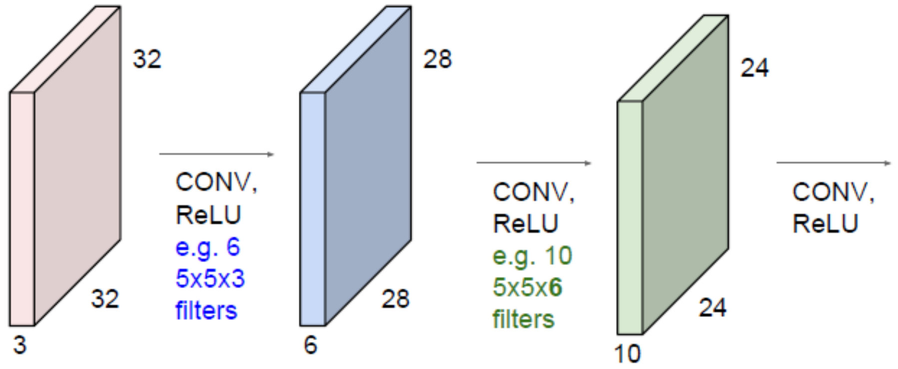
Better approach: scan the input one layer at a time

- Hierarchical features
- Compositionality



Convolutional Layers





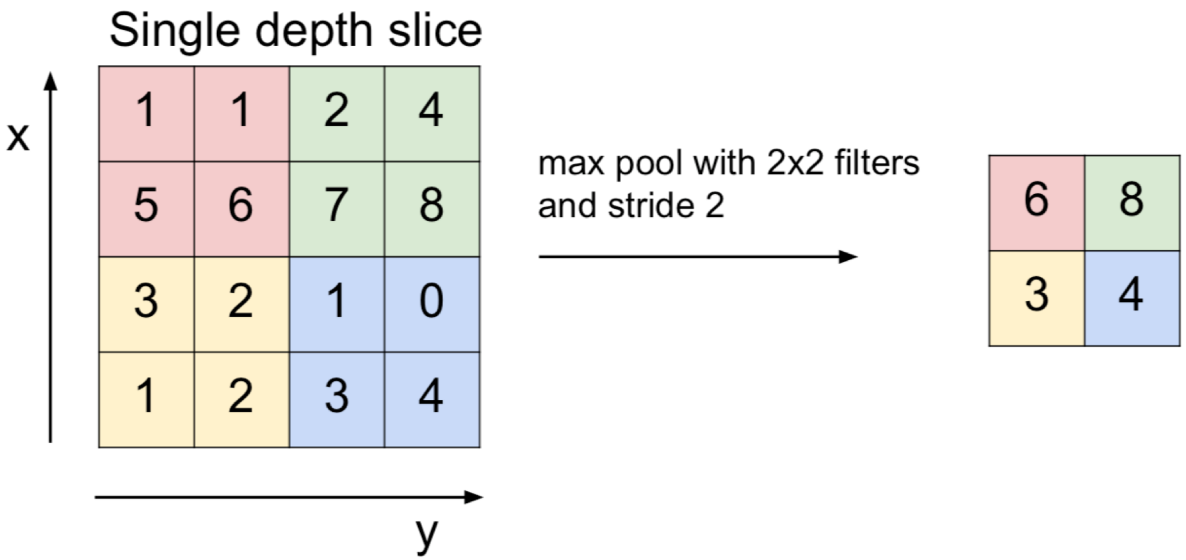
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

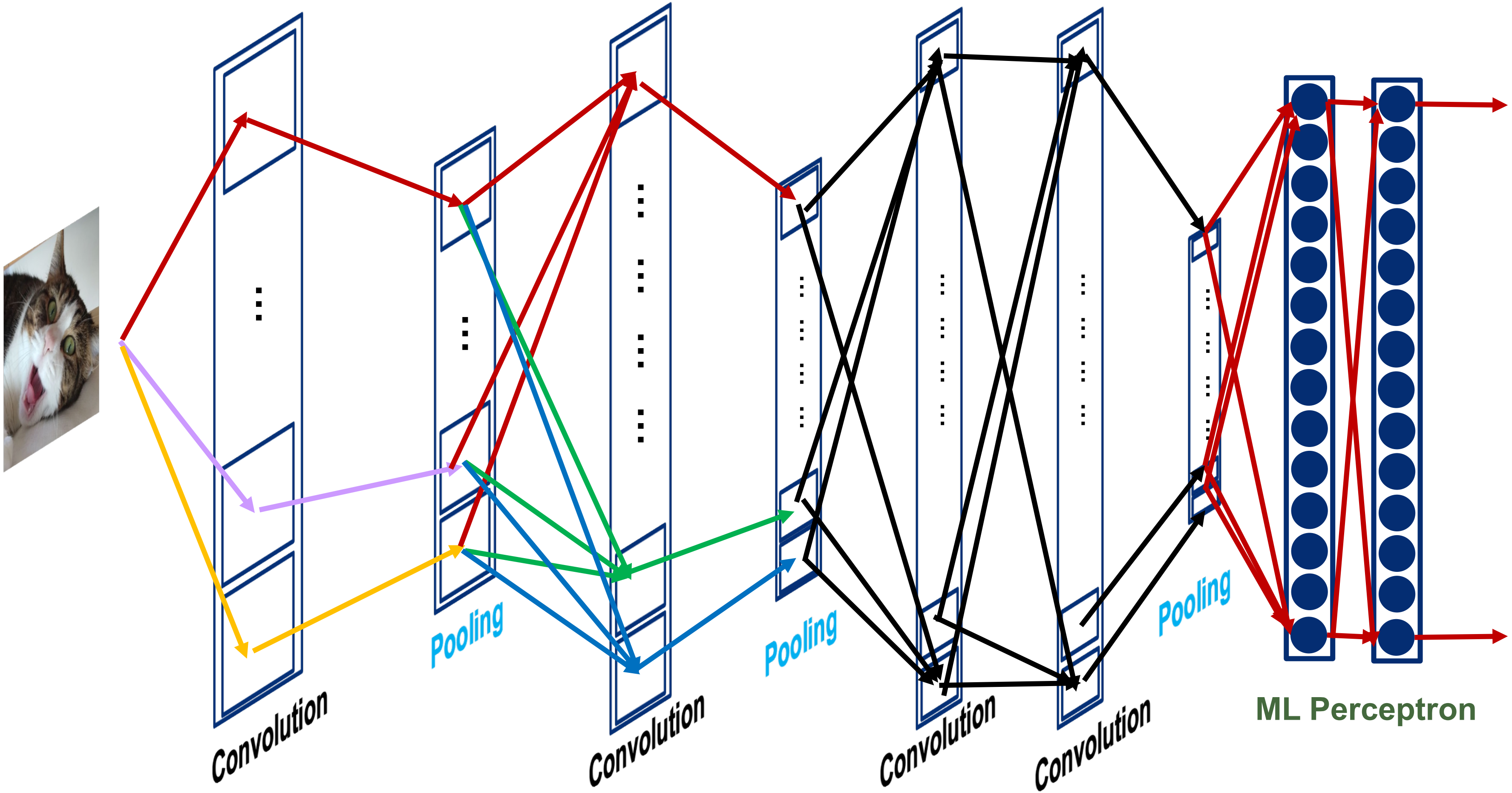
Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Pooling Layers

Max Pooling



Convolutional Neural Networks



Our Deep Learning Toolbox

Components

Linear classifiers f(**x**,**W**) = **W x + b**

Loss Functions

Mean Squared Error
Negative Log Likelihood
Hinge Loss
Norm regularization

Activations

Softmax
Sigmoid

Optimization

Gradient Descent

Our Deep Learning Toolbox

Components

Linear classifiers f(**x**,**W**) = **W x + b**
Multilayer Perceptron
Fully Connected (Dense) Layers
Shared Weight Layers,
Convolutional Layers
Pooling Layers

Loss Functions

Mean Squared Error
Negative Log Likelihood
Hinge Loss
Norm regularization

Activations

Softmax
Sigmoid

Optimization

Gradient Descent
Backpropagation

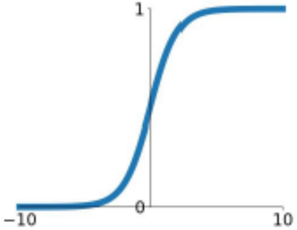
Recap

Deep Neural Networks

The Zoo of Common Activation Functions

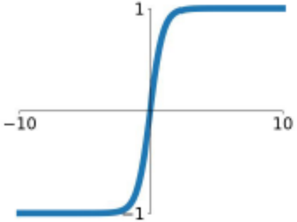
Sigmoid

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



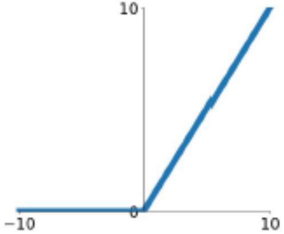
tanh

$$\tanh(x)$$



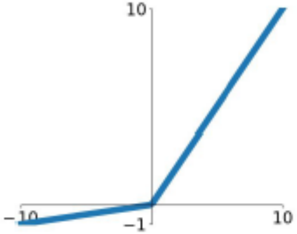
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

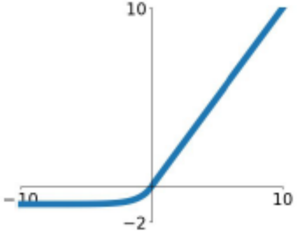


Maxout

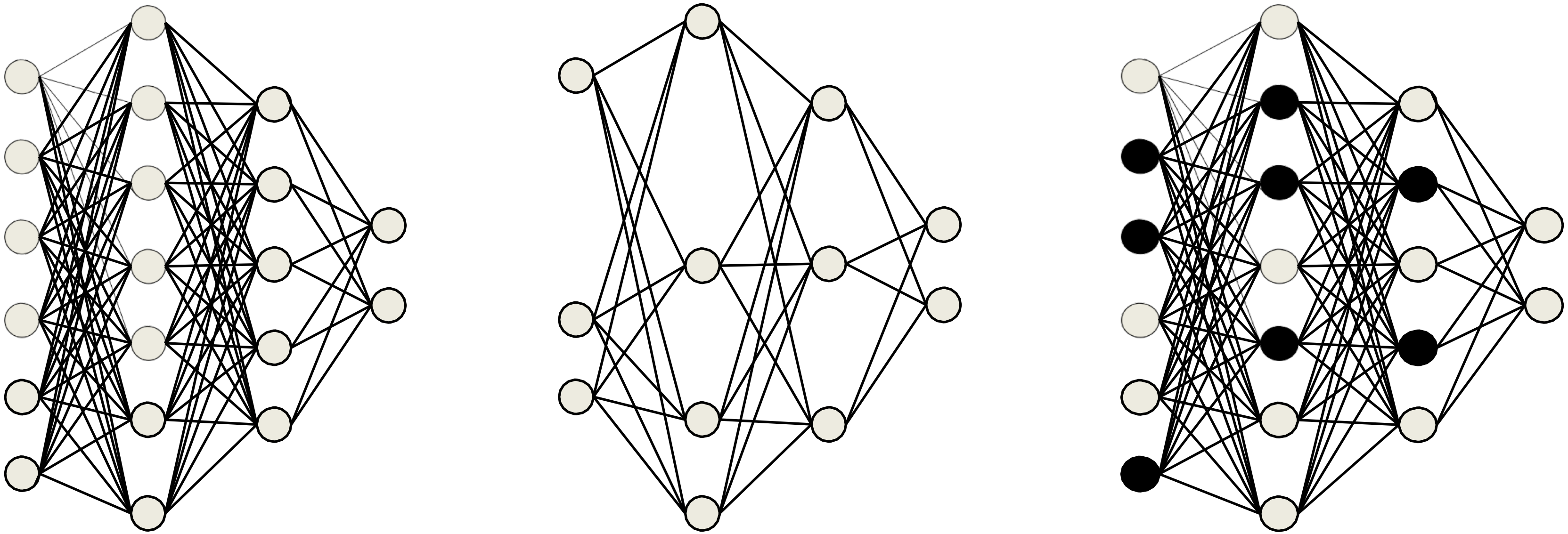
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Dropout

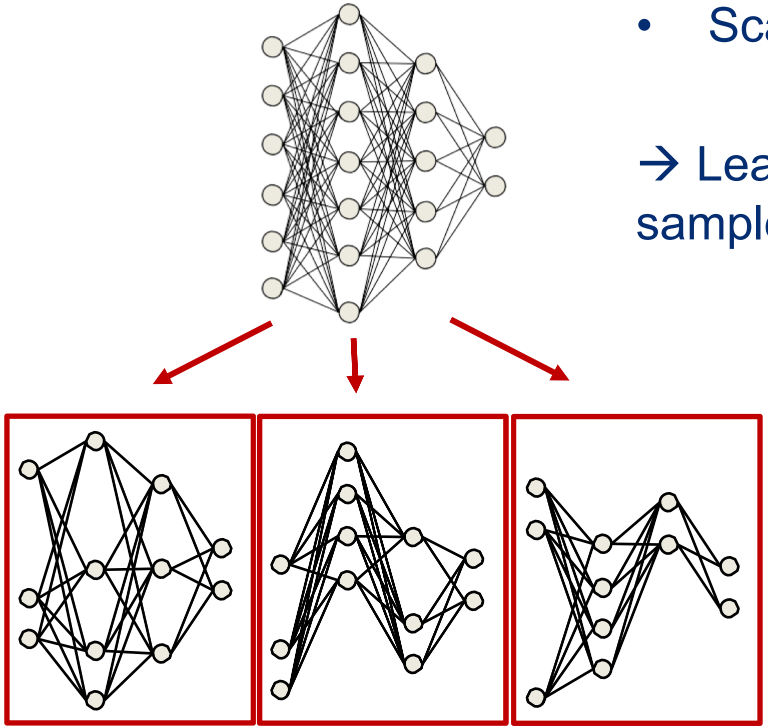


During training

- At each iteration, in each layer, “knock out” each neuron with probability 1-α
- In practice, we do not drop connections but set inputs/outputs to zero

[Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. JMLR, 15(1), 1929-1958.]

Dropout



Inference

- Scale weights or activations

→ Learns a N-neuron network that averages over a sample of 2^N possible sub-networks

Batch Normalization

1. Compute empirical mean and variance for each channel

$$E[x^{(k)}], \mathrm{Var}[x^{(k)}]$$

2. Normalize to unit Gaussian

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

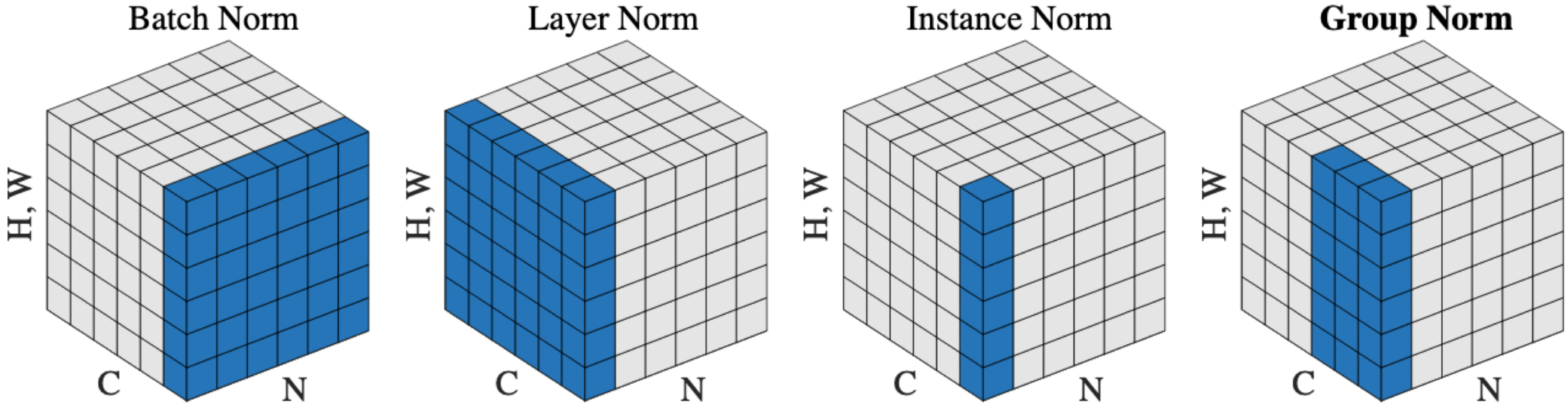
3. Squash output to beneficial range

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

These are parameters and are learned during training.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.

Normalization Methods

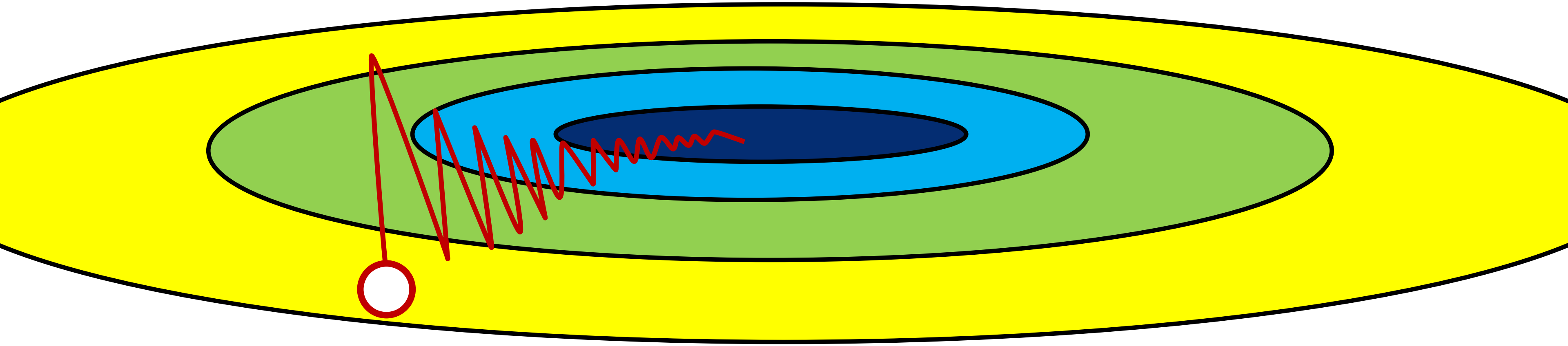


Wu, Y. and He, K., 2018. Group normalization. arXiv preprint arXiv: 1803.08494.

Problems with SGD Optimization

→ Slow progress along shallow dimension, jitter along the other

→ Problematic: Neural networks have millions of parameters!

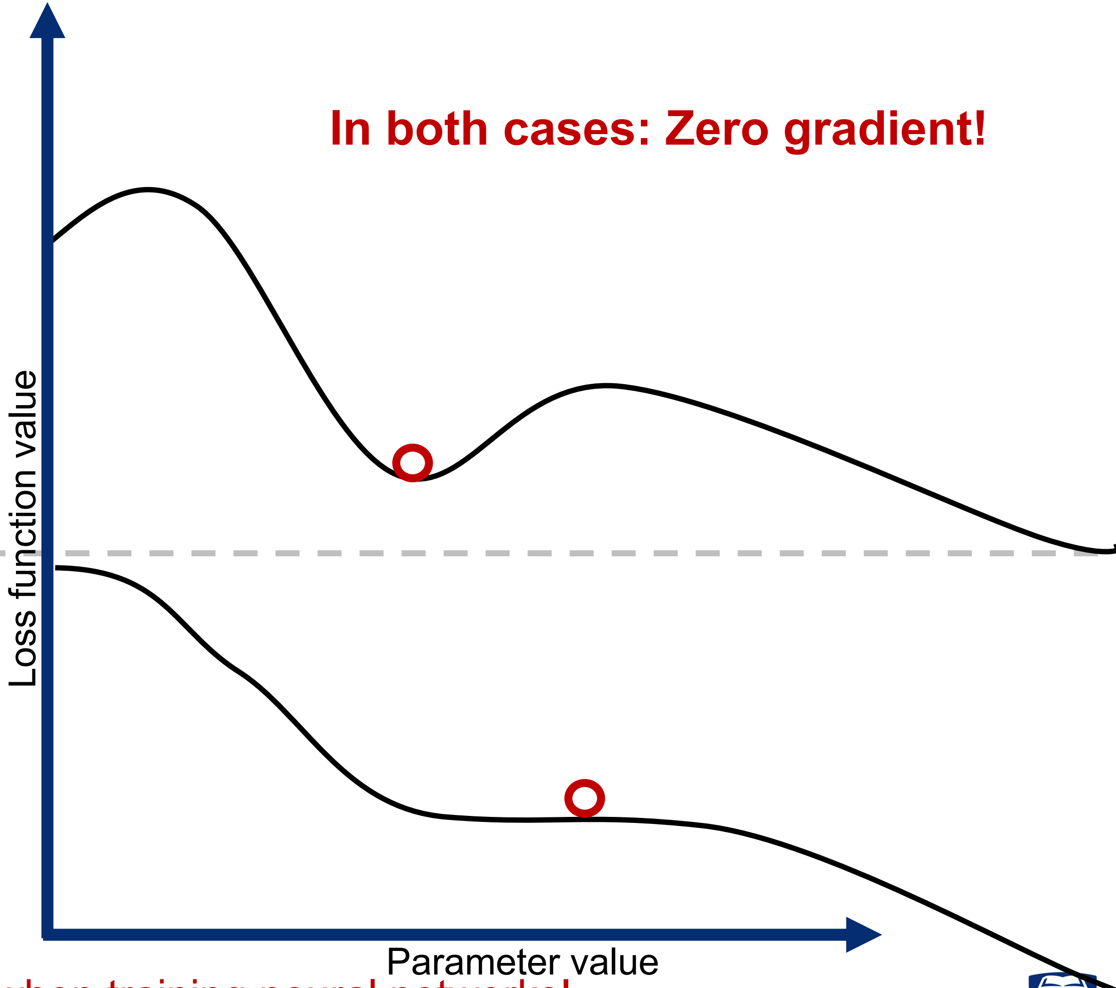


Why is this?

Condition number: Ratio of largest to smallest singular value of the Hessian

→ If large, then loss function at this point badly conditioned

Another Problem



Local minimum

In every direction, loss will go up.

Saddle point

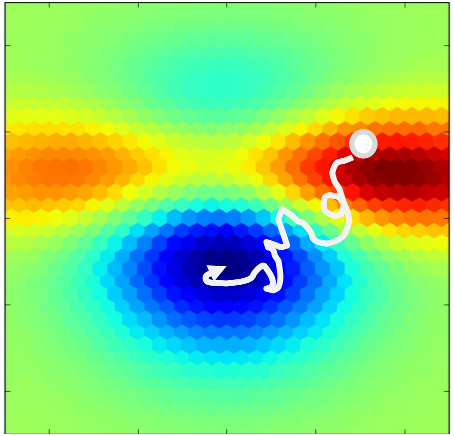
In some direction loss will go up,
in other direction loss will go down.

In high dimensional space, this scenario is much more common.

→ Saddle points are the big problem when training neural networks!

And Another Problem

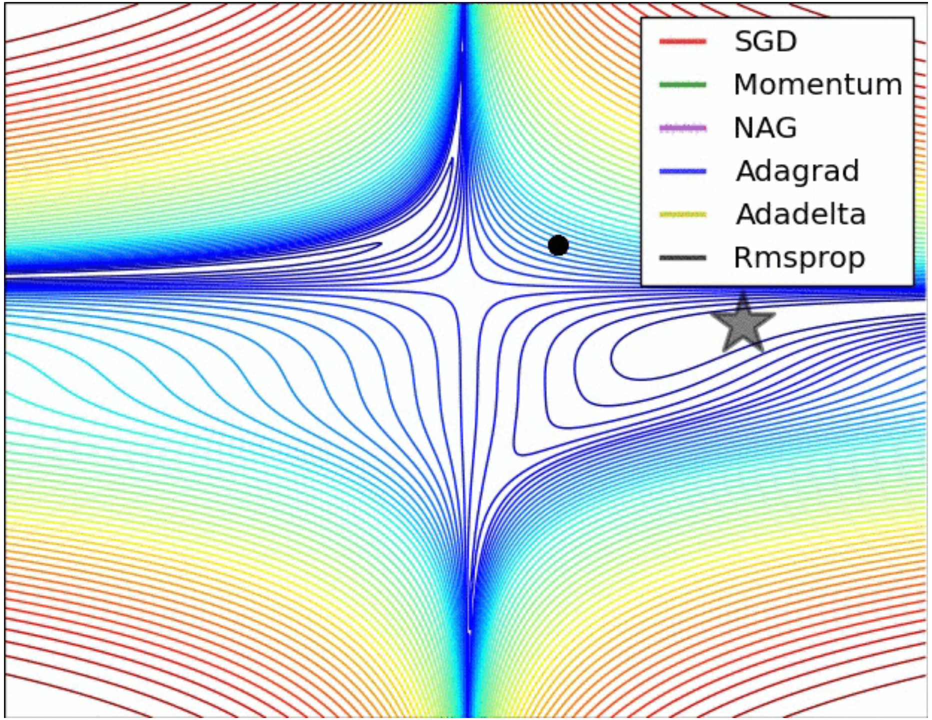
```
while not_converged:
    data_batch = sample_training_data(data, batch_size)
    gradient = eval_gradient(loss, data_batch, weights)
    weights += - step_size * gradient
```



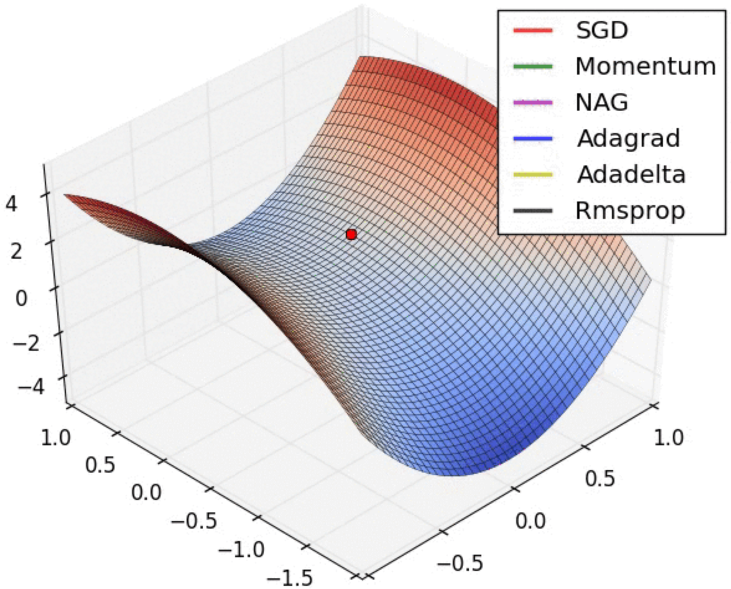
Gradient is computed over mini-batches

- Mini-batches do not necessarily represent the full dataset
- **Gradients can be noisy!**

Optimizers



SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop



SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

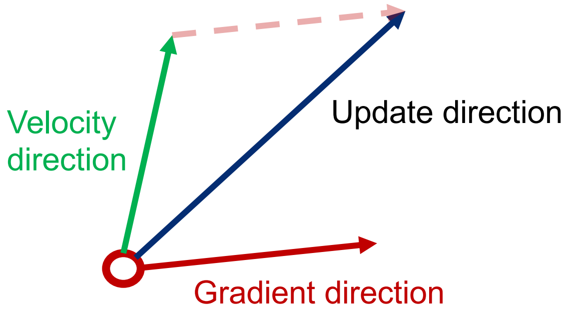
Adding Momentum

Nesterov, Y. E. (1983). A method for solving the convex programming problem with convergence rate O (1/k^ 2). In *Dokl. Akad. Nauk SSSR* (Vol. 269, pp. 543-547).
Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013, February). On the importance of initialization and momentum in deep learning. In International conference on machine learning (pp. 1139-1147).

SGD + Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla_W L(W_t)$$
$$W_{t+1} = W_t + v_{t+1}$$

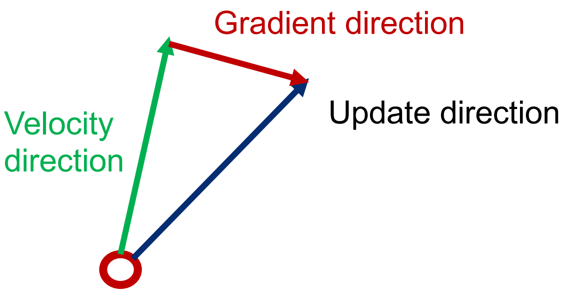
Combine gradient at current point with velocity to get update



Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla_W L(W_t + \rho v_t)$$
$$W_{t+1} = W_t + v_{t+1}$$

Evaluate gradient at where velocity would take us, then mix with velocity



AdaGrad

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 2121-2159.

$$g_t = \nabla_W L(W_t)$$

$$S_i = S_i + (g_t)_i^2 \quad \text{with } S_i(t=0) = 0$$

$$(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i} + \epsilon}(g_t)_i$$

$$\boxed{W_{t+1} = W_t - \mathrm{d}W_t}$$

1. Compute gradient
2. Compute and accumulate element-wise squared gradient
3. Compute gradient update with **parameter-wise** learning rate
4. Apply gradient update

Adam

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.

$$g_t = \nabla_W L(W_t)$$

Bias correction

$$S_i^{(1)} = \left(\rho_1 S_i^{(1)} + (1-\rho_1)(g_t)_i\right)\boxed{(1-\rho_1^t)^{(-1)}}$$

$$S_i^{(2)} = \left(\rho_2 S_i^{(2)} + (1-\rho_2)(g_t)_i^2\right)\boxed{(1-\rho_2^t)^{(-1)}}$$

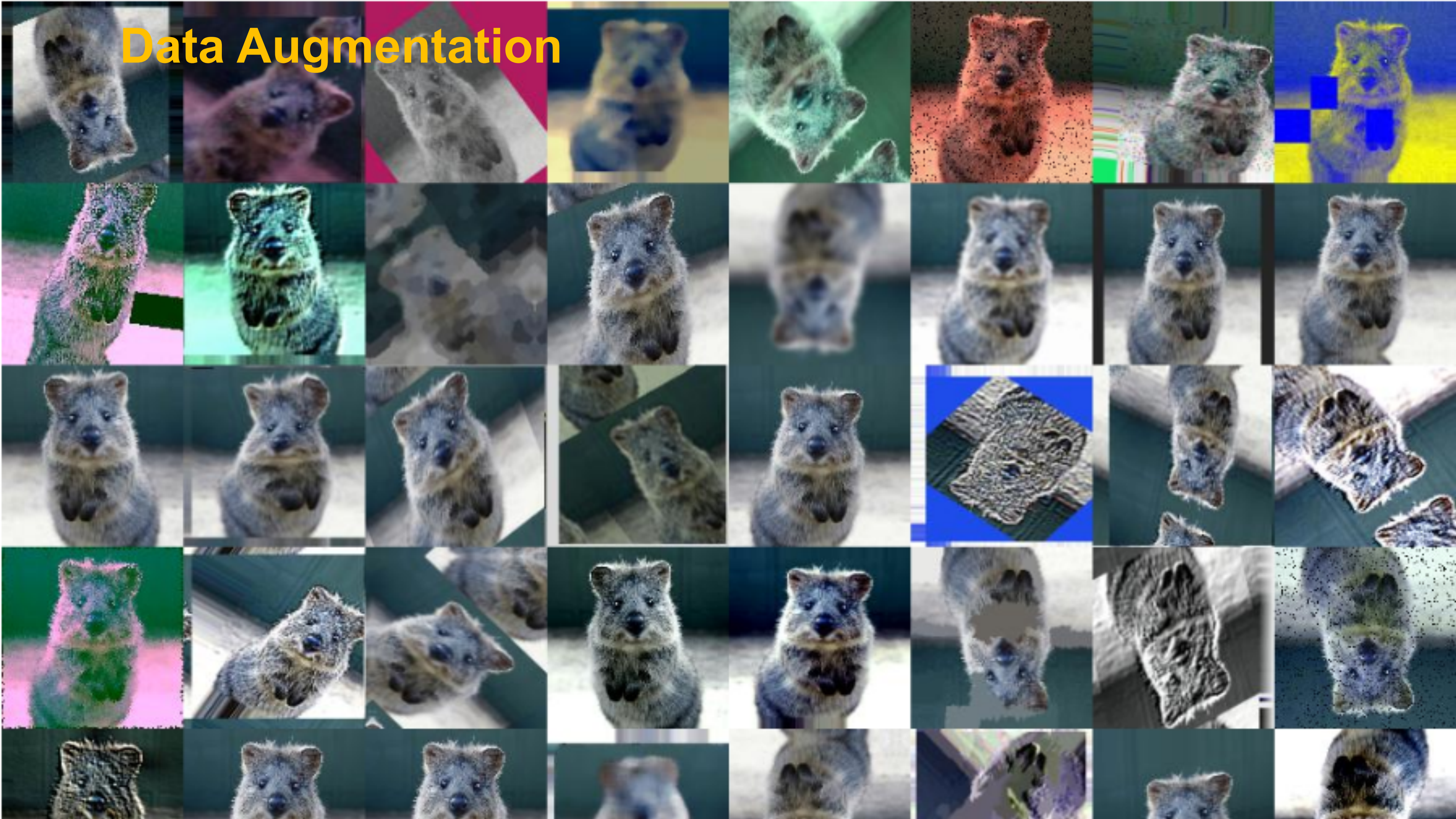
$$(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i^{(2)}} + \epsilon} S_i^{(1)}$$

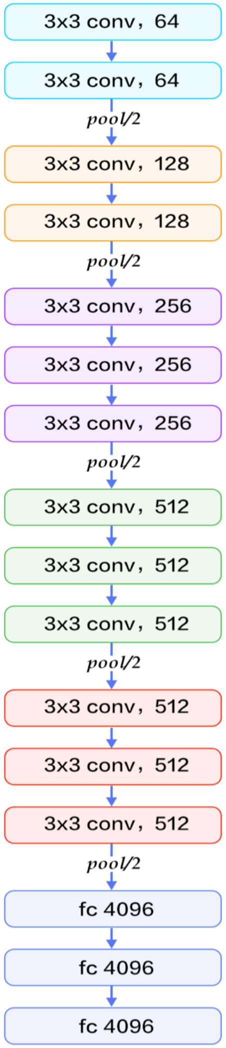
$$W_{t+1} = W_t - \mathrm{d}W_t$$

$$\boxed{\begin{aligned} S_i^{(1)}(t=0) &= 0 \\ S_i^{(2)}(t=0) &= 0 \end{aligned}}$$

1. Compute gradient
2. Compute first momentum (“velocity”)
3. Compute second momentum (parameter-wise normalization)
4. Compute update with **momentum** and **parameter-wise** learning rate
5. Apply update

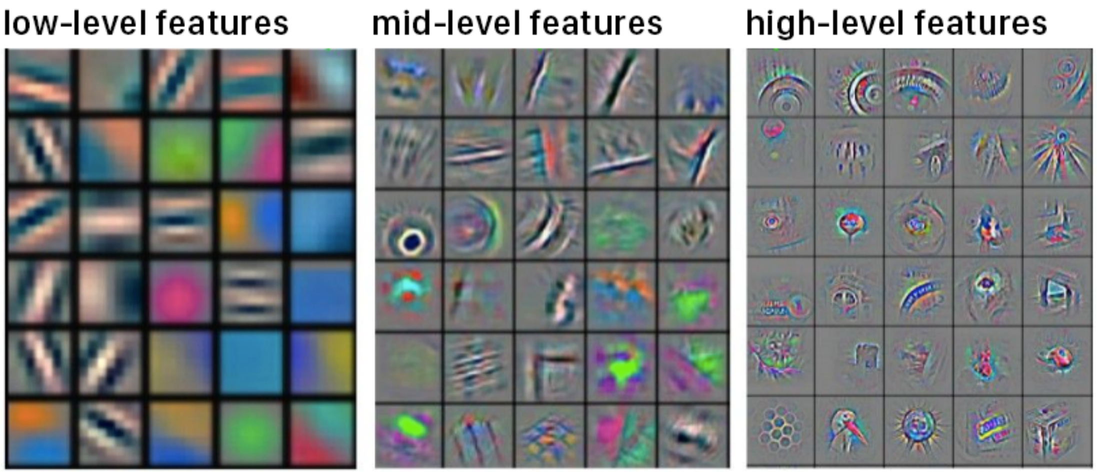
Data Augmentation





Transfer Learning

Fairly generic



Sharif Razavian, A., Azizpour, H., Sullivan, J., & Carlsson, S. (2014). CNN features off-the-shelf: an astounding baseline for recognition. CVPR (pp. 806-813).

Rather specific

Our Deep Learning Toolbox

Components

Linear classifiers f(**x**,**W**) = **W x + b**
Multilayer Perceptron
Fully Connected (Dense) Layers
Shared Weight Layers,
Convolutional Layers
Pooling Layers

Loss Functions

Mean Squared Error
Negative Log Likelihood
Hinge Loss
Norm regularization

Activations

Softmax
Sigmoid

Optimization

Gradient Descent
Backpropagation

Our Deep Learning Toolbox

Components

Linear classifiers f(**x**,**W**) = **W x + b**
Multilayer Perceptron
Fully Connected (Dense) Layers
Shared Weight Layers,
Convolutional Layers
Pooling Layers

Loss Functions

Mean Squared Error
Negative Log Likelihood
Hinge Loss
Norm regularization

Activations

Softmax
Sigmoid
Tanh
ReLu
eLu
PreLu
MaxOut

Optimization

Gradient Descent
Backpropagation
Momentum methods
Adaptive gradient methods

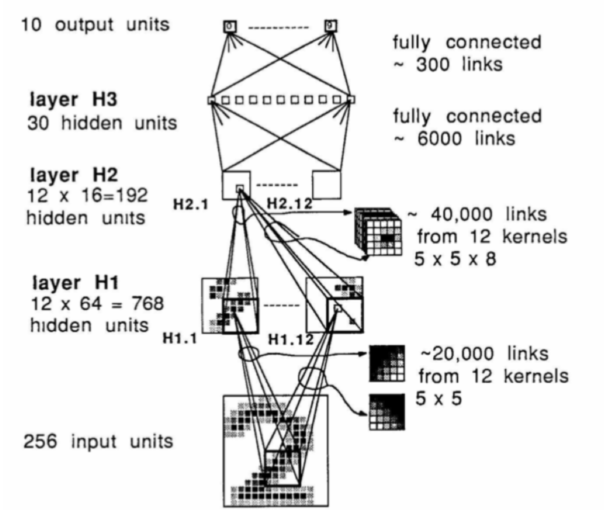
Tricks of the trade

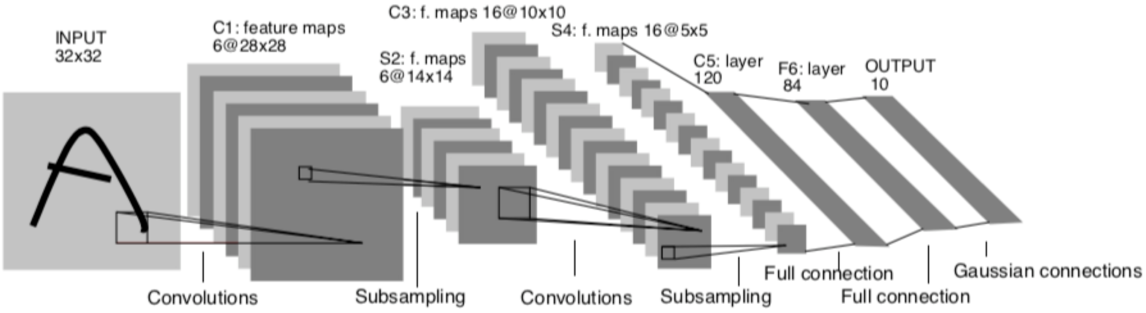
Dropout
Batch Normalization
Initialization
Data Augmentation
Transfer Learning

Recap

Computer Vision Architectures

LeNet

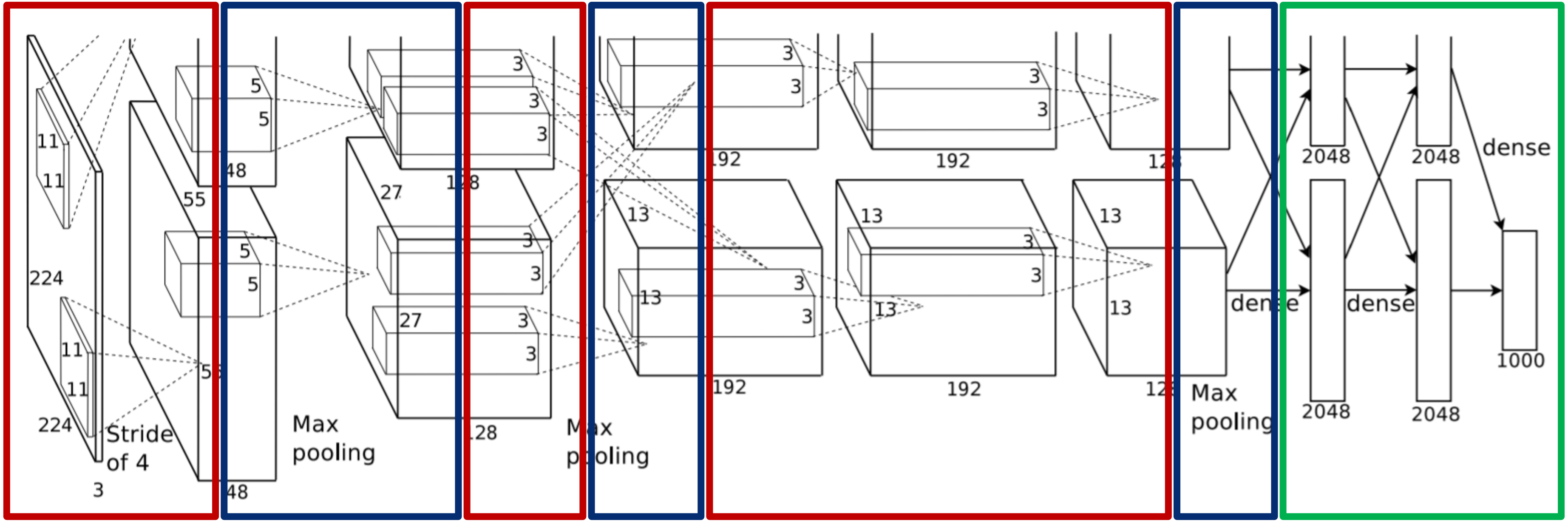




LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, *1*(4), 541-551.

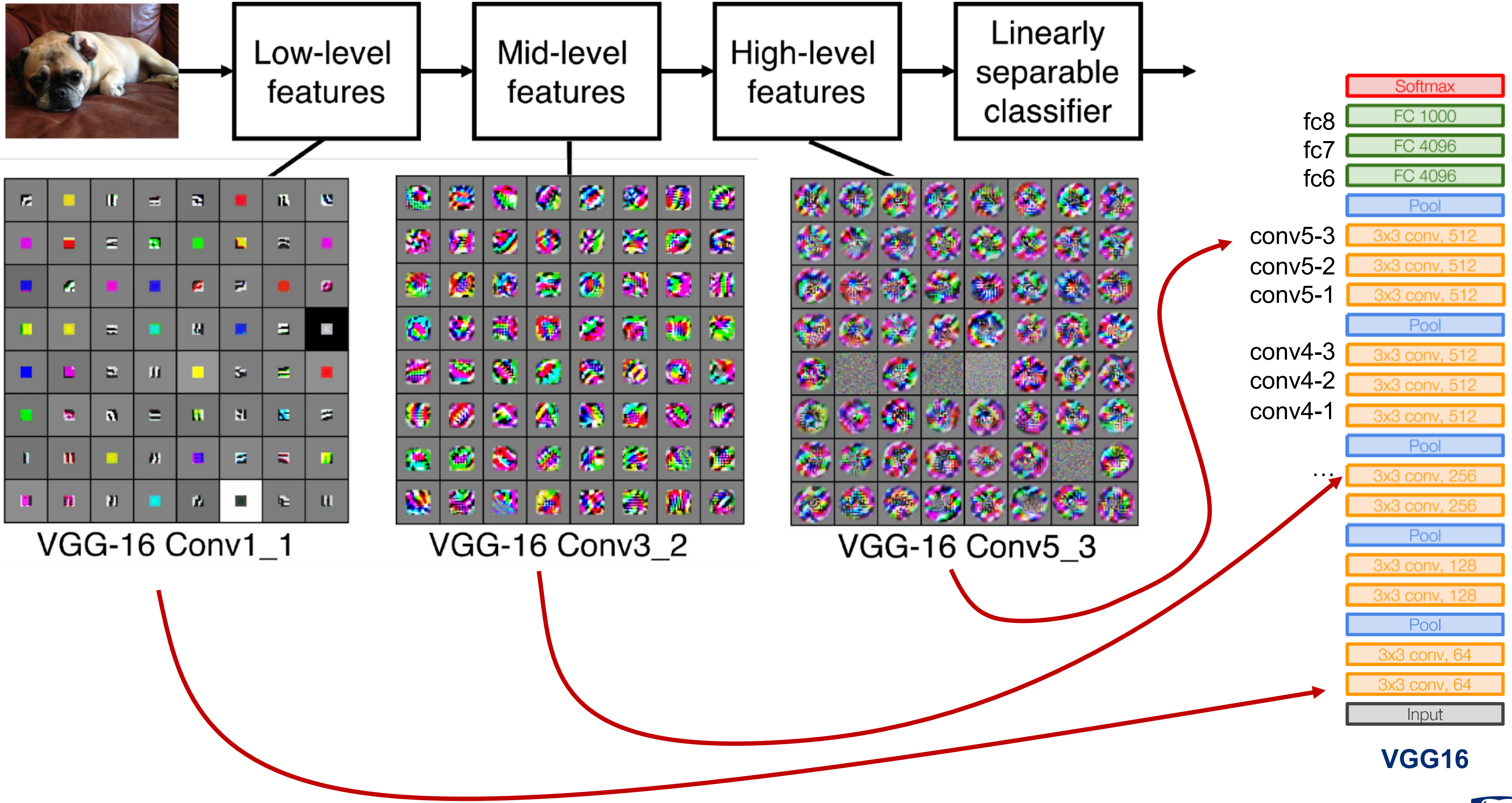
LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

AlexNet



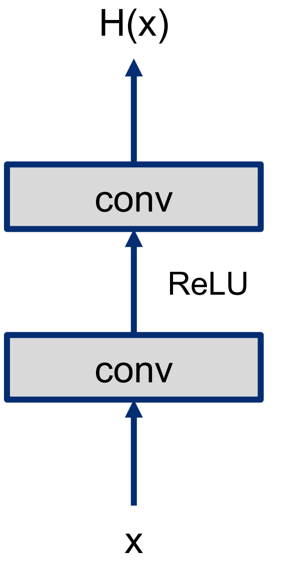
Conv 1 MaxPool 1 Conv 2 MaxPool 2 Conv 3,4,5 MaxPool 3 FCN 6,7,8

[Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. NeurIPS (pp. 1097-1105).]()

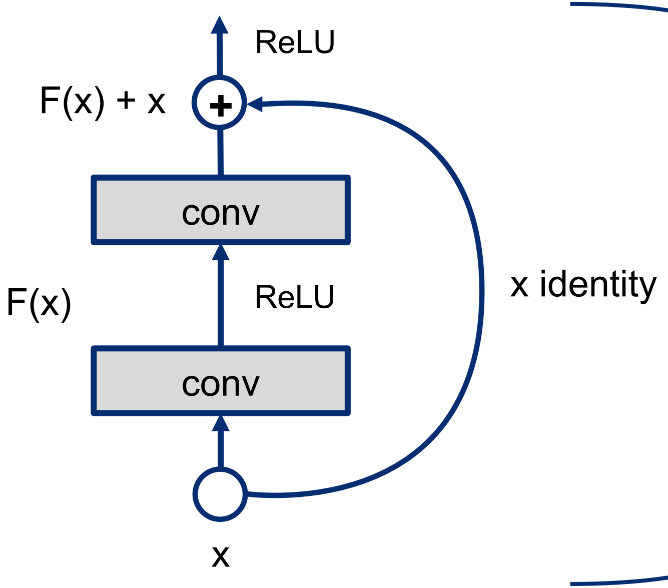


Low-level features
Mid-level features
High-level features
Linearly separable classifier
VGG-16 Conv1_1
VGG-16 Conv3_2
VGG-16 Conv5_3
Softmax
fc8
FC 1000
fc7
FC 4096
fc6
FC 4096
Pool
conv5-3
3x3 conv, 512
conv5-2
3x3 conv, 512
conv5-1
3x3 conv, 512
Pool
conv4-3
3x3 conv, 512
conv4-2
3x3 conv, 512
conv4-1
3x3 conv, 512
Pool
...
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input
VGG16

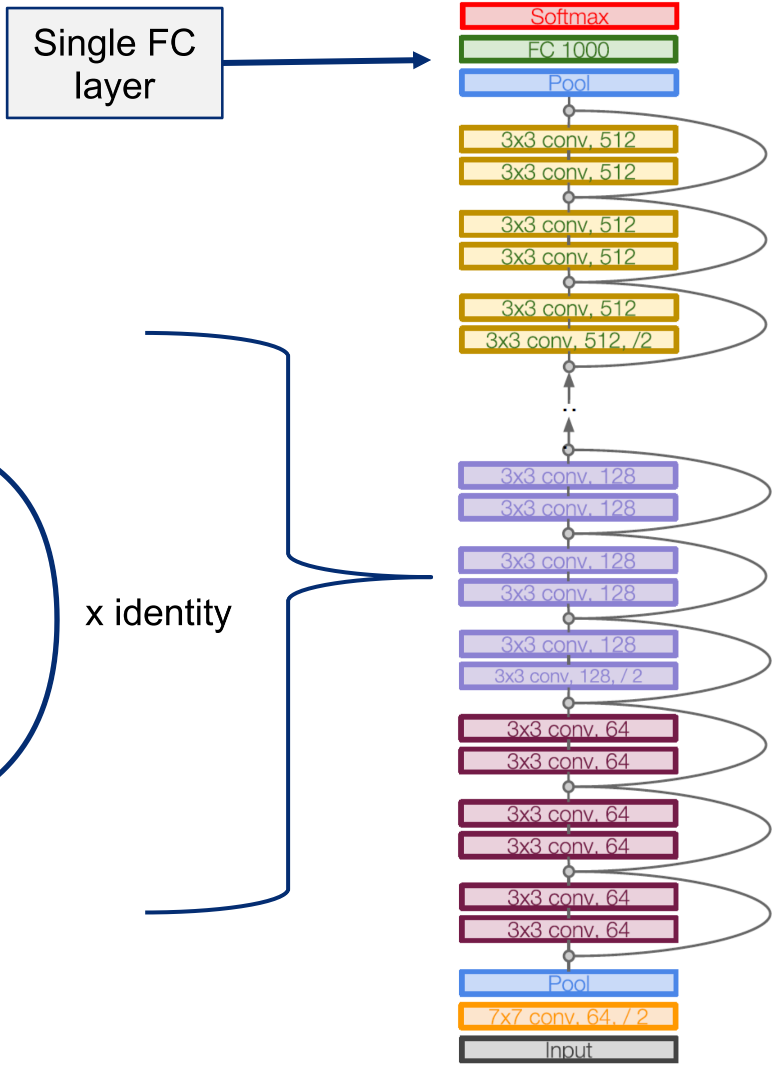
ResNet



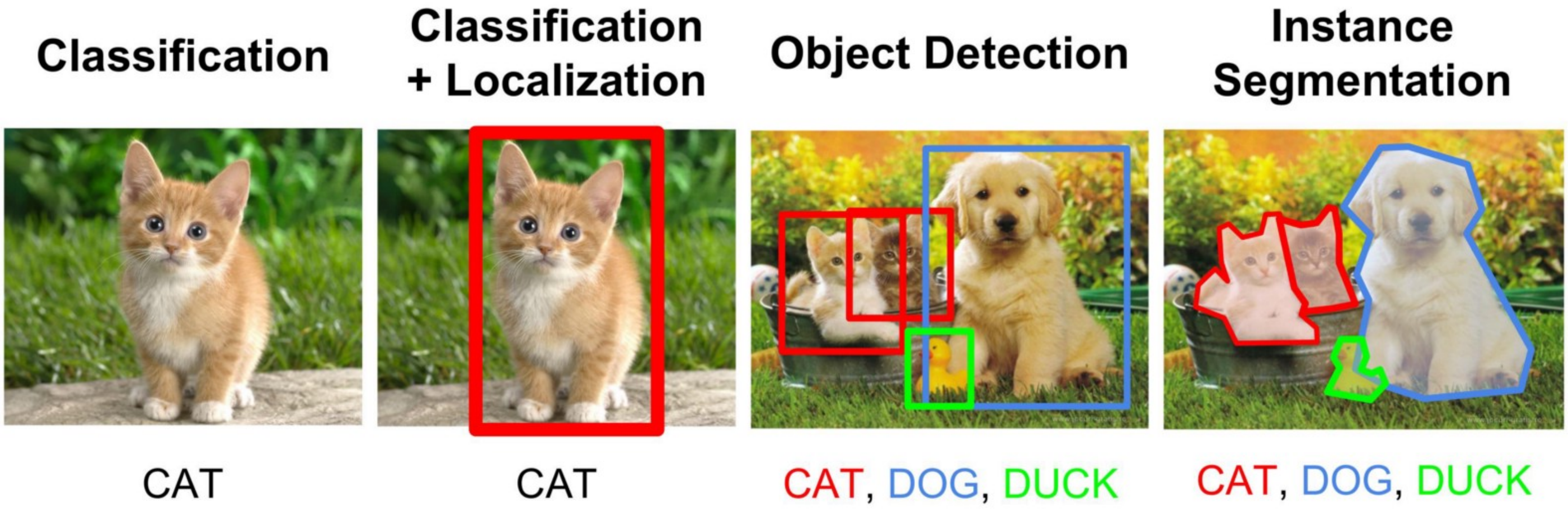
"Plain" conv layers



Residual block



From Classification to Instance Segmentation



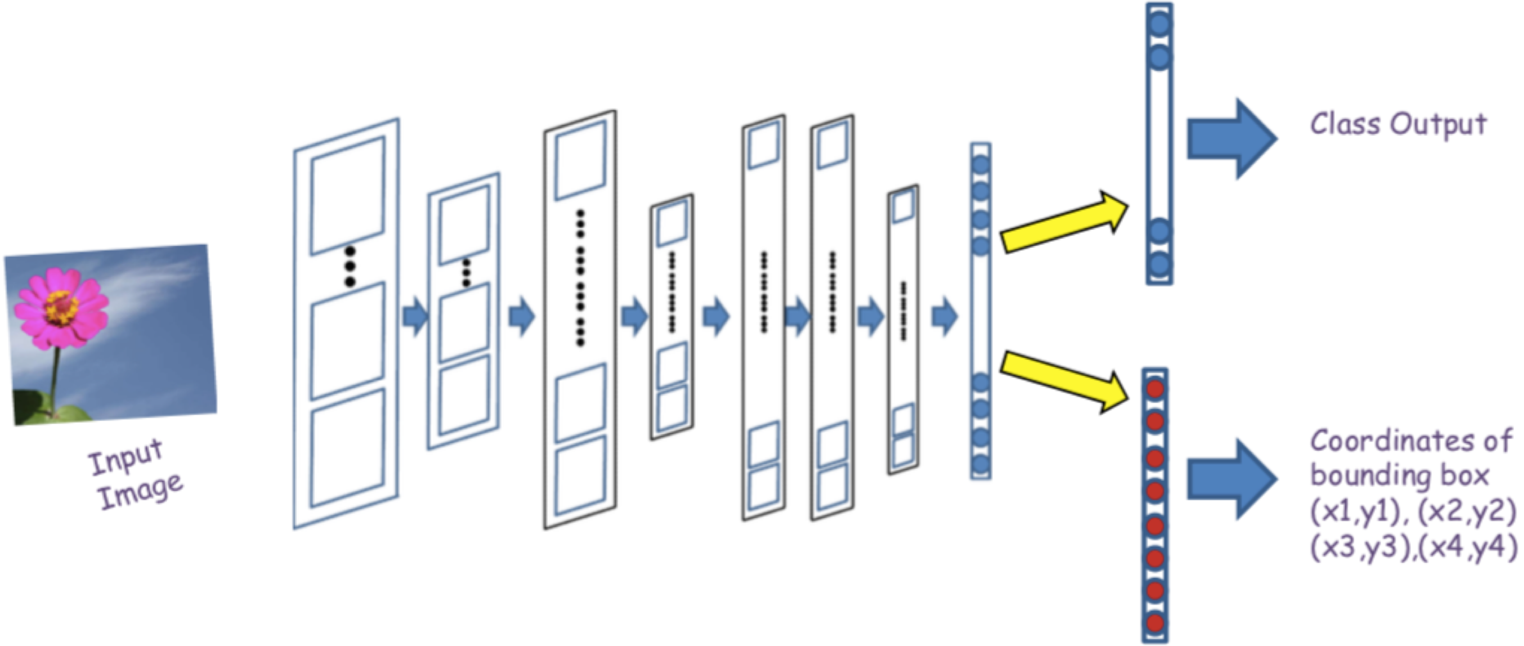
Single object

Multiple objects

[Ouaknine, A. (2018) Review of DL for Object Detection Blog Post]

Classification + Localization

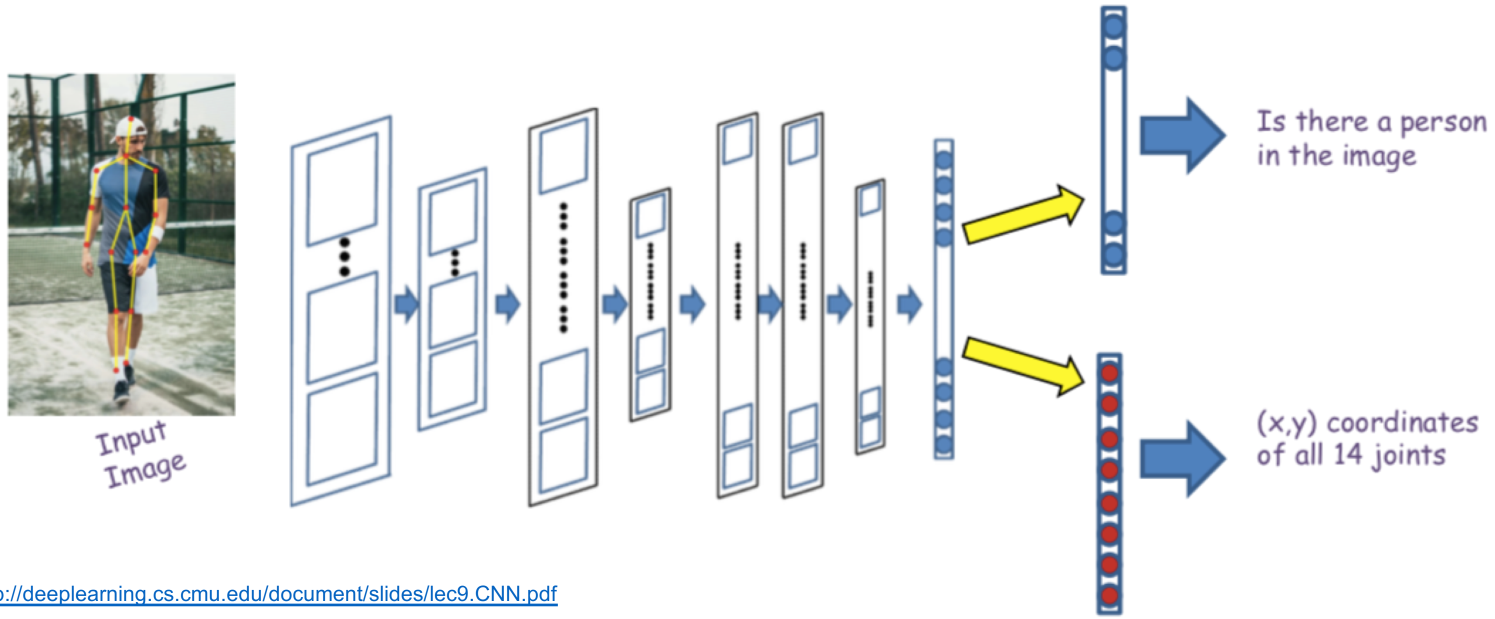
Learn to predict object **and bounding box coordinates** (Multitask learning)



http://deeplearning.cs.cmu.edu/document/slides/lec9.CNN.pdf

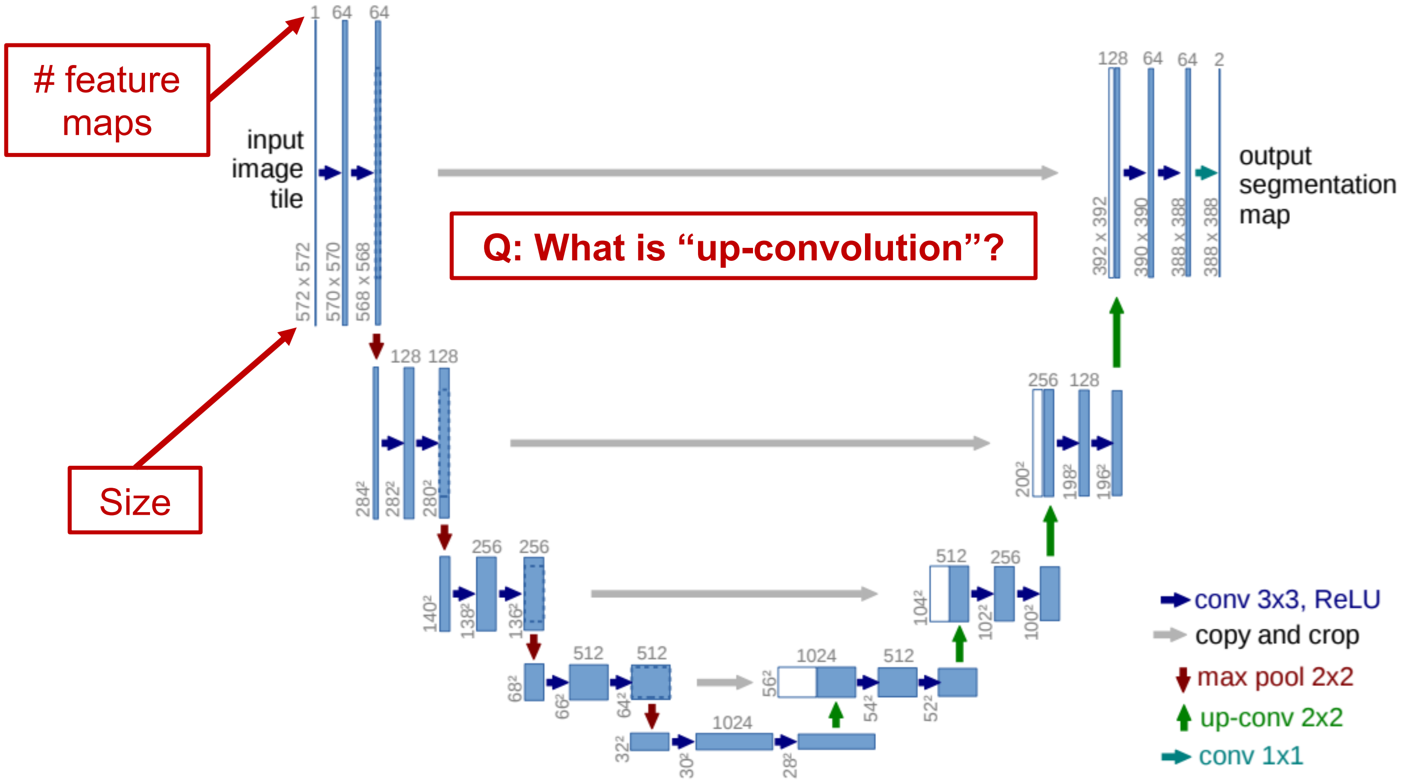
Pose Estimation/Landmark detection

Learn to predict object **and landmark coordinates** (Multitask learning)



http://deeplearning.cs.cmu.edu/document/slides/lec9.CNN.pdf

Image Segmentation



[Ronneberger, O., Fischer, P., & Brox, T. (2015, October). U-net: Convolutional networks for biomedical image segmentation. MICCAI (pp. 234-241). Springer, Cham.]

Our Deep Learning Toolbox

Components

Linear classifiers f(**x**,**W**) = **W x + b**
Multilayer Perceptron
Fully Connected (Dense) Layers
Shared Weight Layers,
Convolutional Layers
Pooling Layers

Loss Functions

Mean Squared Error
Negative Log Likelihood
Hinge Loss
Norm regularization

Activations

Softmax
Sigmoid
Tanh
ReLu
eLu
PreLu
MaxOut

Tricks of the trade

Dropout
Batch Normalization
Initialization
Data Augmentation
Transfer Learning

Optimization

Gradient Descent
Backpropagation
Momentum methods
Adaptive gradient methods

Our Deep Learning Toolbox

Components

Linear classifiers f(**x**,**W**) = **W x + b**
Multilayer Perceptron
Fully Connected (Dense) Layers
Shared Weight Layers,
Convolutional Layers
Pooling Layers
Residual Layers

Loss Functions

Mean Squared Error
Negative Log Likelihood
Hinge Loss
Norm regularization

Activations

Softmax
Sigmoid
Tanh
ReLu
eLu
PreLu
MaxOut

Optimization

Gradient Descent
Backpropagation
Momentum methods
Adaptive gradient methods

Tricks of the trade

Dropout
Batch Normalization
Initialization
Data Augmentation
Transfer Learning
Multitask Learning

CV Architectures

LeNet-5
AlexNet
VGG
RestNet
U-net

Sources

Content adapted from materials created by Mathias Unberath for the Machine Learning: Deep Learning (Fall 2019) course

Images from Stanford cs231n.