Basics of DL

What we assume you know

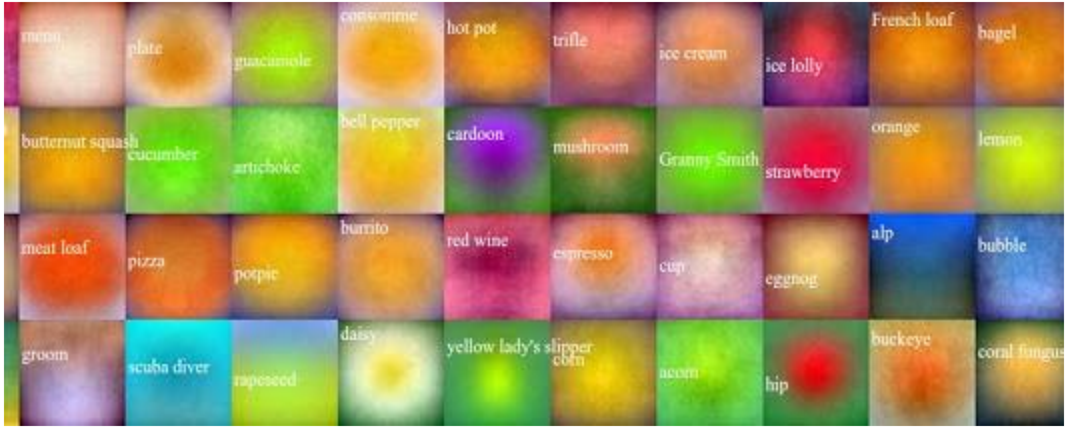
- Linear Algebra & Programming!
- Basics from the Introduction to Deep Learning lecture
- PyTorch (can use TensorFlow…)
- You have trained already several models and know how to debug problems, observe training curves, prepare training/validation/test data.

What is a Neural Network?

Neural Network

- Linear score function $f = Wx$

On CIFAR-10



On ImageNet

Credit: Li/Karpathy/Johnson

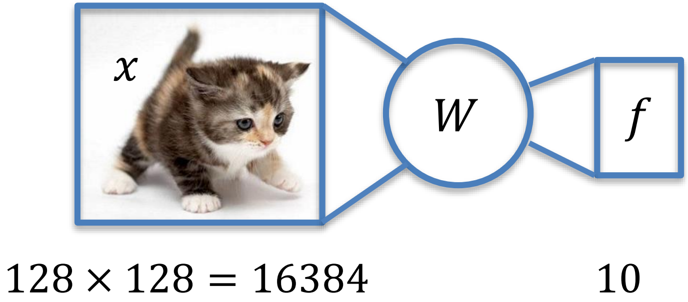
Neural Network

- Linear score function $f = Wx$

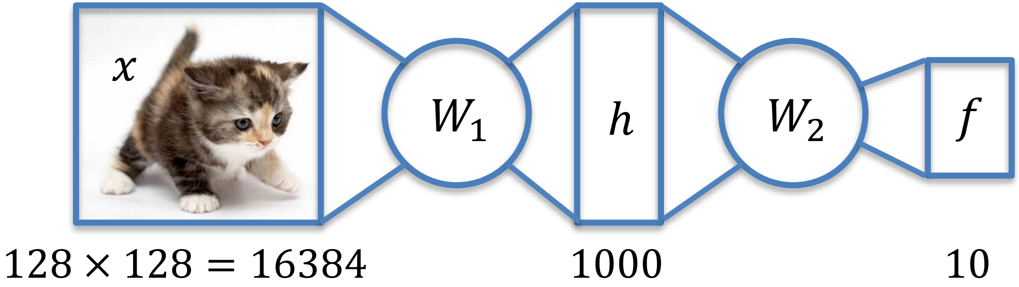
- Neural network is a nesting of 'functions'
  - 2-layers: $f = W_2 \max(0, W_1 x)$
  - 3-layers: $f = W_3 \max(0, W_2 \max(0, W_1 x))$
  - 4-layers: $f = W_4 \tanh\,(\mathrm{W}_3, \max(0, W_2 \max(0, W_1 x)))$
  - 5-layers: $f = W_5 \sigma(W_4 \tanh(\mathrm{W}_3, \max(0, W_2 \max(0, W_1 x))))$
  - … up to hundreds of layers

Neural Network

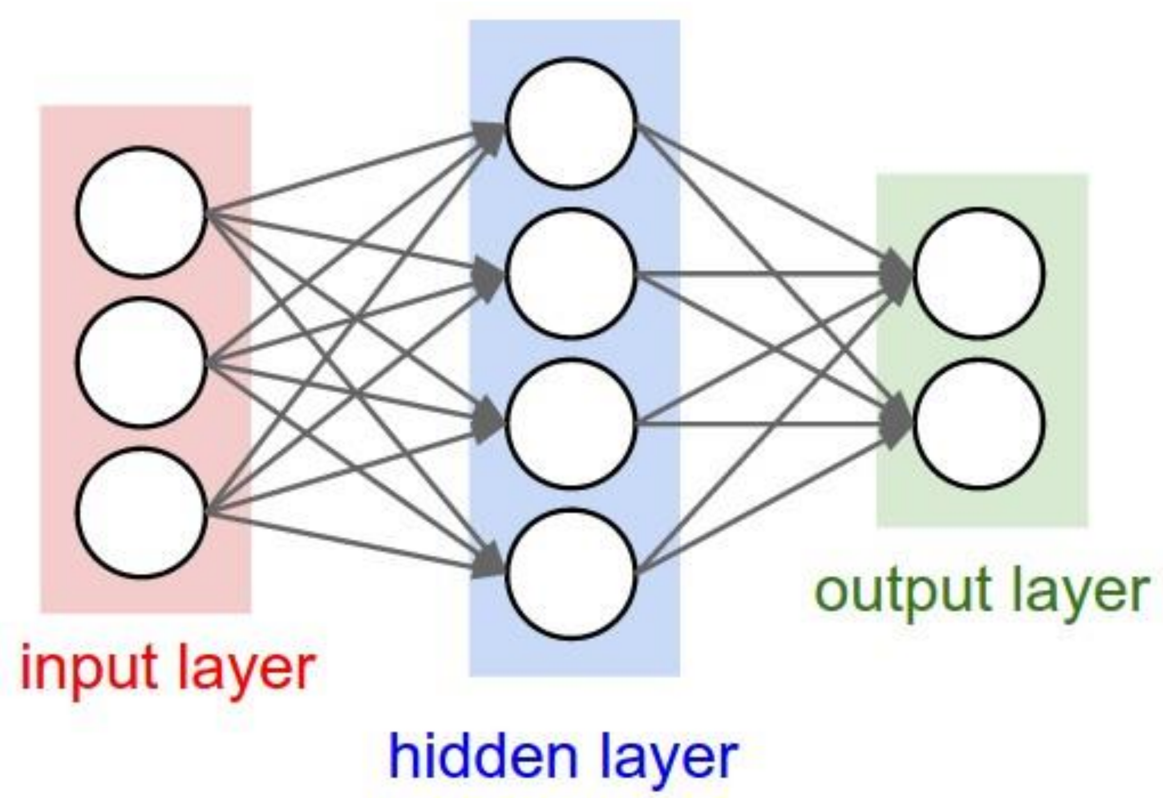
1-layer network: $f = Wx$



2-layer network: $f = W_2 \max(0, W_1 x)$



Neural Network

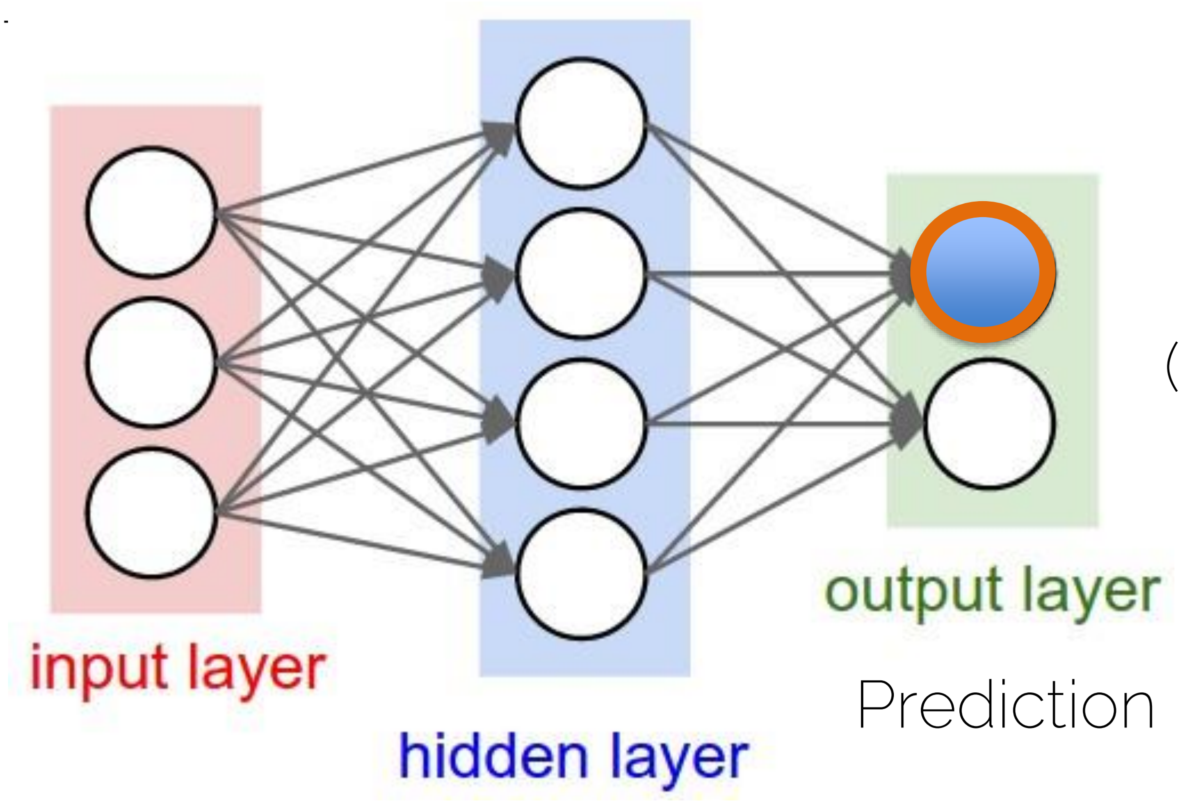


Credit: Li/Karpathy/Johnson

Loss functions

Neural networks

What is ·



Loss (Softmax, Hinge)

Prediction

Loss functions

- Softmax loss function

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}}\right)$$

Evaluate the ground truth score for the image

- Hinge Loss (derived from the Multiclass SVM loss)

$$L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1)$$

Loss functions

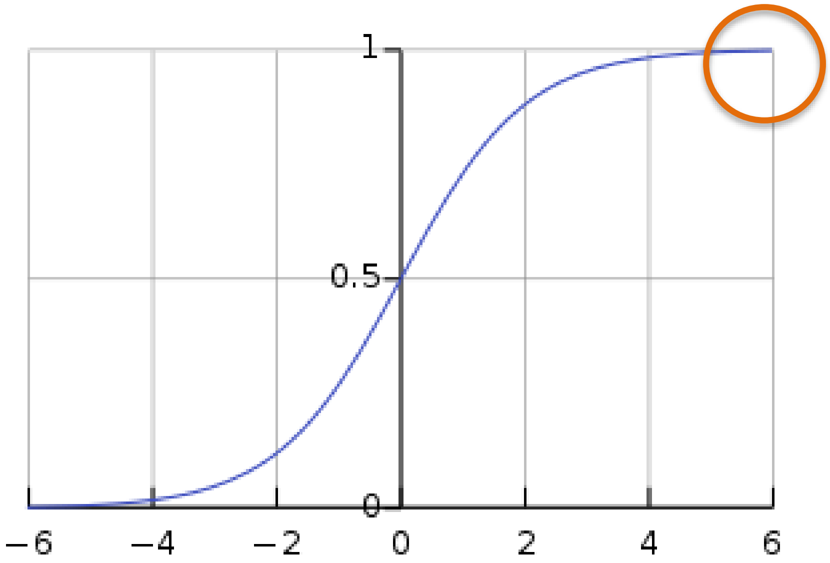
- Softmax loss function
  - Optimizes until the loss is zero
- Hinge Loss (derived from the Multiclass SVM loss)
  - Saturates whenever it has learned a class “well enough”

Activation functions

Sigmoid

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

Forward



$$x = 6$$

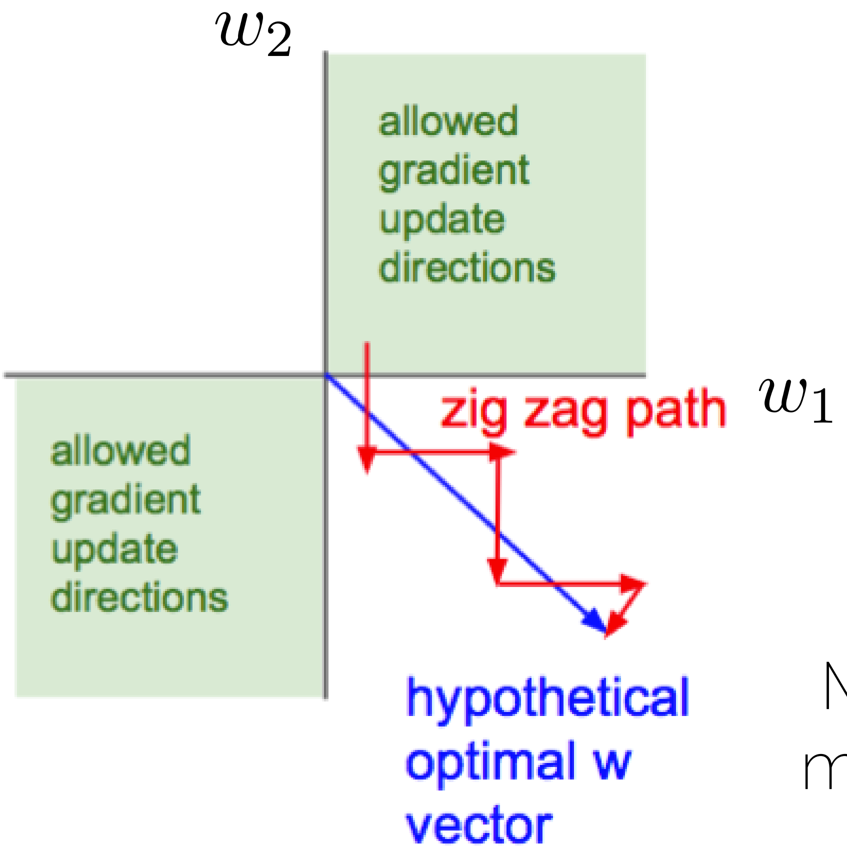
✗ Saturated neurons kill the gradient flow

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

$$\frac{\partial \sigma}{\partial x}$$

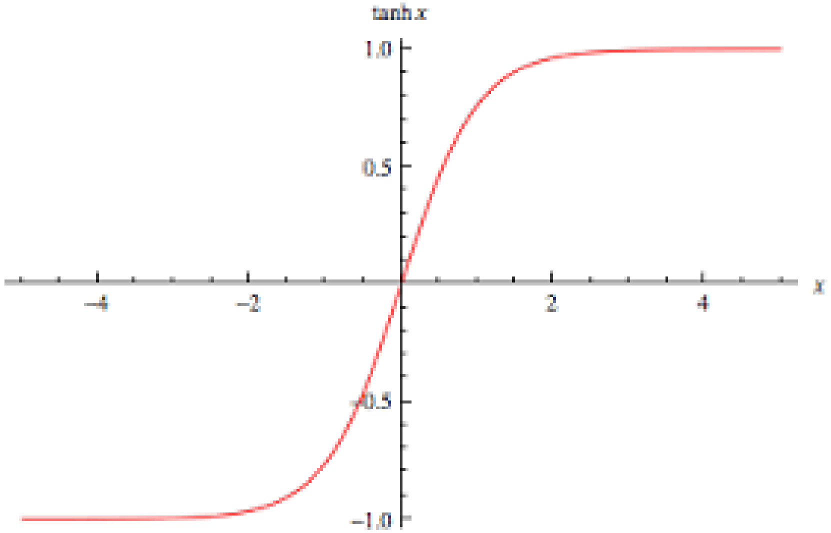
$$\frac{\partial L}{\partial \sigma}$$

Problem of positive output



More on zero-mean data later

tanh



✗ Still saturates

✓ Zero-centered

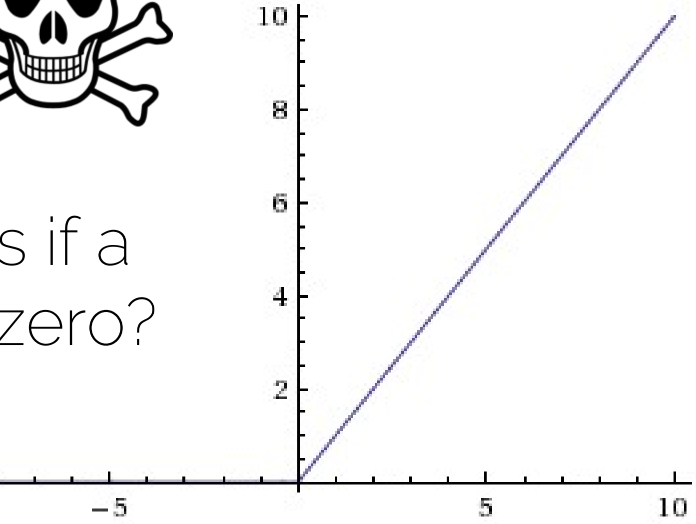
✗ Still saturates

LeCun 1991

Rectified Linear Units (ReLU)

✗ Dead ReLU

What happens if a ReLU outputs zero?

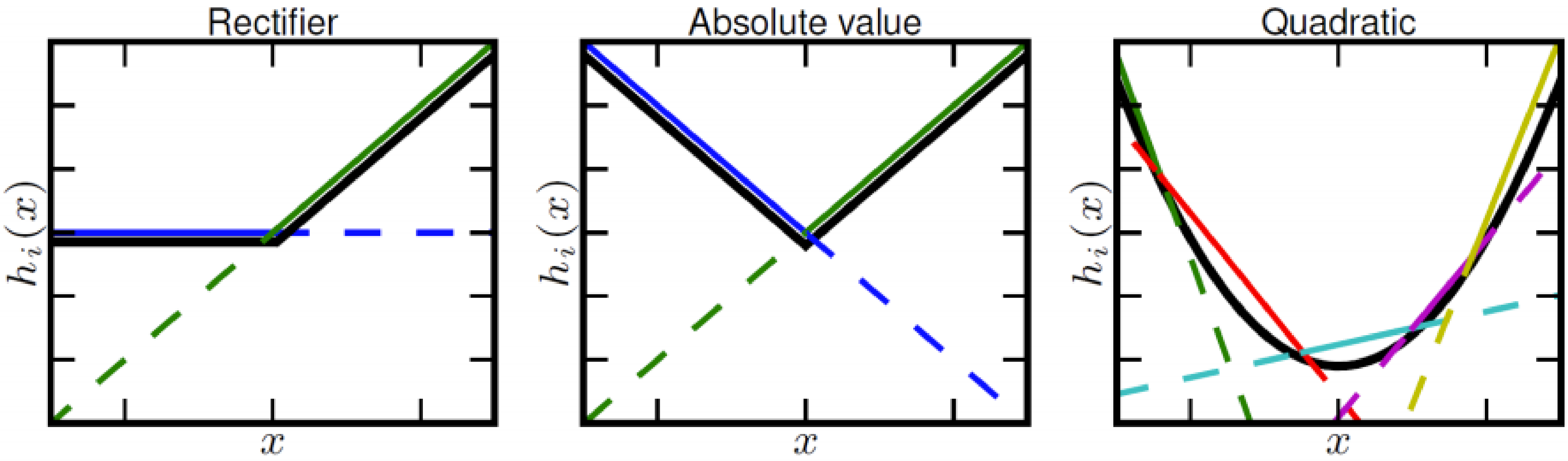


Large and consistent gradients ✓

✓ Fast convergence

✓ Does not saturate

Maxout units



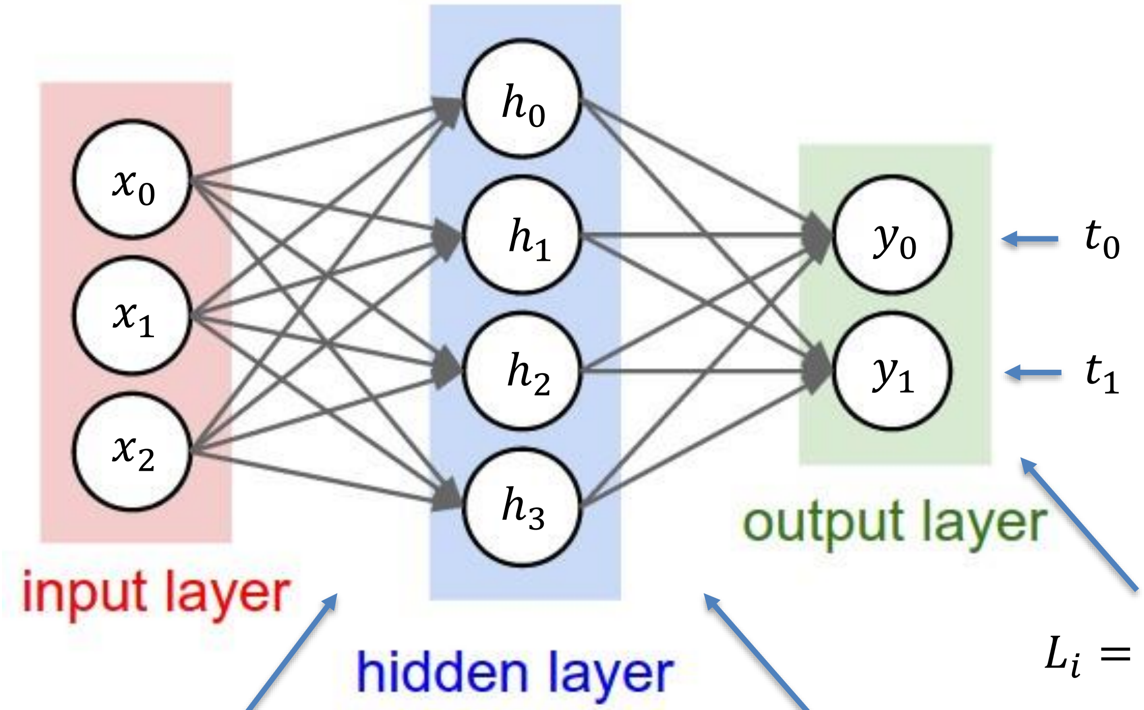
k=2 k=2 k=5

✓ Generalization of ReLUs
✓ Linear regimes
✓ Does not die
✓ Does not saturate

✗ Increase of the number of parameters

Optimization

Gradient Descent for Neural Networks



$$\nabla_{w,b} f_{\{x,t\}}(w) = \begin{bmatrix} \frac{\partial f}{\partial w_{0,0,0}} \\ \cdots \\ \cdots \\ \frac{\partial f}{\partial w_{l,m,n}} \\ \cdots \\ \cdots \\ \frac{\partial f}{\partial b_{l,m}} \end{bmatrix}$$

$$L_i = (y_i - t_i)^2$$

$$h_j = A(b_{0,j} + \sum_k x_k w_{0,j,k})$$

$$y_i = A(b_{1,i} + \sum_j h_j w_{1,i,j})$$

Just simple: $A(x) = \max(0, x)$

Stochastic Gradient Descent (SGD)

$$\theta^{k+1} = \theta^k - \alpha \nabla_\theta L(\theta^k, x_{\{1..m\}}, y_{\{1..m\}})$$

k now refers to k-th iteration

$$\nabla_\theta L = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L_i$$

m training samples in the current batch

Gradient for the k-th batch

Note the terminology: iteration vs epoch

Gradient Descent with Momentum

$$v^{k+1} = \beta \cdot v^k + \nabla_\theta L(\theta^k)$$

accumulation rate
('friction', momentum)

velocity

Gradient of current minibatch

$$\theta^{k+1} = \theta^k - \alpha \cdot v^{k+1}$$

model

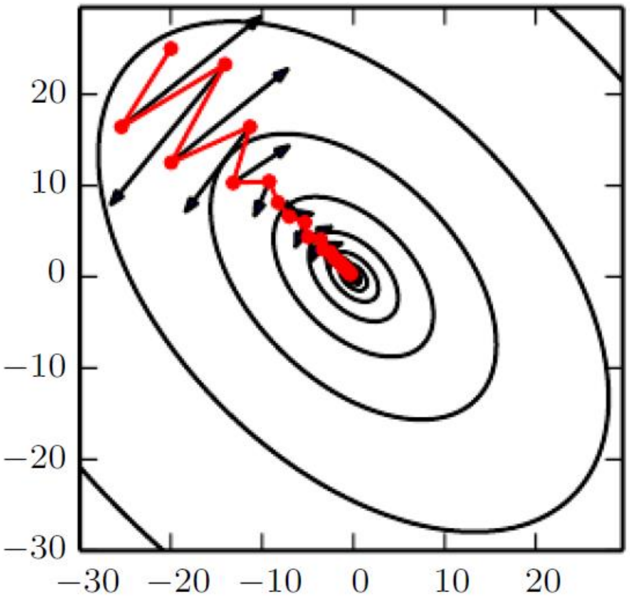
learning rate

velocity

Exponentially-weighted average of gradient

Important: velocity $\boldsymbol{v}^k$ is vector-valued!

Gradient Descent with Momentum



Step will be largest when a sequence of gradients all point to the same direction

Hyperparameters are α, β
β is often set to 0.9

$$\theta^{k+1} = \theta^{k} - \alpha \cdot v^{k+1}$$

Fig. credit: I. Goodfellow

RMSProp

$$s^{k+1} = \beta \cdot s^k + (1-\beta)\boxed{[\nabla_\theta L \circ \nabla_\theta L]}$$

Element-wise multiplication

$$\theta^{k+1} = \theta^k - \alpha \cdot \frac{\nabla_\theta L}{\sqrt{s^{k+1}} + \epsilon}$$

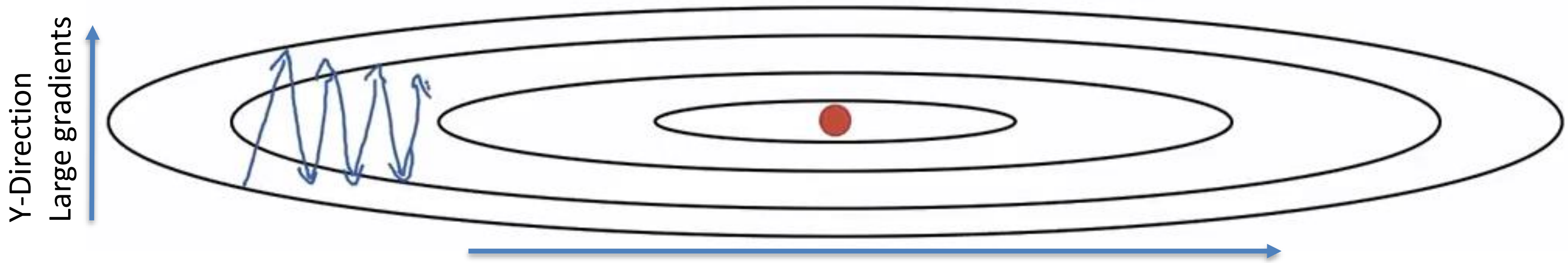
Hyperparameters: α, β, ϵ

Needs tuning!

Often 0.9

Typically 10^{-8}

RMSProp



(uncentered) variance of gradients
-> second momentum

$$\boxed{s^{k+1} = \beta \cdot s^k + (1-\beta)[\nabla_\theta L \circ \nabla_\theta L]}$$

$$\theta^{k+1} = \theta^k - \alpha \cdot \frac{\nabla_\theta L}{\sqrt{s^{k+1}} + \epsilon}$$

We're dividing by square gradients:
- Division in Y-Direction will be large
- Division in X-Direction will be small

Can increase learning rate!

Fig. credit: A. Ng

Adaptive Moment Estimation (Adam)

Combines Momentum and RMSProp

$$\boxed{m^{k+1} = \beta_1 \cdot m^k + (1-\beta_1)\nabla_\theta L\left(\theta^k\right)}$$

First momentum:
mean of gradients

$$\boxed{v^{k+1} = \beta_2 \cdot v^k + (1-\beta_2)[\nabla_\theta L\left(\theta^k\right) \circ \nabla_\theta L\left(\theta^k\right)]}$$

Second momentum:
variance of gradients

$$\theta^{k+1} = \theta^k - \alpha \cdot \frac{m^{k+1}}{\sqrt{v^{k+1}}+\epsilon}$$

Adam

Combines Momentum and RMSProp

$$m^{k+1} = \beta_1 \cdot m^k + (1-\beta_1)\nabla_\theta L(\theta^k)$$

$$v^{k+1} = \beta_2 \cdot v^k + (1-\beta_2)[\nabla_\theta L(\theta^k) \circ \nabla_\theta L(\theta^k)]$$

m^{k+1} and v^{k+1} are initialized with zero
-> bias towards zero

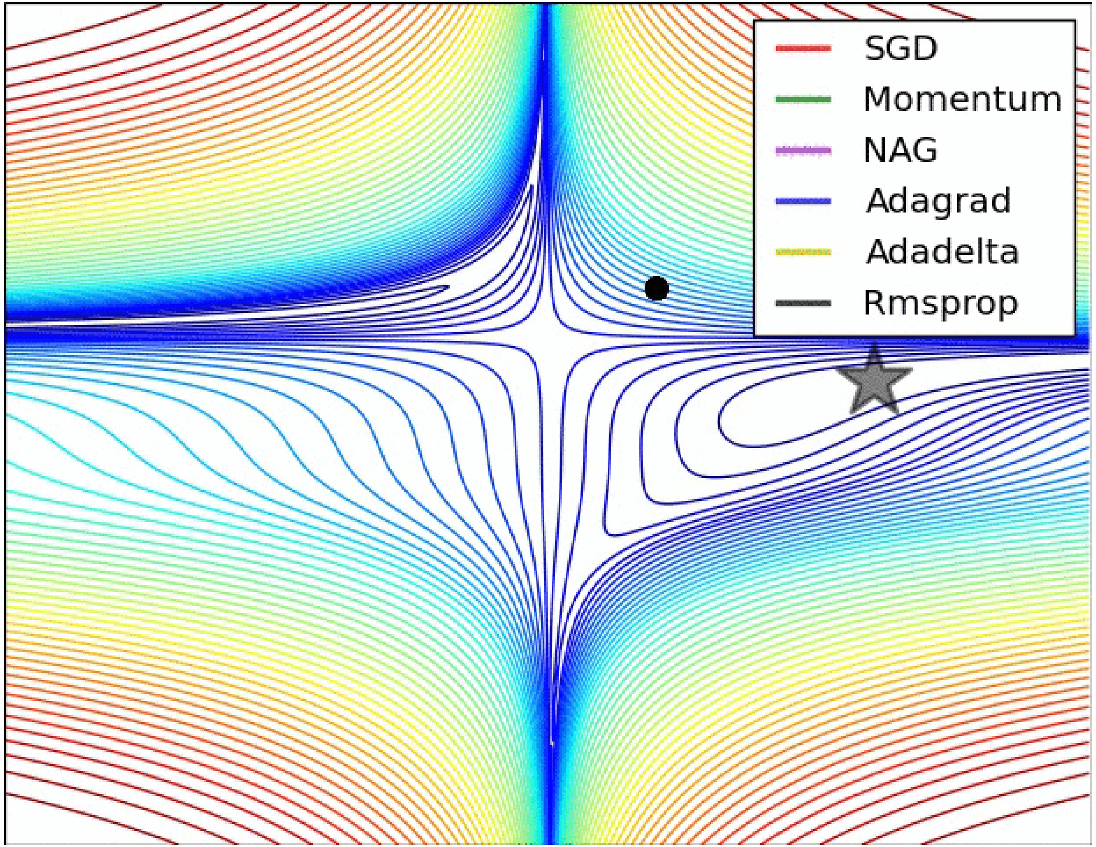
Typically, bias-corrected moment updates

$$\hat{m}^{k+1} = \frac{m^k}{1-\beta_1}$$

$$\hat{v}^{k+1} = \frac{v^k}{1-\beta_2}$$

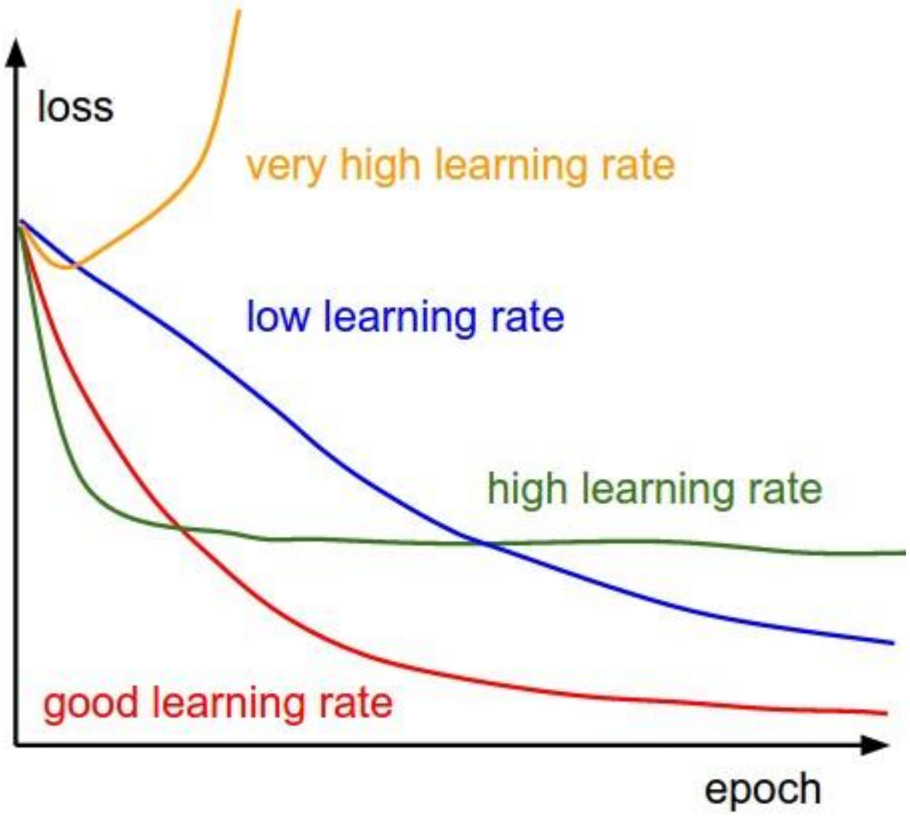
$$\theta^{k+1} = \theta^k - \alpha \cdot \frac{\hat{m}^{k+1}}{\sqrt{\hat{v}^{k+1}}+\epsilon}$$

Convergence



Training NNs

Importance of Learning Rate



Over- and Underfitting

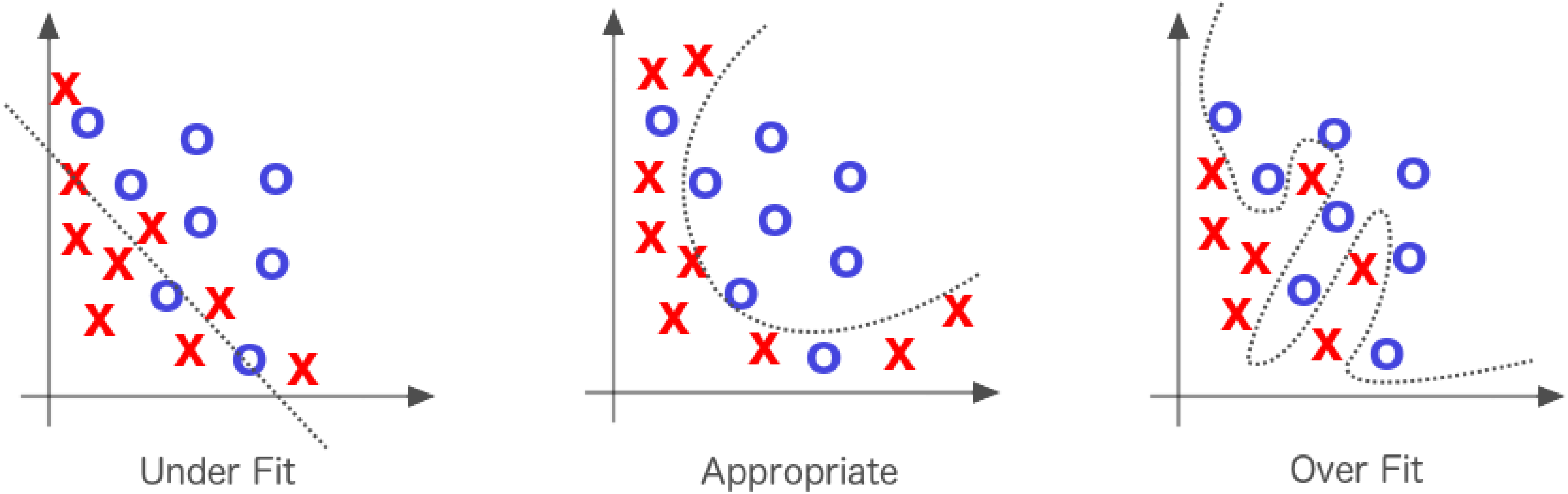
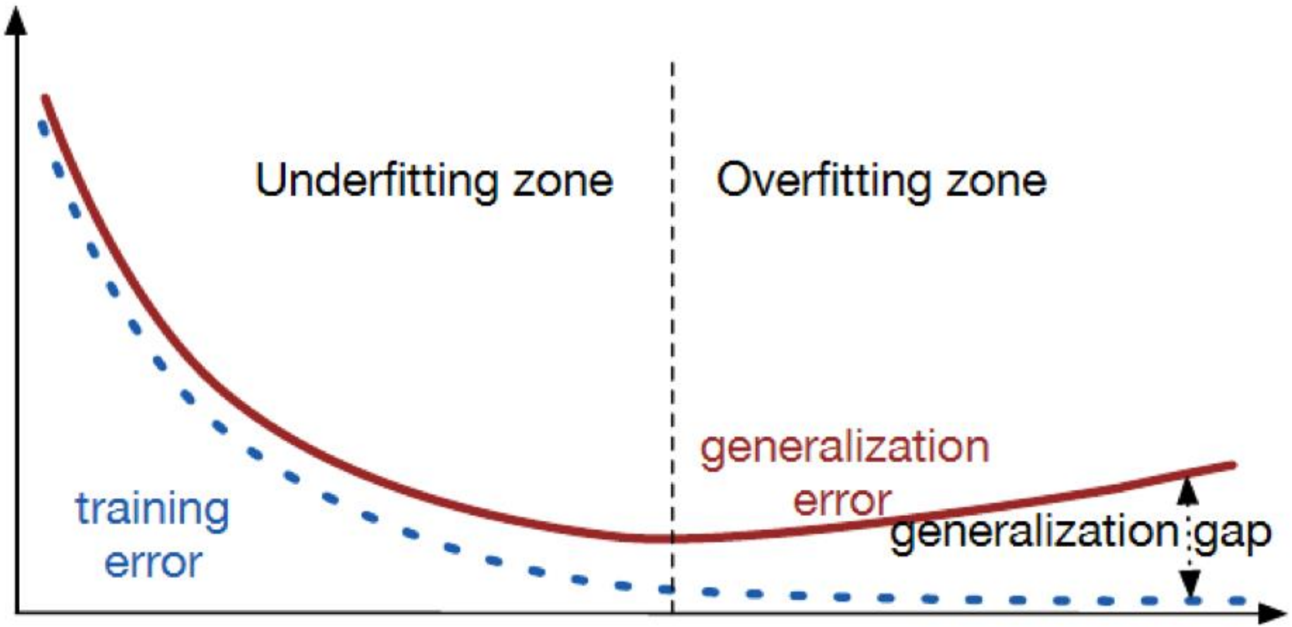


Figure extracted from Deep Learning by Adam Gibson, Josh Patterson, O'Reily Media Inc., 2017

Over- and Underfitting



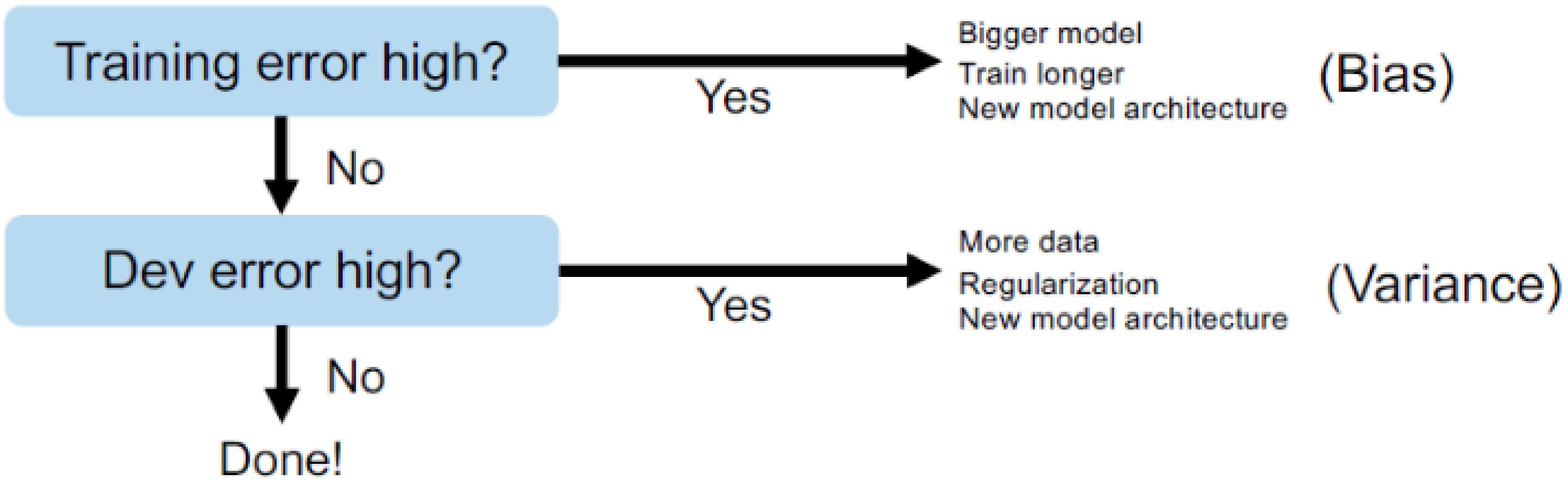
Source: http://srdas.github.io/DLBook/ImprovingModelGeneralization.html

Basic recipe for machine learning

- Split your data

Find your hyperparameters

Basic recipe for machine learning



Regularization

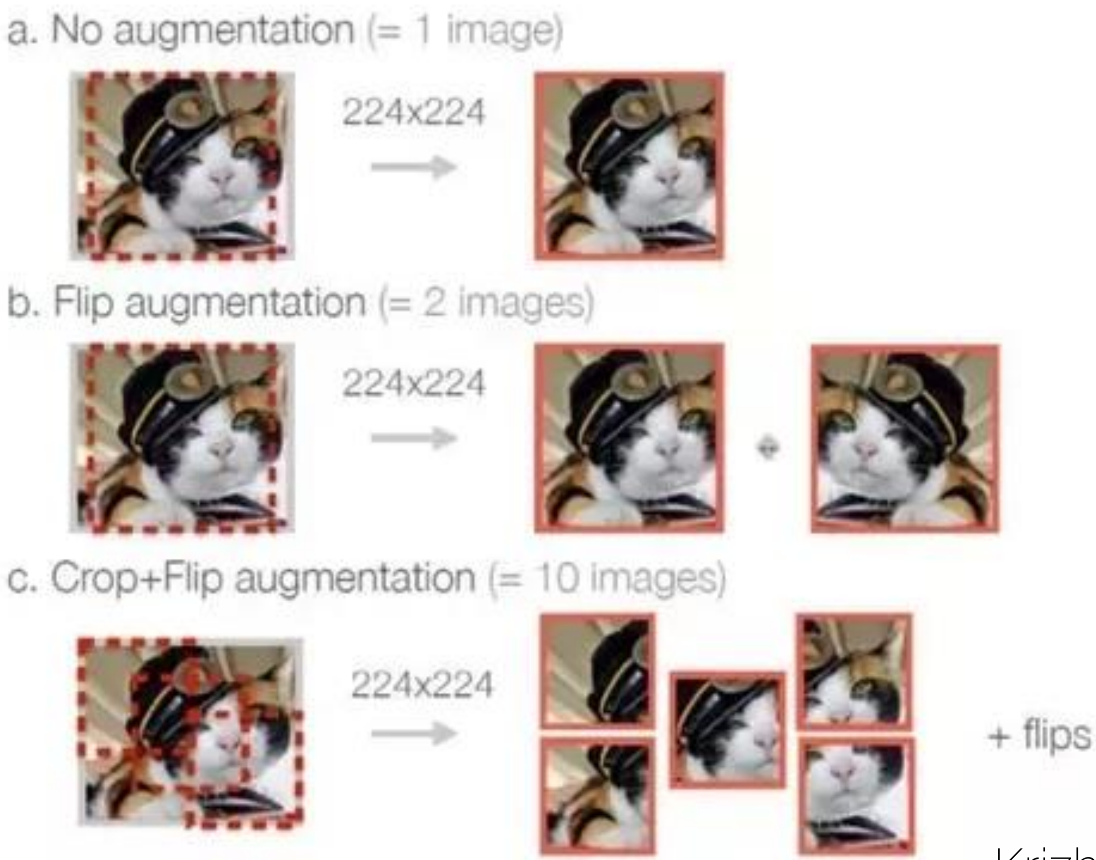
Regularization

- Any strategy that aims to

Lower validation error

Increasing training error

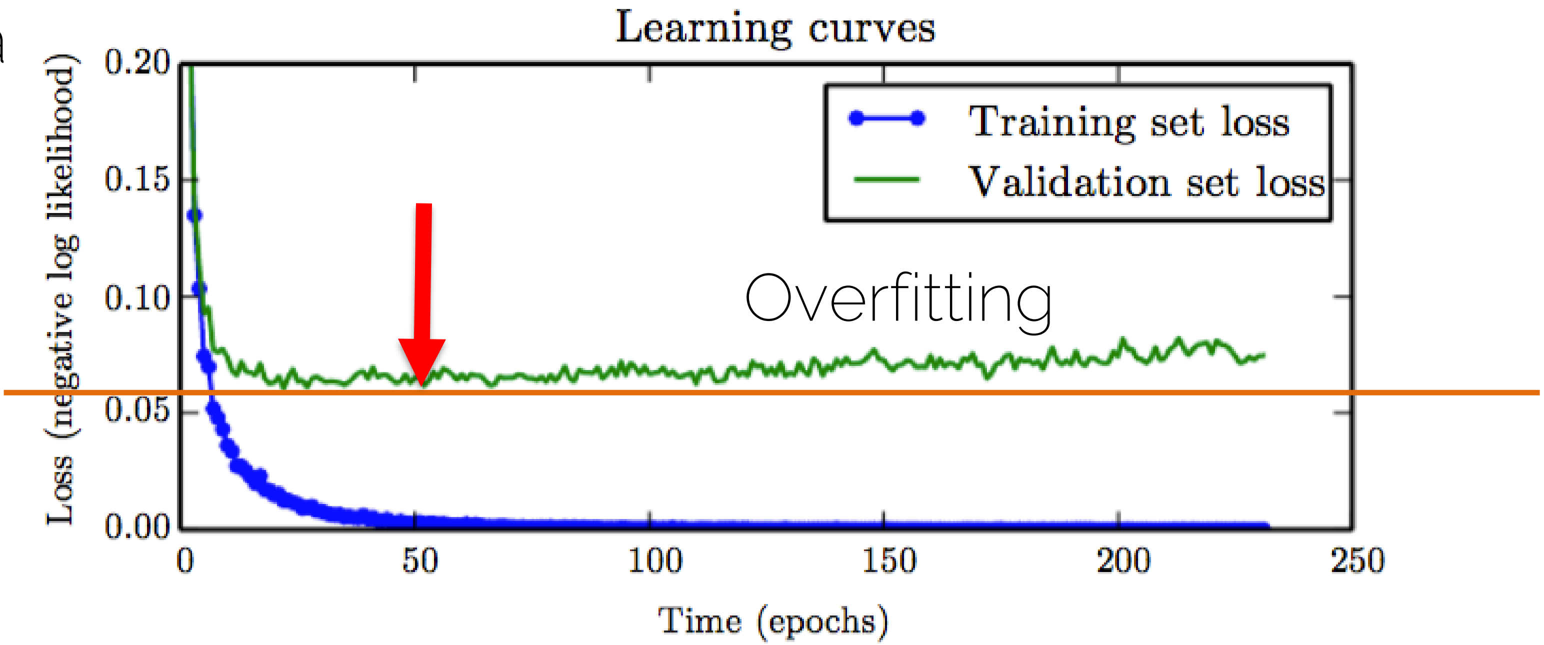
Data augmentation



Krizhevsky 2012

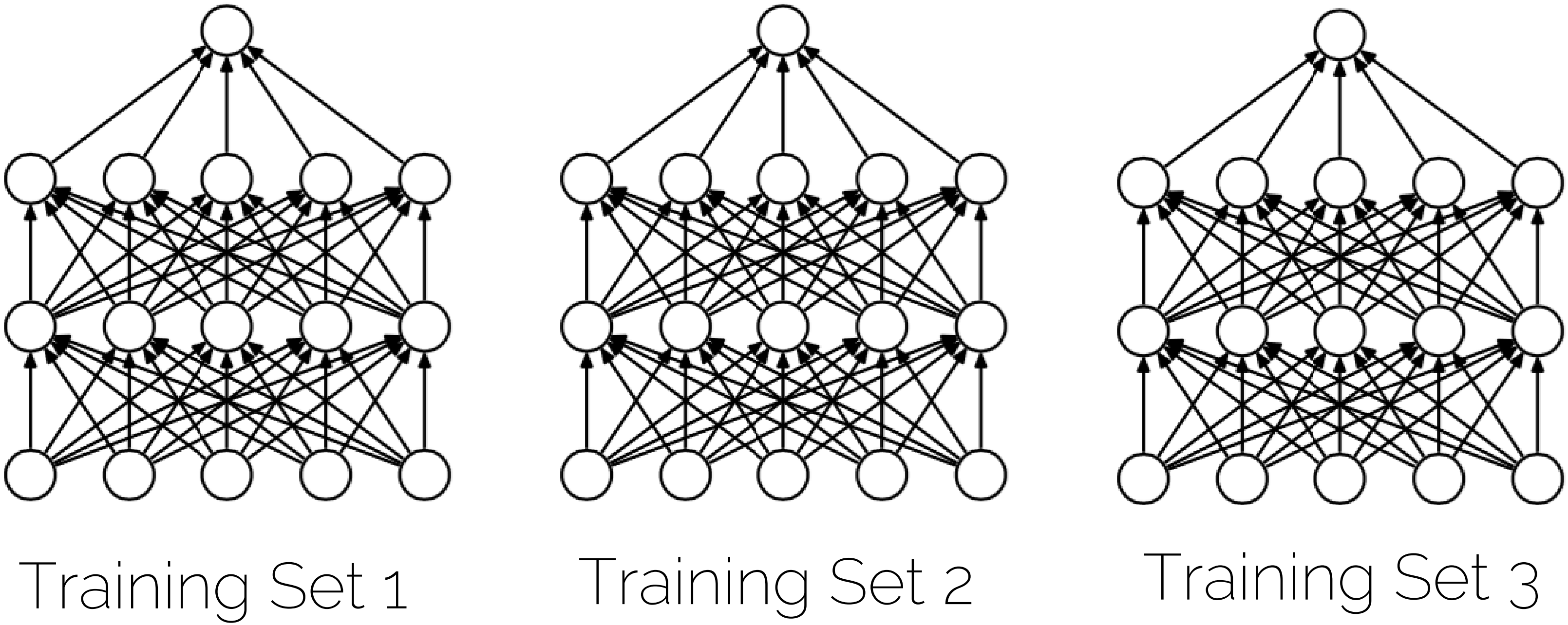
Early stopping

Tra



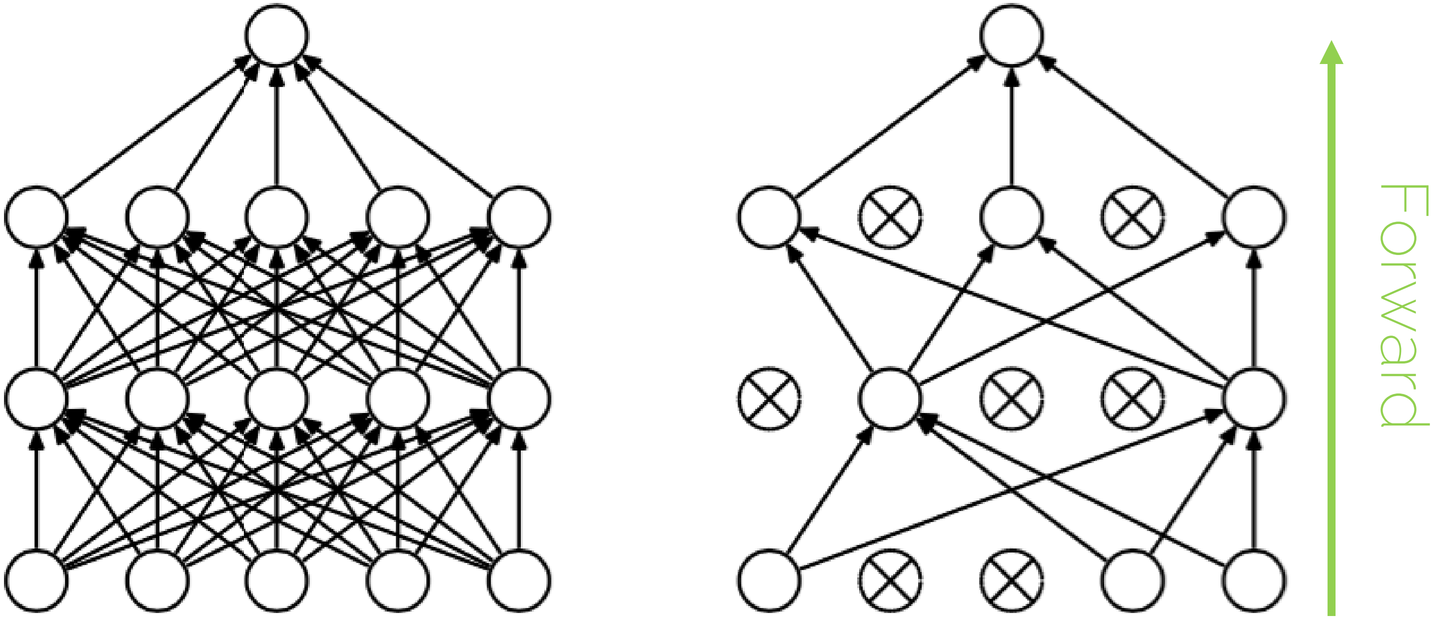
Bagging and ensemble methods

- Bagging: uses k different datasets



Dropout

- Disable a random set of neurons (typically 50%)



(a) Standard Neural Net (b) After applying dropout.

Srivastava 2014

How to deal with images?

Using CNNs in Computer Vision

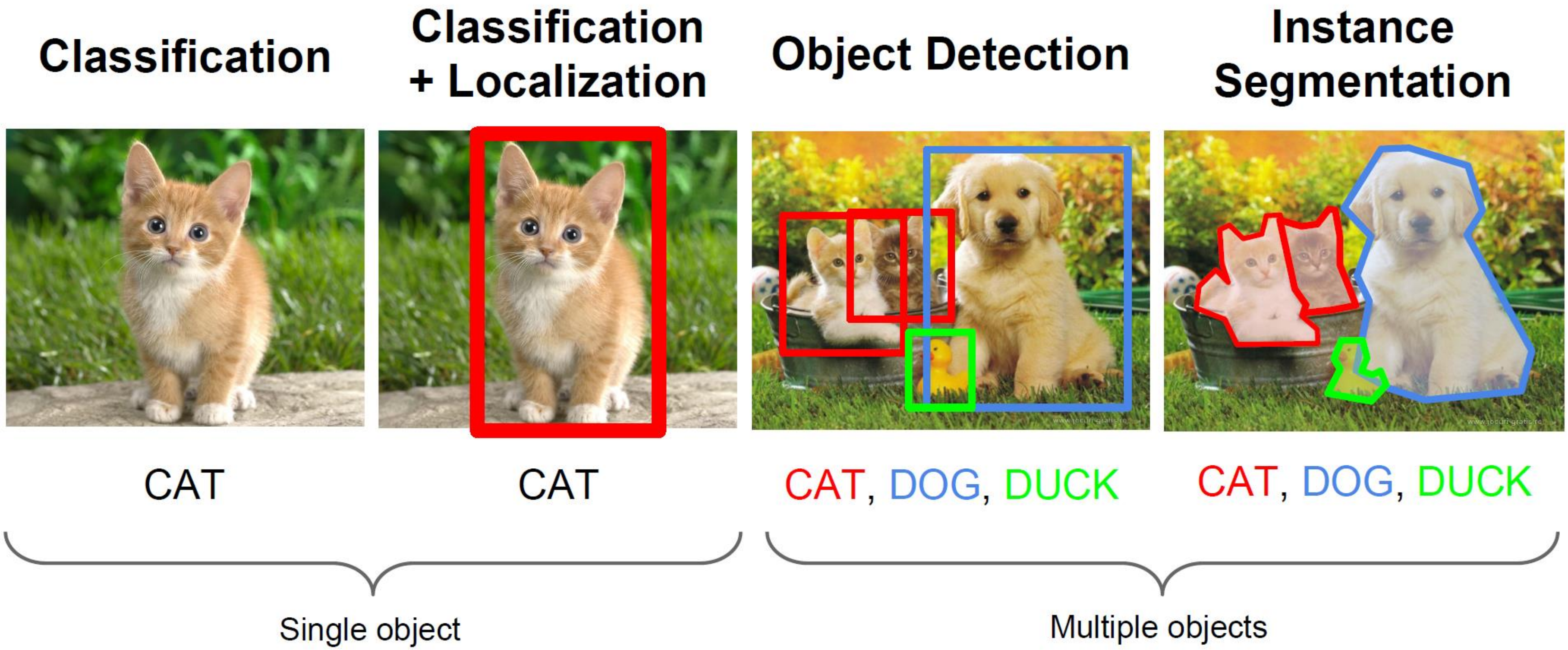


Image filters

- Each kernel gives us a different image filter

Input

Edge detection

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Box mean

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpen

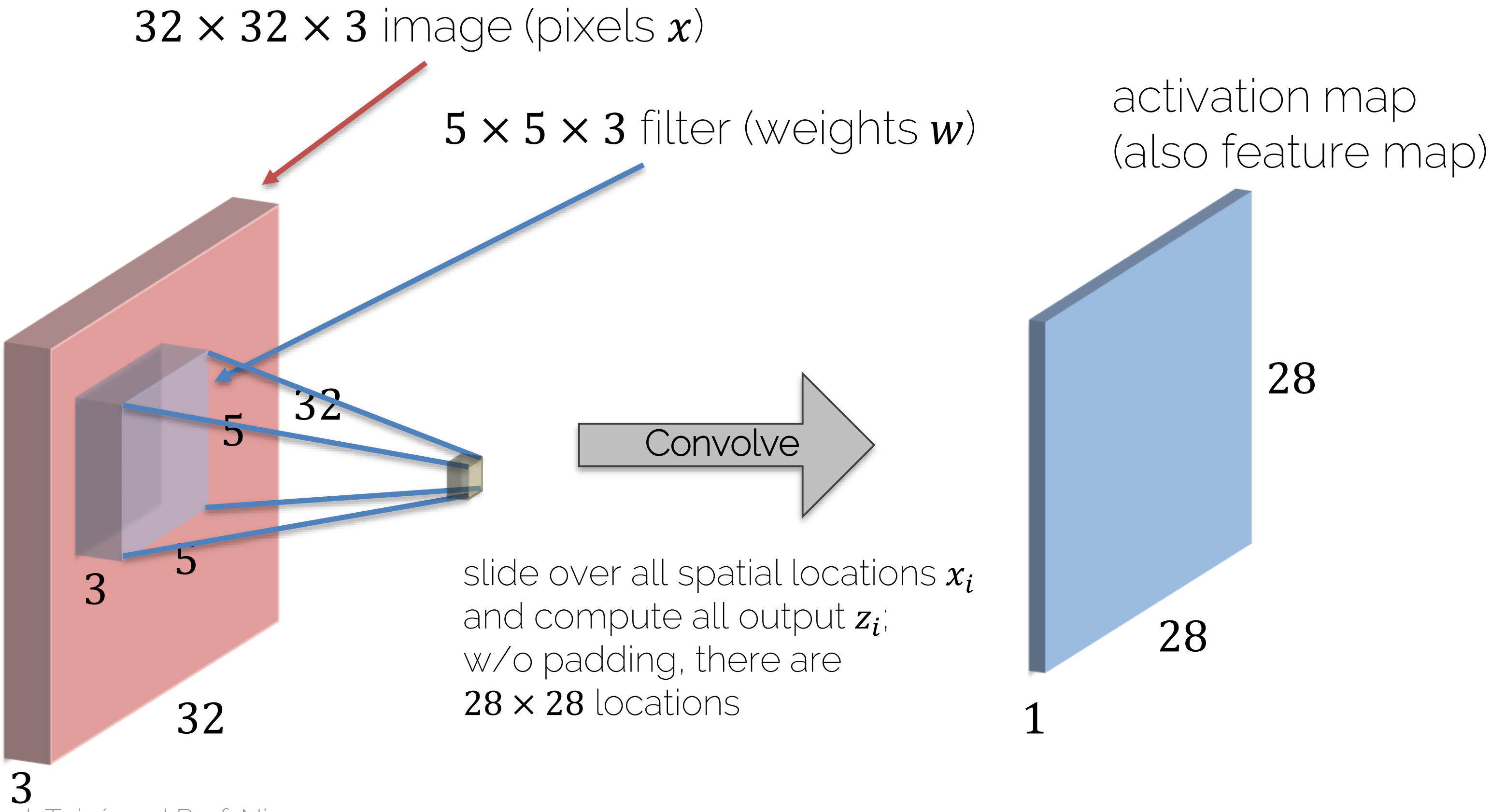
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Gaussian blur

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

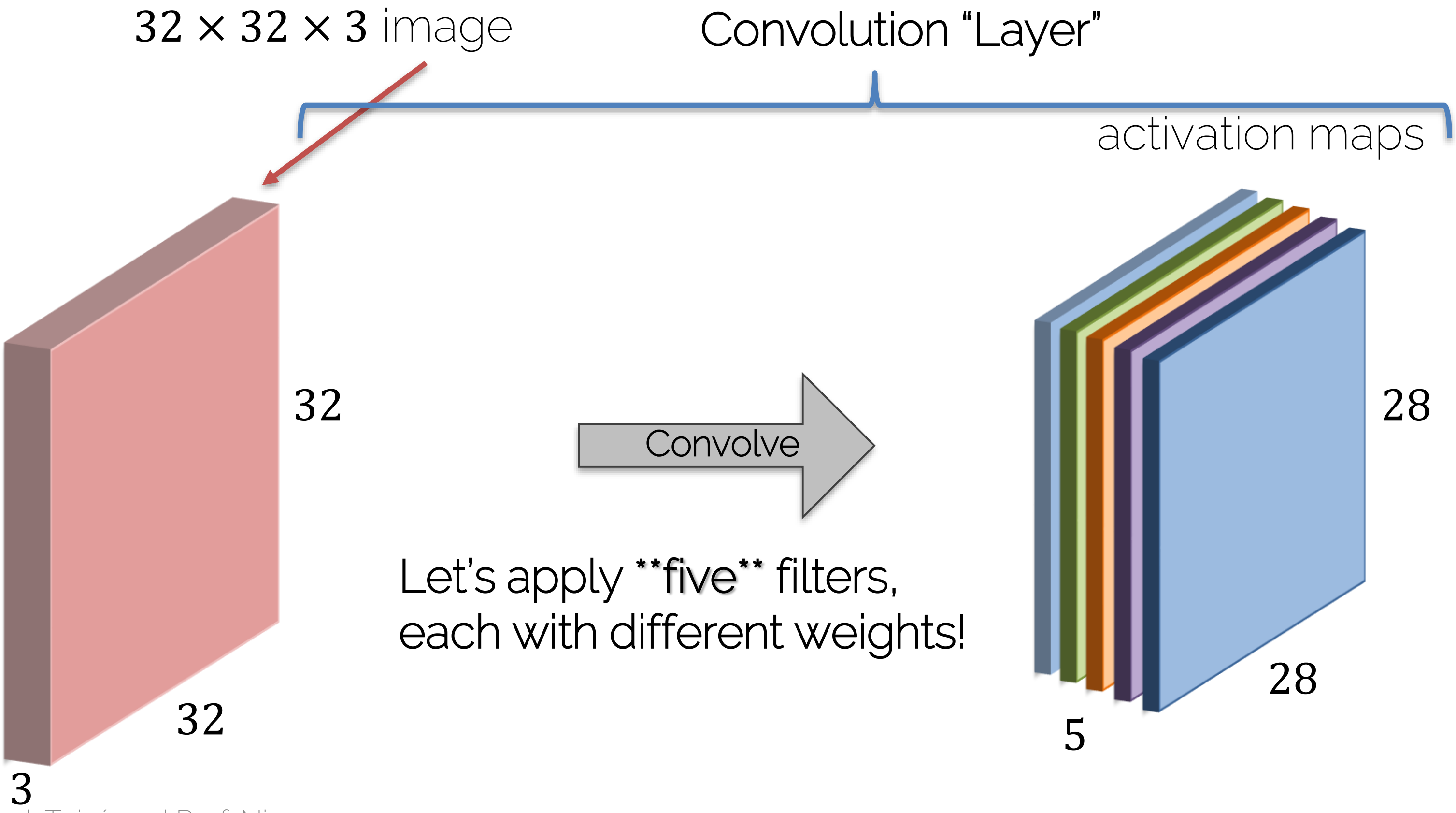
LET'S LEARN THESE FILTERS!

Convolutions on RGB Images



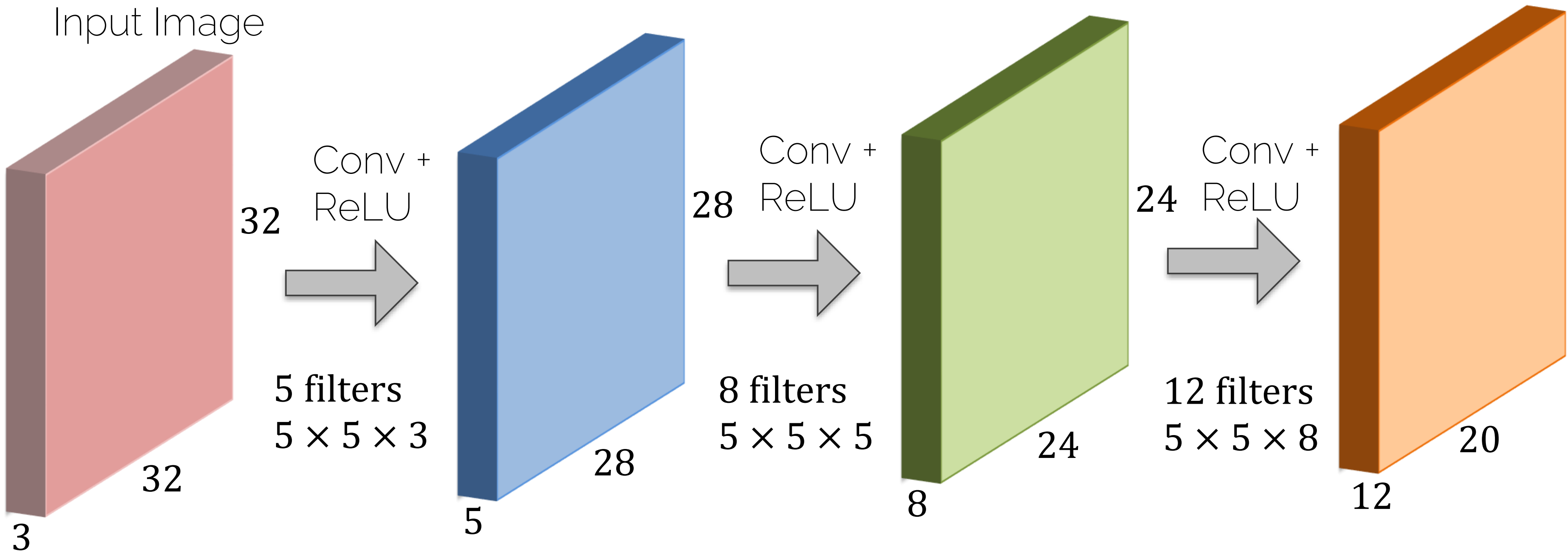
slide over all spatial locations x_i and compute all output z_i; w/o padding, there are 28×28 locations

Convolution Layer

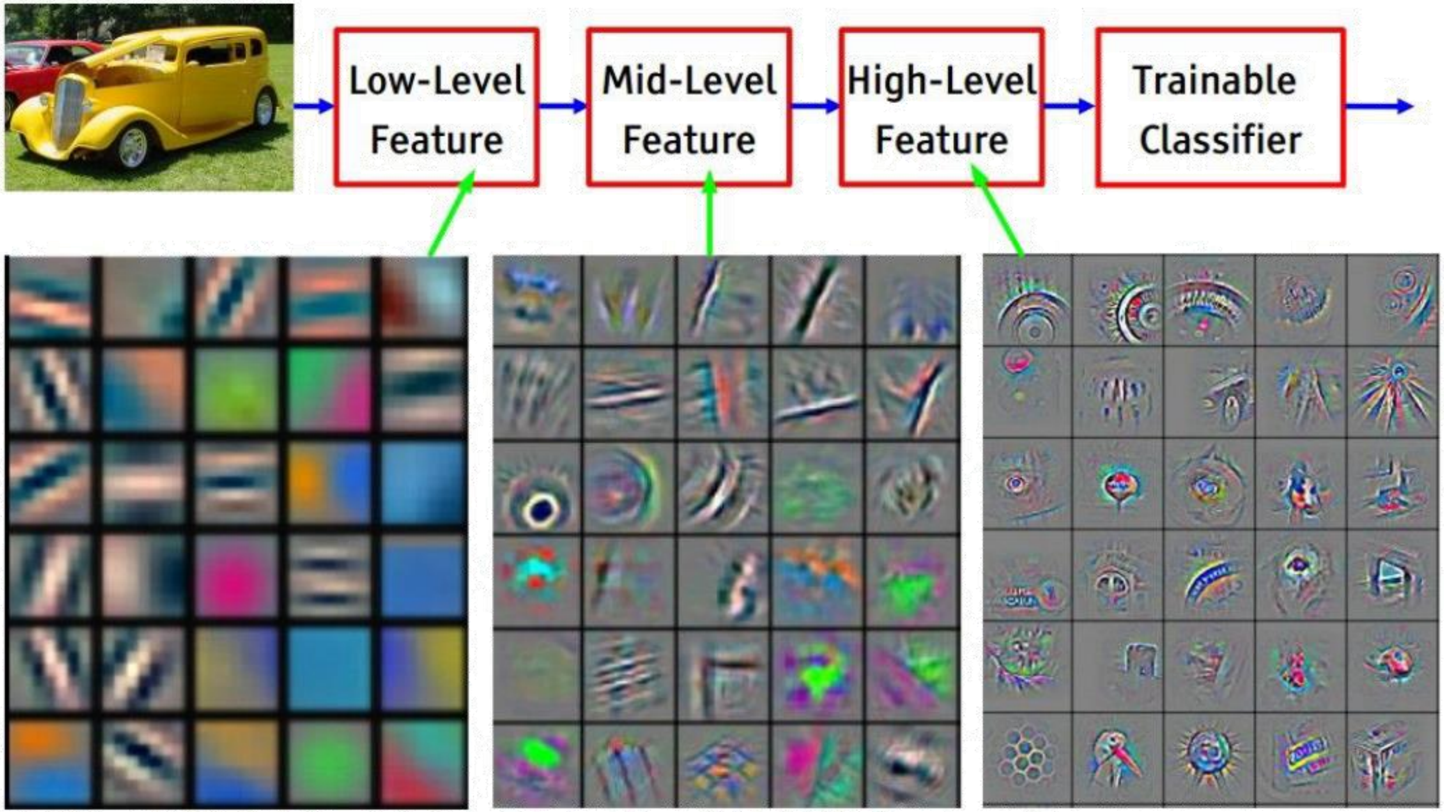


CNN Prototype

ConvNet is concatenation of Conv Layers and activations



CNN learned filters



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

Pooling Layer: Max Pooling

Single depth slice of input

3	1	3	5
6	0	7	9
3	2	1	4
0	2	4	3

Max pool with
2×2 filters and stride 2

'Pooled' output

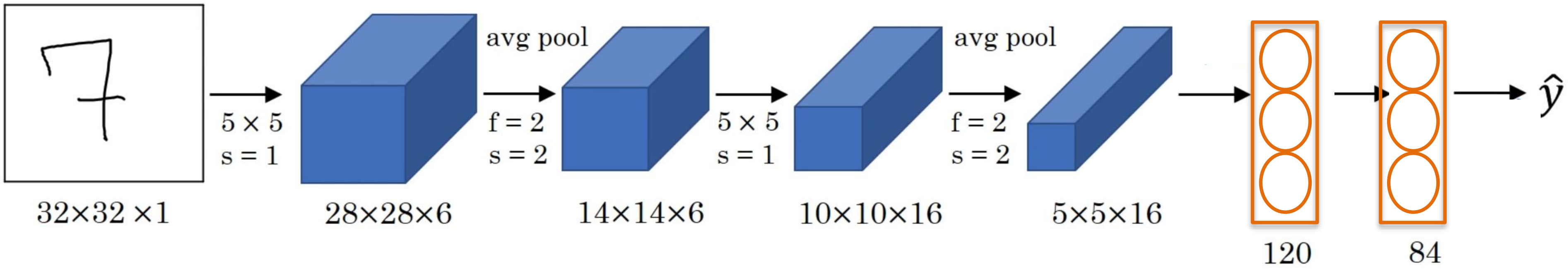
6	9
3	4

Classic CNN architectures

LeNet

- Digit recognition: 10 classes

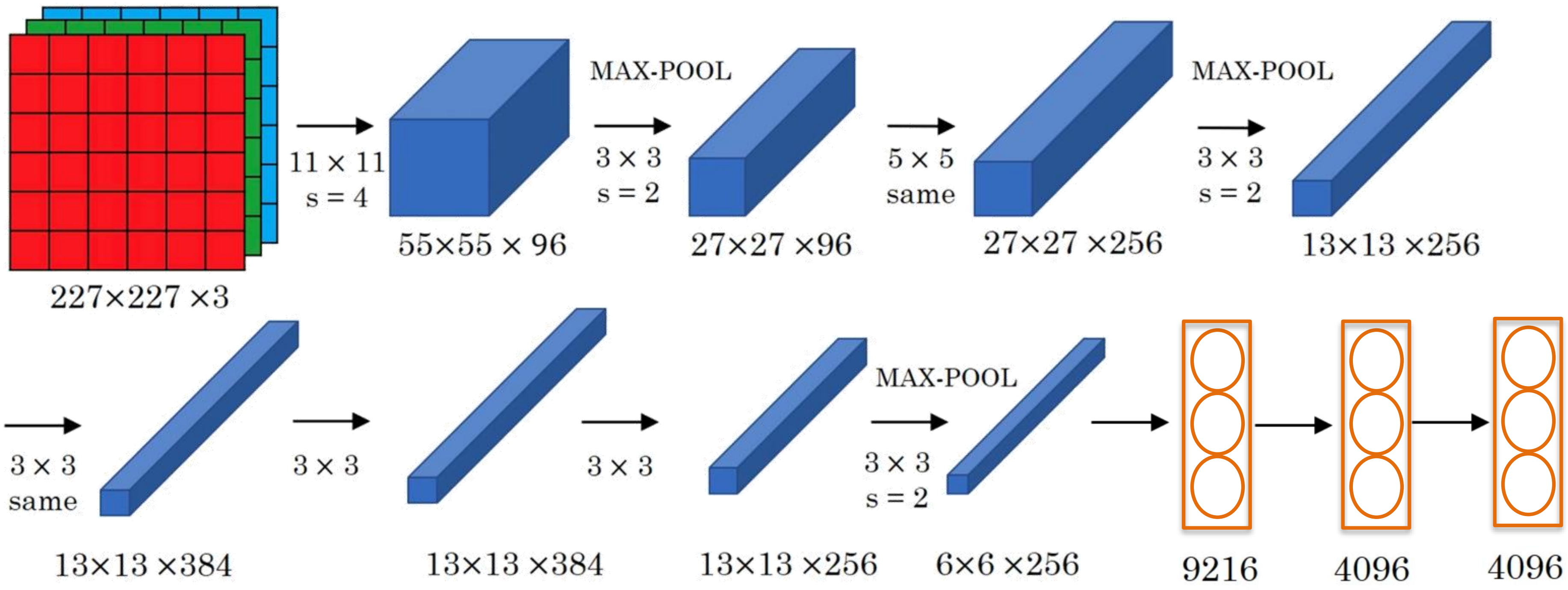
60k parameters



- Conv -> Pool -> Conv -> Pool -> Conv -> FC
- As we go deeper: Width, height ↓ Number of filters ↑

AlexNet

[Krizhevsky et al. 2012]



- Softmax for 1000 classes

VGGNet

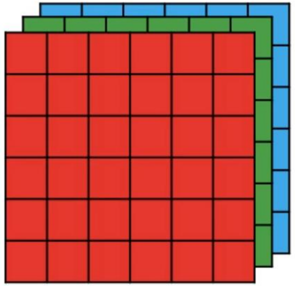
[Simonyan and Zisserman 2014]

- Striving for simplicity
- CONV = 3x3 filters with stride 1, same convolutions
- MAXPOOL = 2x2 filters with stride 2

VGGNet

Conv=3x3,s=1,same
Maxpool=2x2,s=2

Still very common: VGG-16



224×224 ×3

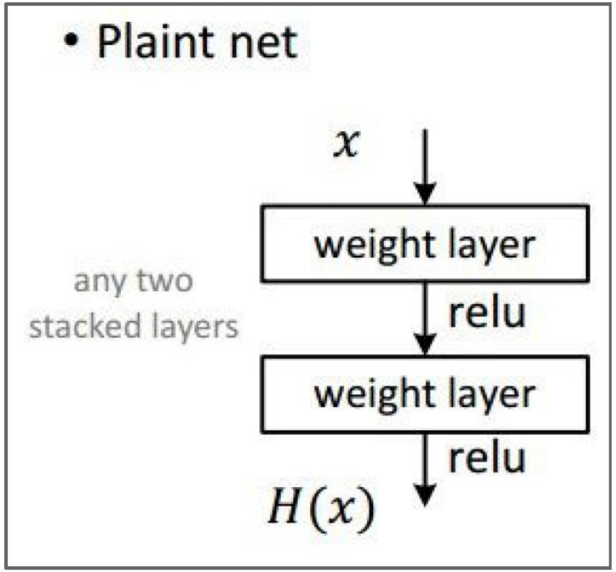
→ [CONV 64] ×2 → 224×224×64 → POOL → 112×112 ×64 → [CONV 128] ×2 → 112×112 ×128 → POOL → 56×56 ×128

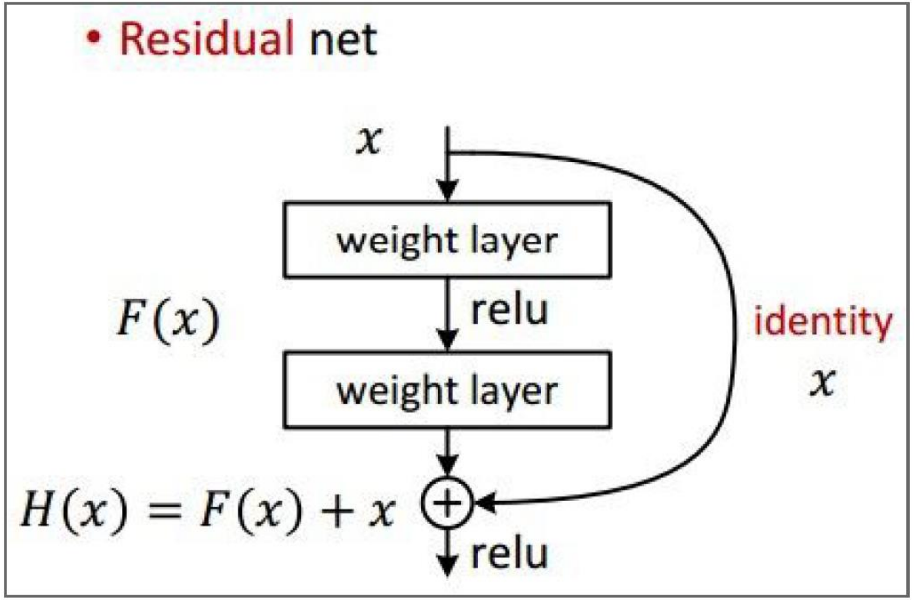
→ [CONV 256] ×3 → 56×56 ×256 → POOL → 28×28 ×256 → [CONV 512] × 3 → 28×28 ×512 → POOL → 14×14×512

→ [CONV 512] ×3 → 14×14 ×512 → POOL → 7×7×512 → FC 4096 → FC 4096 → Softmax 1000

ResNet

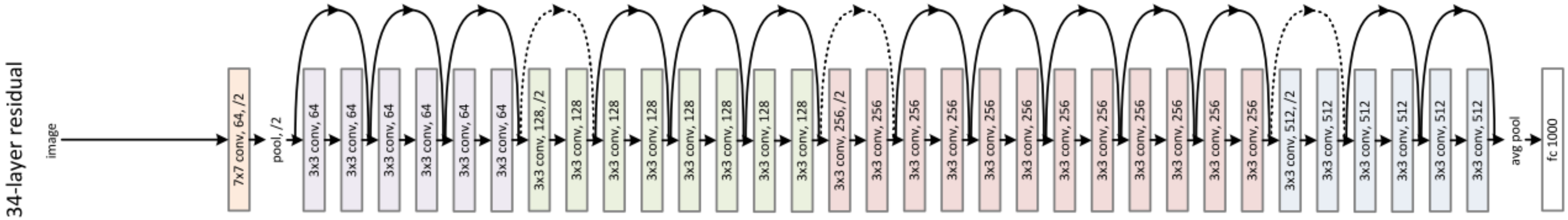
[He et al. 2015]





ResNet

[He et al. 2015]

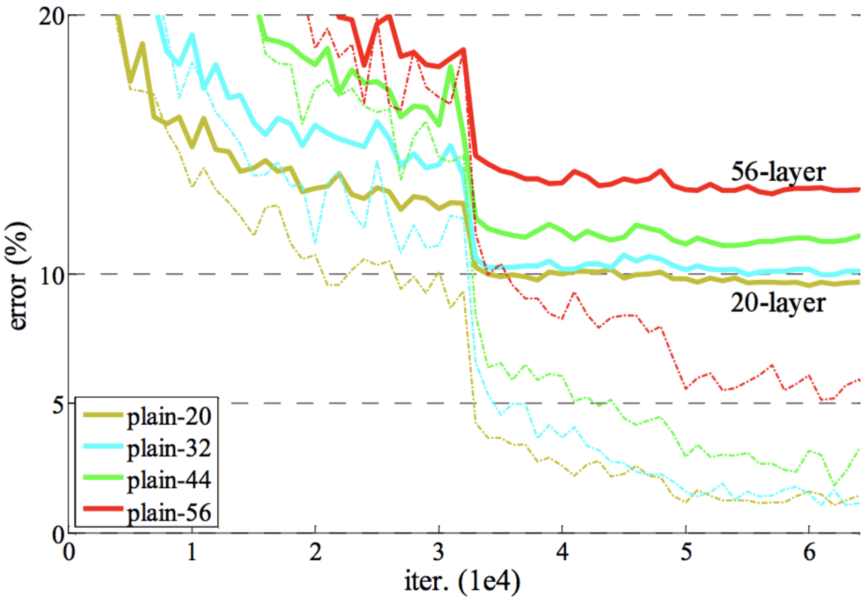


- Xavier/2 initialization
- SGD + Momentum (0.9)
- Learning rate 0.1, divided by 10 when plateau
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout

ResNet

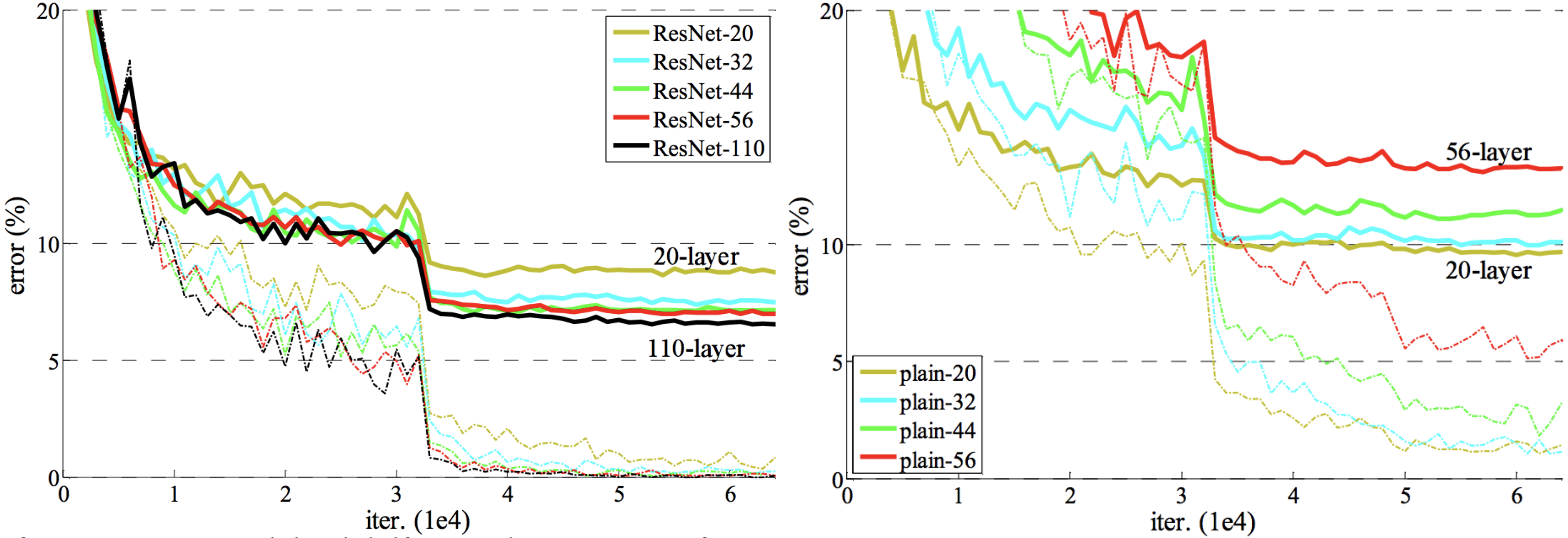
- If we make the network deeper, at some point performance starts to degrade

- Too many parameters, the optimizer cannot properly train the network

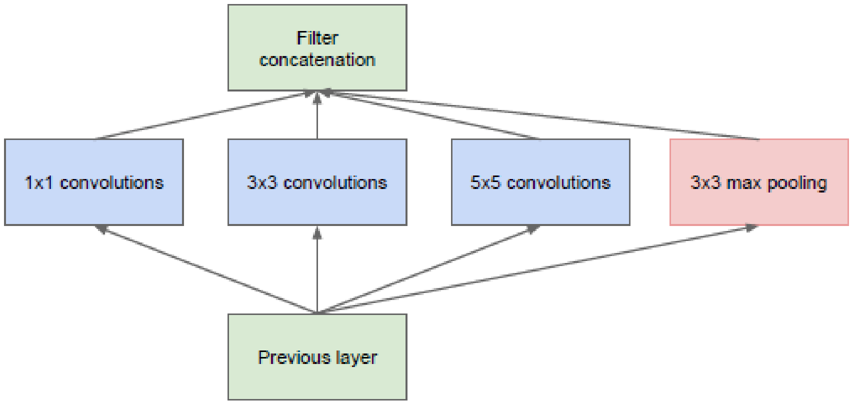


ResNet

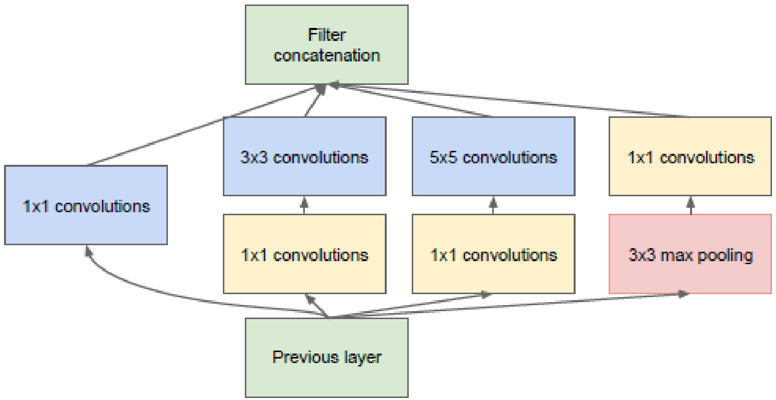
- If we make the network deeper, at some point performance starts to degrade



Inception layer



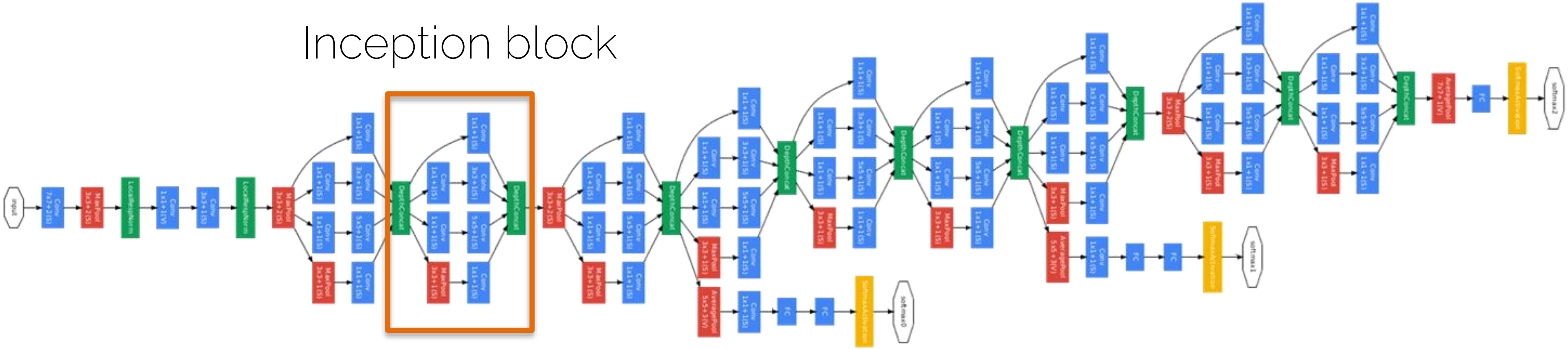
(a) Inception module, naïve version



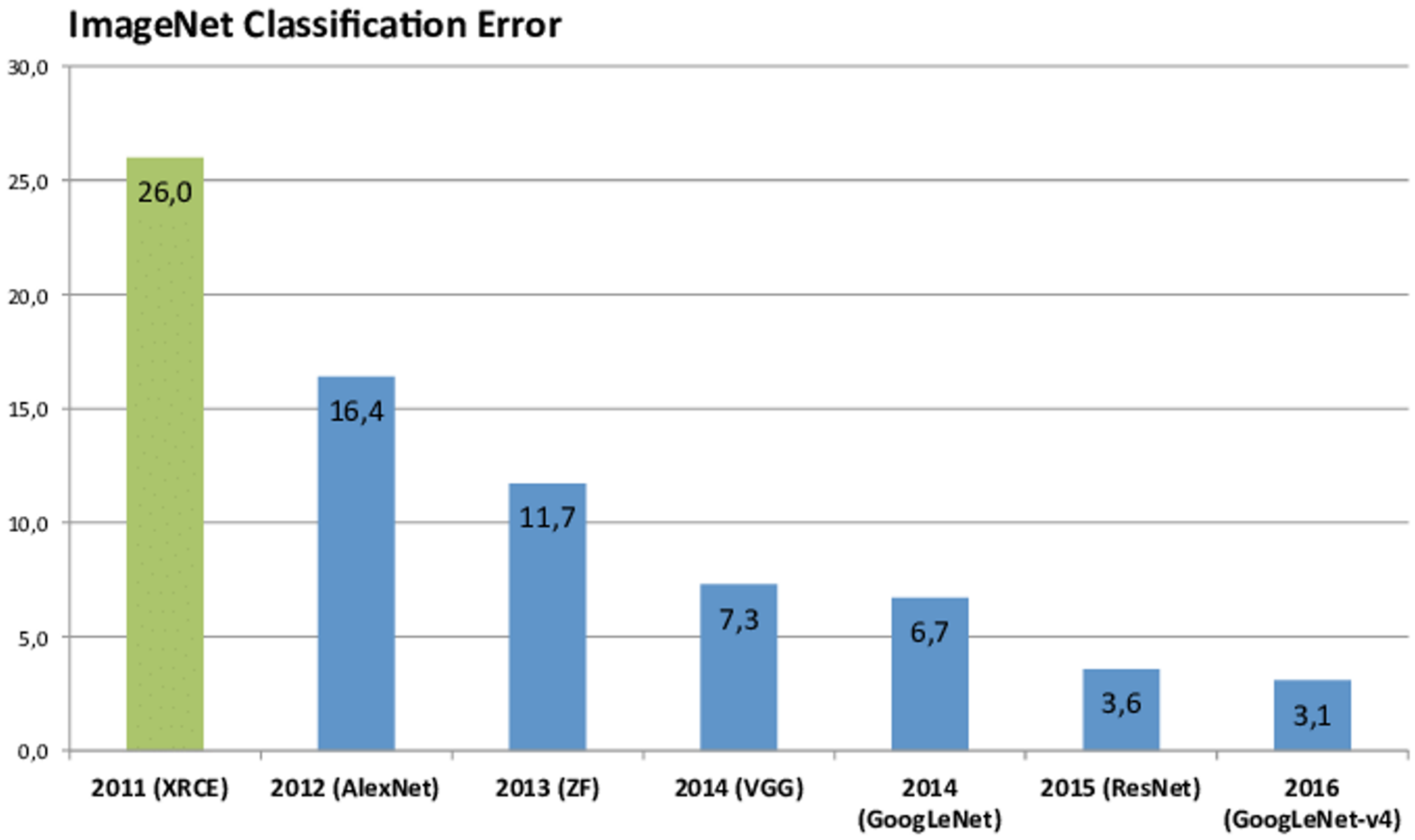
(b) Inception module with dimensionality reduction

GoogLeNet: using the inception layer

[Szegedy et al. 2014]



CNN Architectures



Recurrent Neural Networks

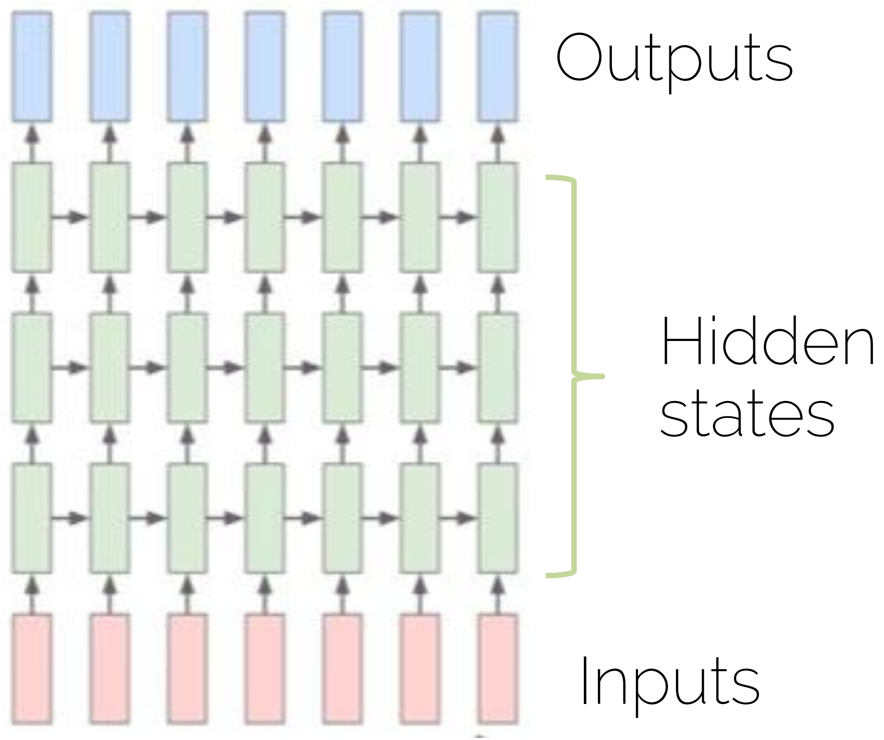
Basic structure of a RNN

- Multi-layer RNN

The hidden state will have its own internal dynamics

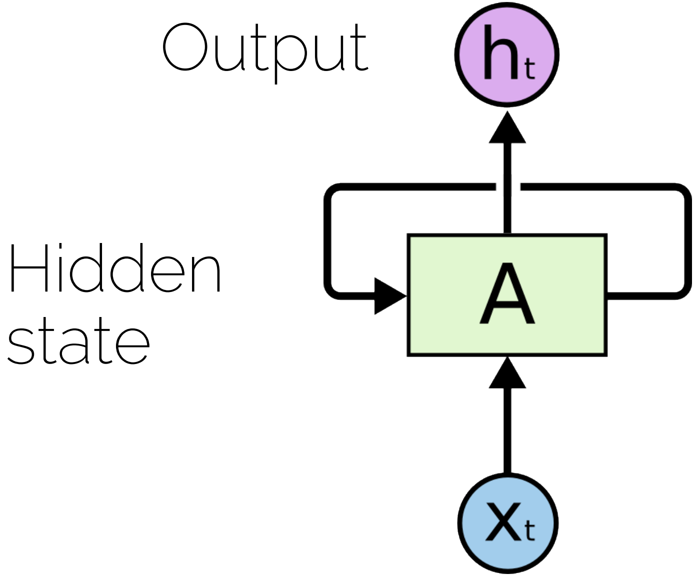
↓

More expressive model!



Basic structure of a RNN

- We want to have notion of “time” or “sequence”



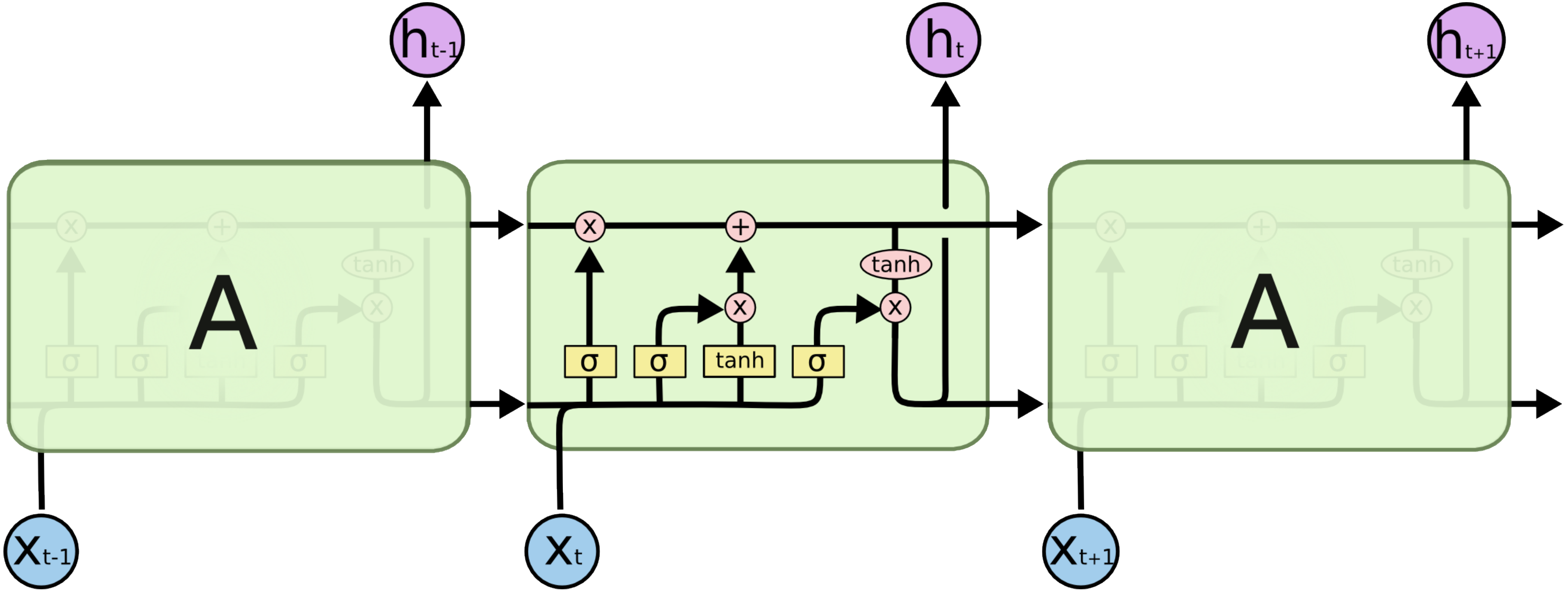
$$\mathbf{A}_t = \boldsymbol{\theta}_c \mathbf{A}_{t-1} + \boldsymbol{\theta}_x \mathbf{x}_t$$

$$\mathbf{h}_t = \boldsymbol{\theta}_h \mathbf{A}_t$$

Same parameters for each time step = generalization!

Long-Short Term Memory Units

- LSTM



ADL4CV Content

Rough Outline

- Lecture 1: introduction
- Lecture 2: advanced architectures (e.g. siamese, capsules, attention)
- Lecture 3: advanced architectures con't
- Lecture 4: Visualization, t-sne, grad-cam (active heatmaps), deep dream, excitation backprop
- Lecture 5: Bayesian Deep Learning
- Lecture 6: Autoencoders, VAE
  Lecture 7: GANs 1: Generative models, GANs.
- Lecture 8: GANs 2: Generative models, GANs
- Lecture 9: CNN++ / Audio<->Visual - autoregressive, pixelcnn
- Lecture 10: RNN -> NLP <-> Visual Q&A (focus on the cross domain: CNN for image, RNN for text) /
- Lecture 11: Multi-dimensional CNN, 3D DL, video DL: pooling vs fully-conv, operations… Self-supervised / unsupervised learning
- Lecture 12: Domain Adaptation / Transfer Learning

How to train your neural network?

Is data loading correct?

- Data output (target): overfit to single training sample (needs to have 100% because it just memorizes input)
  - It's irrespective of input !!!

- Data input: overfit to a handful (e.g., 4) training samples
  - It's now conditioned on input data

- Save and re-load data from PyTorch / TensorFlow

Network debugging

- Move from overfitting to a hand-full of samples
  - 5, 10, 100, 1000…
  - At some point, we should see generalization

- Apply common sense: can we overfit to the current number of samples?

- Always be aware of network parameter count!

Check timings

- How long does each iteration take?
  - **Get precise timings!!!**
  - If an iteration takes > 500ms, things get dicey…
- Where is the bottleneck: data loading vs backprop?
  - Speed up data loading: smaller resolutions, compression, train from SSD – e.g., network training is good idea
  - Speed up backprop:
- Estimate total timings: how long until you see some pattern? How long till convergence?

Network Architecture

- 100% mistake so far: "let's use super big network and train for two weeks and we see where we stand." [because we desperately need those 2%...]

- Start with simplest network possible: rule of thumb divide #layers you started with by 5.

- Get debug cycles down – ideally, minutes!!!

Debugging

- Need train/val/test curves
  - Evaluation needs to be consistent!
  - Numbers need to be comparable

- Only make one change at a time
  - "I've added 5 more layers and double the training size, and now I also trained 5 days longer" – it's better, but WHY?

Overfitting

ONLY THINKG ABOUT THIS ONCE YOU'R TRAINING LOSS GOES DOWN AND YOU CAN OVERFIT!

Typically try this order:

- Network too big – makes things also faster ☺
- More regularization; e.g., weight decay
- Not enough data - makes things slower!
- Dropout - makes things slower!
- Guideline: make training harder -> generalize better

Pushing the limits!

PROCEED ONLY IF YOU GENERALIZE AND YOU ADDRESSED OVERFITTING ISSUES!

- Bigger network -> more capacity, more power - needs also more data!
- Better architecture -> ResNet is typically standard, but InceptionNet architectures perform often better (e.g., InceptionNet v4, XceptionNet, etc.)
- Schedules for learning rate decay
- Class-based re-weighting (e.g., give under-represented classes higher weight)
- Hyperparameter tuning: e.g., grid search; apply common sense!

Bad signs…

- Train error doesn't go down…
- Validation error doesn't go down… (ahhh we don't learn)
- Validation performs better than train… (trust me, this scenario is very unlikely – unless you have a bug ☺)
- Test on train set is different error than train… (forgot dropout?)
- Often people mess up the last batch in an epoch…

- You are training set contains test data…
- You debug your algorithm on test data…

“Most common” neural net mistakes

1) you didn't try to overfit a single batch first.
2) you forgot to toggle train/eval mode for the net.
3) you forgot to .zero_grad() (in pytorch) before .backward().
4) you passed softmaxed outputs to a loss that expects raw logits.
5) you didn't use bias=False for your Linear/Conv2d layer when using BatchNorm, or conversely forget to include it for the output layer

Credit: A. Karpathy

Next lecture

- Next Monday: advanced architectures

- Keep projects in mind!
  - Start actively discussing -> reach out to us if you have questions!