High Level Computer Vision

Optimization, Regularization, Recurrent Neural Networks

Bernt Schiele - schiele@mpi-inf.mpg.de
Mario Fritz - fritz@cispa.saarland

https://www.mpi-inf.mpg.de/hlcv

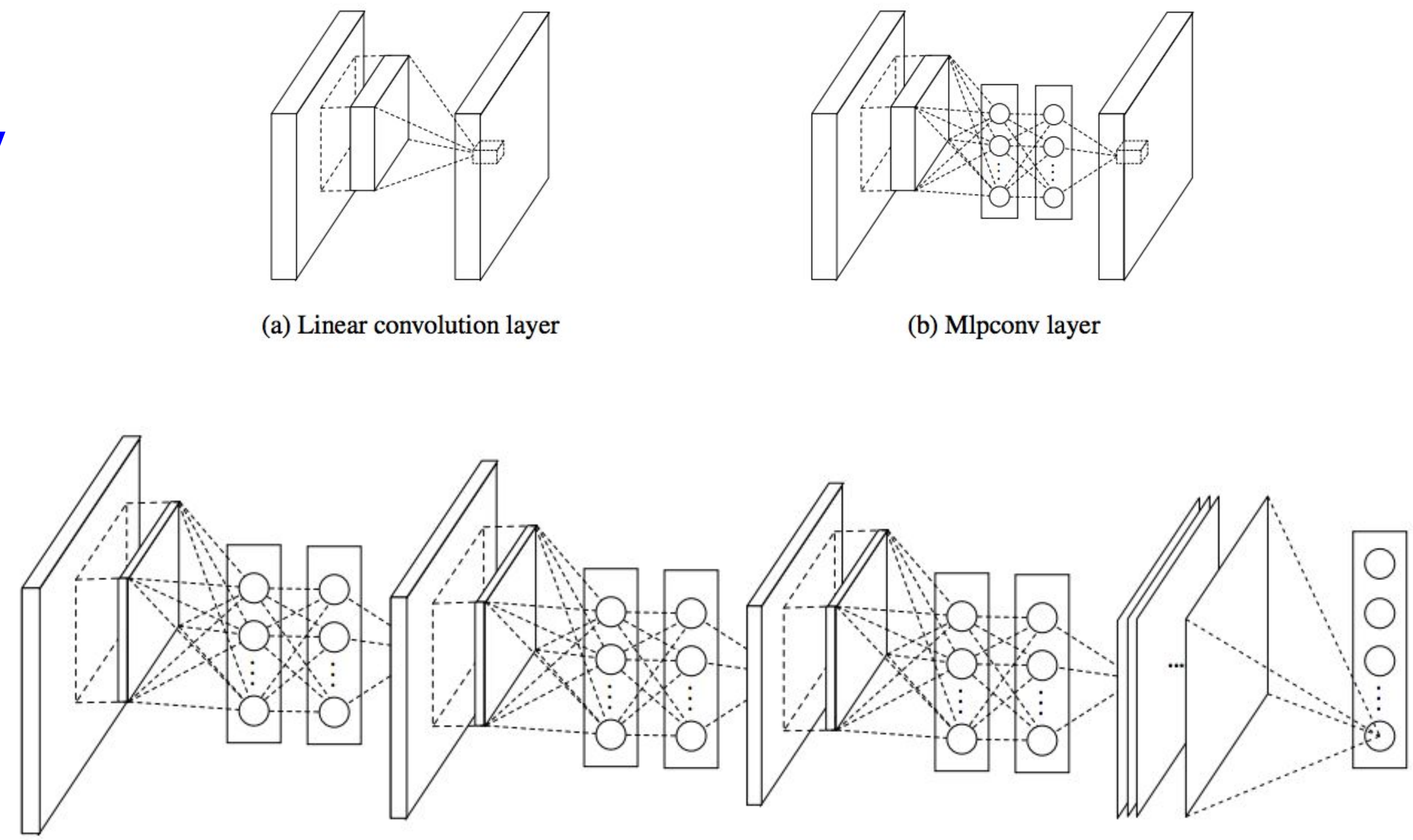
Other architectures to know...

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Network in Network (NiN)

[Lin et al. 2014]

- Mlpconv layer with "micronetwork" within each conv layer to compute more abstract features for local patches
- Micronetwork uses multilayer perceptron (FC, i.e. 1x1 conv layers)
- Precursor to GoogLeNet and ResNet "bottleneck" layers
- Philosophical inspiration for GoogLeNet



Figures copyright Lin et al., 2014. Reproduced with permission.

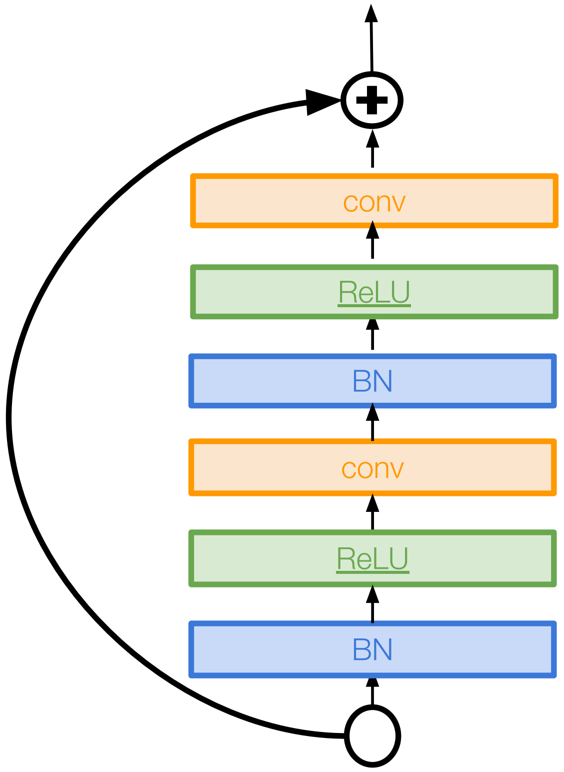
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Improving ResNets...

Identity Mappings in Deep Residual Networks

[He et al. 2016]

- Improved ResNet block design from creators of ResNet
- Creates a more direct path for propagating information throughout network (moves activation to residual mapping pathway)
- Gives better performance



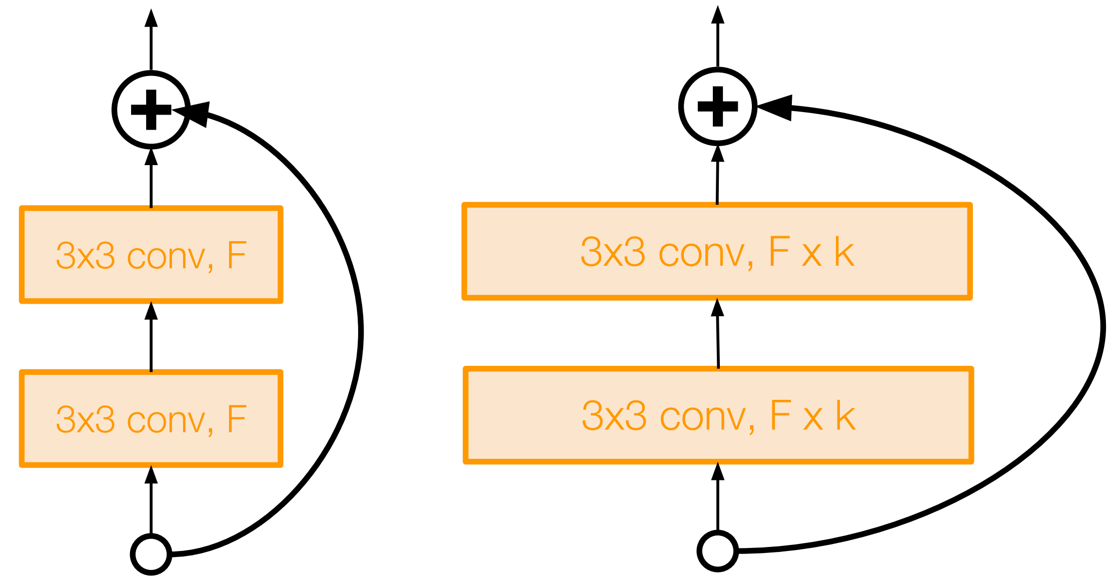
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Improving ResNets...

Wide Residual Networks

[Zagoruyko et al. 2016]

- Argues that residuals are the important factor, not depth
- User wider residual blocks (F x k filters instead of F filters in each layer)
- 50-layer wide ResNet outperforms 152-layer original ResNet
- Increasing width instead of depth more computationally efficient (parallelizable)



Basic residual block Wide residual block

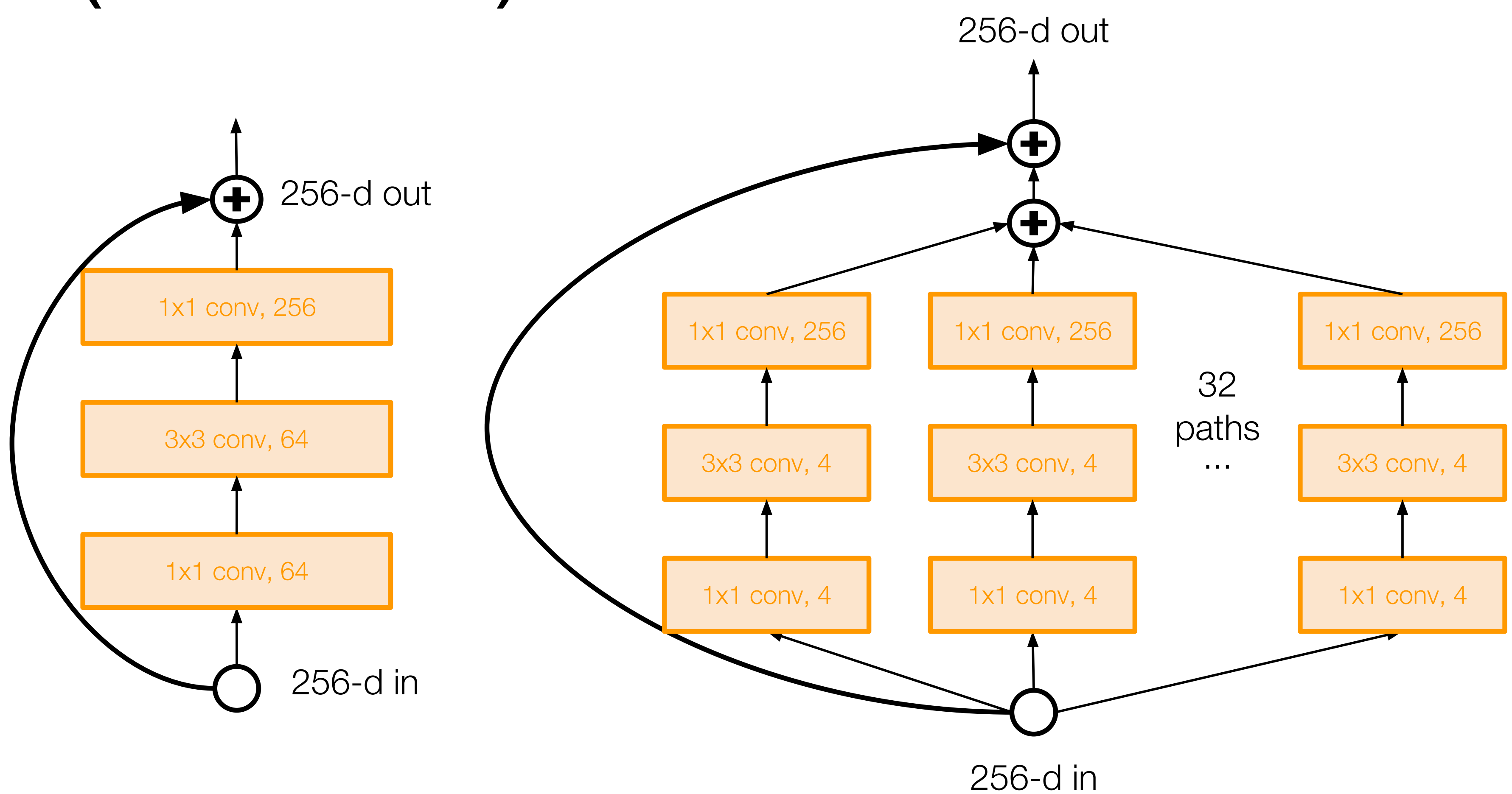
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Improving ResNets...

Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

[Xie et al. 2016]

- Also from creators of ResNet
- Increases width of residual block through multiple parallel pathways ("cardinality")
- Parallel pathways similar in spirit to Inception module



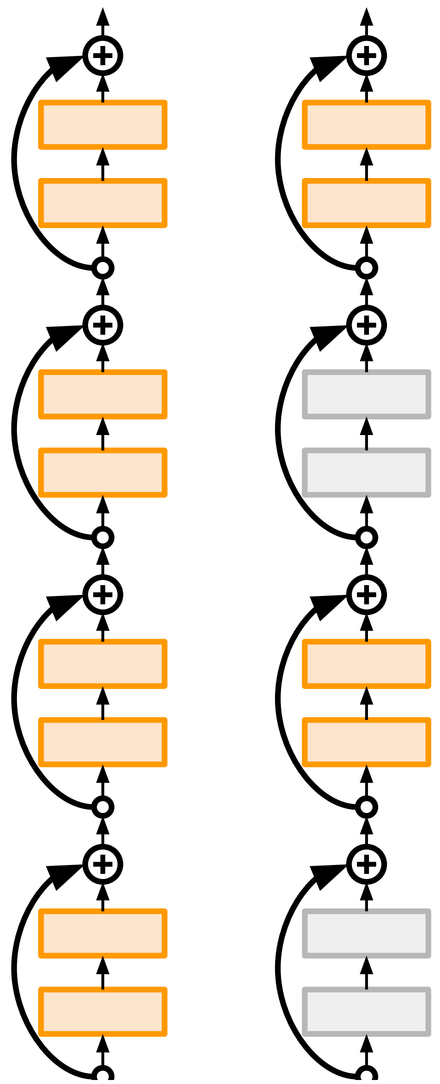
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Improving ResNets...

Deep Networks with Stochastic Depth

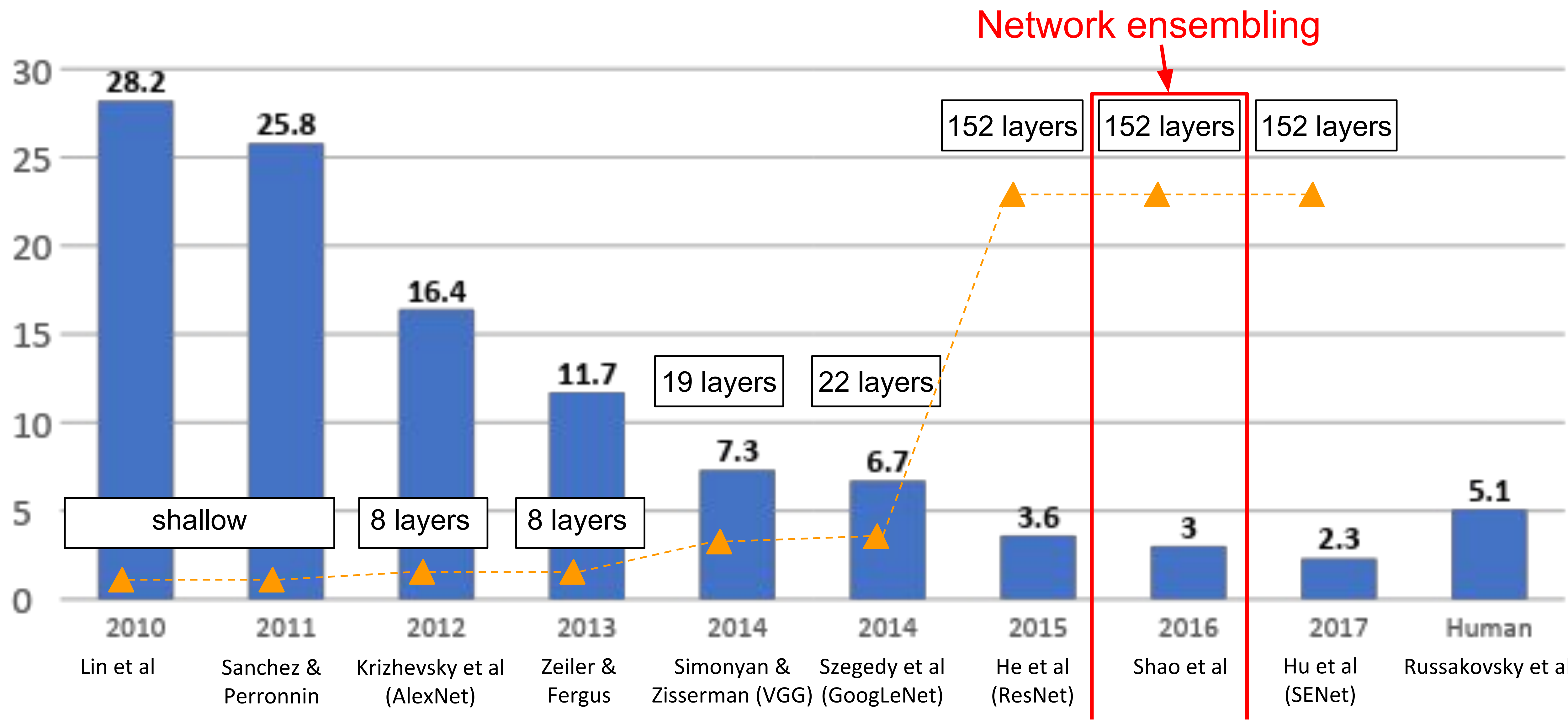
[Huang et al. 2016]

- Motivation: reduce vanishing gradients and training time through short networks during training
- Randomly drop a subset of layers during each training pass
- Bypass with identity function
- Use full deep network at test time



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Improving ResNets...

“Good Practices for Deep Feature Fusion”

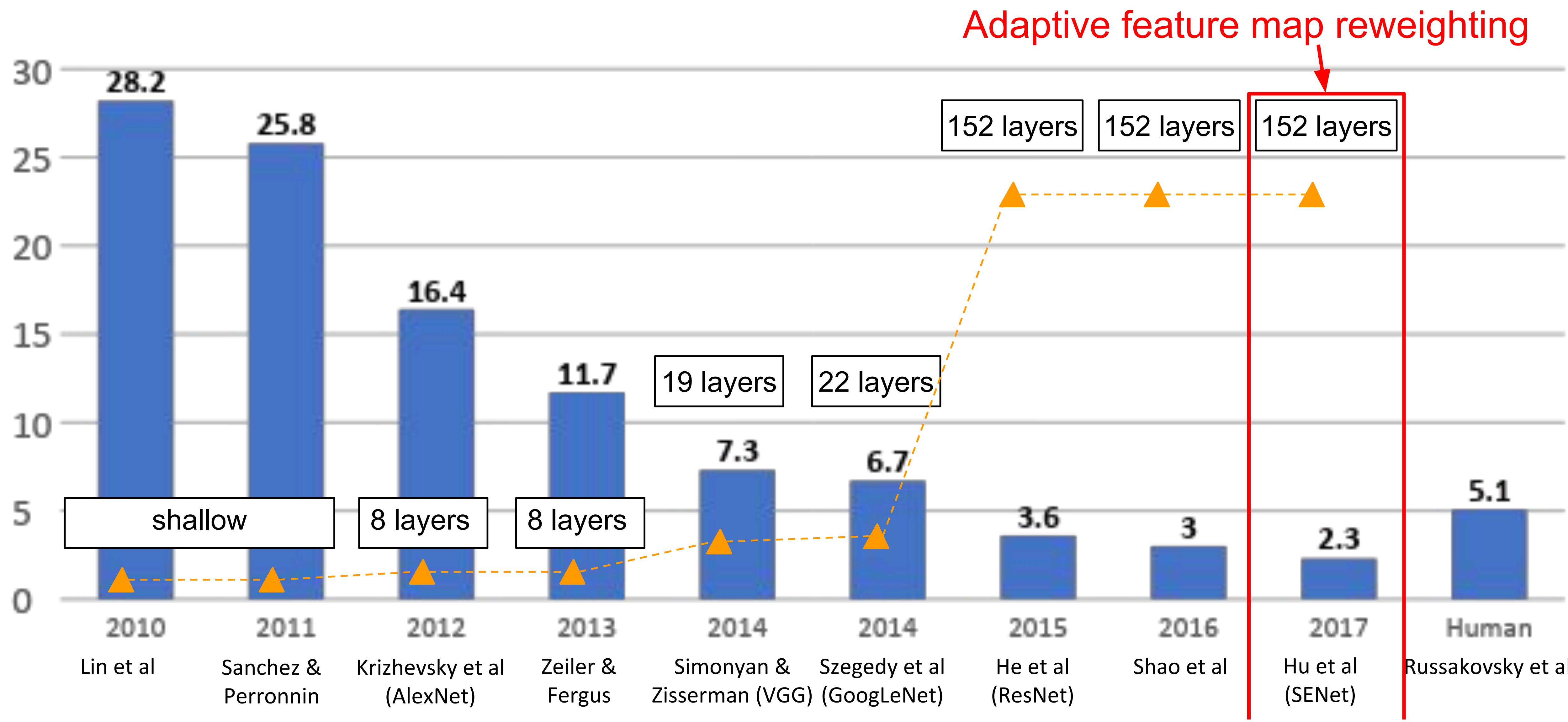
[Shao et al. 2016]

- Multi-scale ensembling of Inception, Inception-Resnet, Resnet, Wide Resnet models
- ILSVRC’16 classification winner

	Inception-v3	Inception-v4	Inception-Resnet-v2	Resnet-200	Wrn-68-3	Fusion（Val.）	Fusion（Test）
Err. (%)	4.20	4.01	3.52	4.26	4.65	2.92 (-0.6)	2.99

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



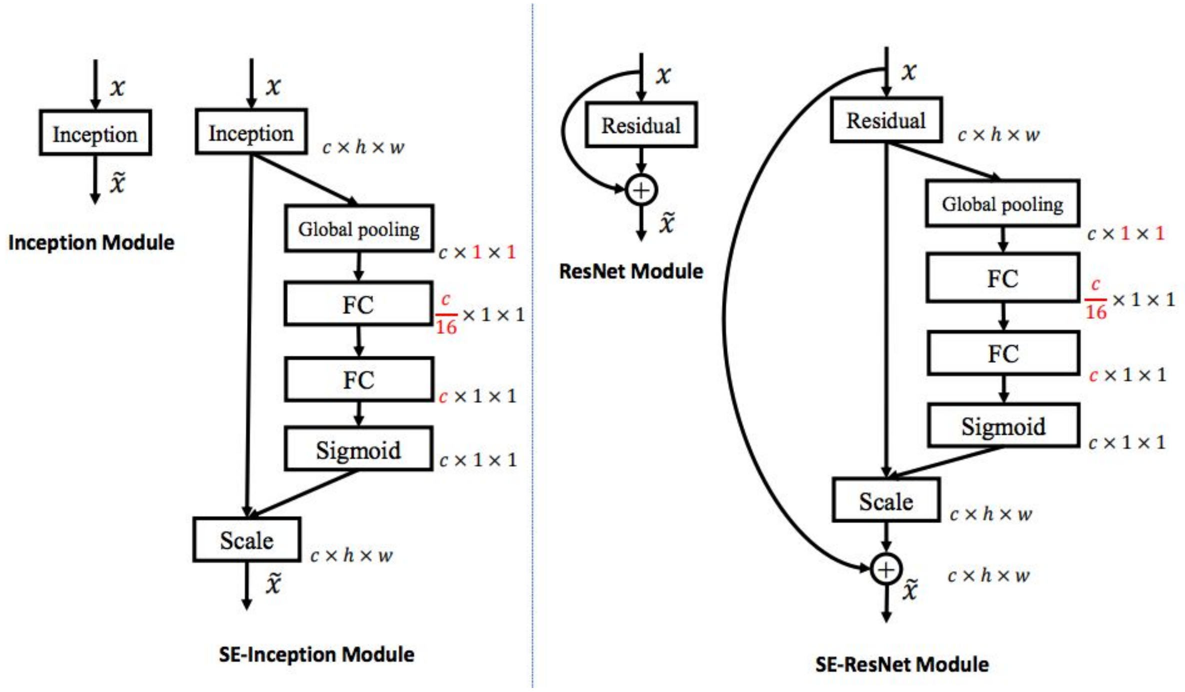
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

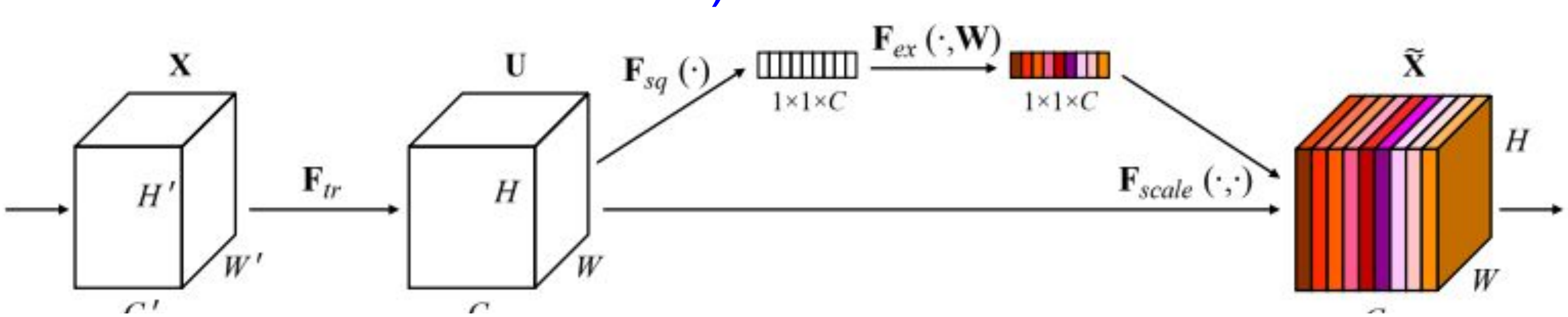
Improving ResNets...

Squeeze-and-Excitation Networks (SENet)

[Hu et al. 2017]

- Add a "feature recalibration" module that learns to adaptively reweight feature maps
- Global information (global avg. pooling layer) + 2 FC layers used to determine feature map weights
- ILSVRC'17 classification winner (using ResNeXt-152 as a base architecture)





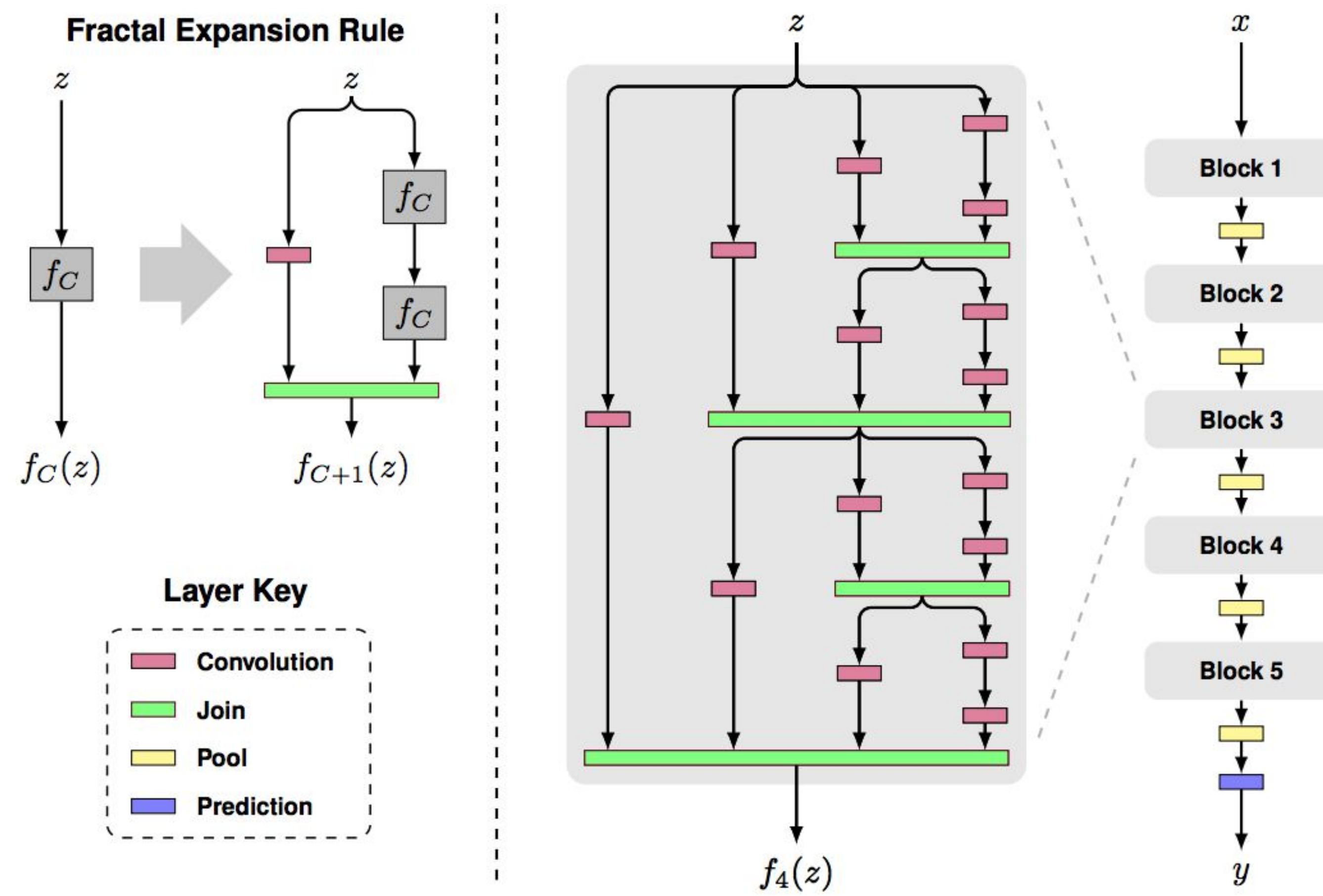
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Beyond ResNets...

FractalNet: Ultra-Deep Neural Networks without Residuals

[Larsson et al. 2017]

- Argues that key is transitioning effectively from shallow to deep and residual representations are not necessary
- Fractal architecture with both shallow and deep paths to output
- Trained with dropping out sub-paths
- Full network at test time



Figures copyright Larsson et al., 2017. Reproduced with permission.

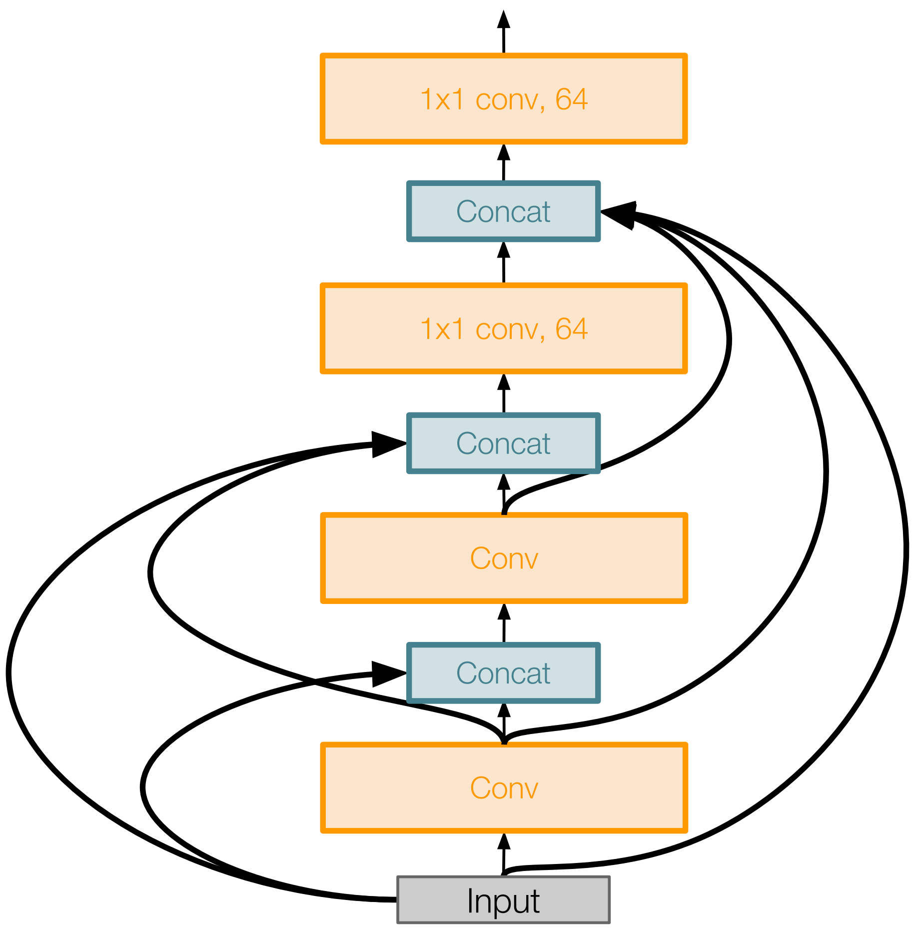
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Beyond ResNets...

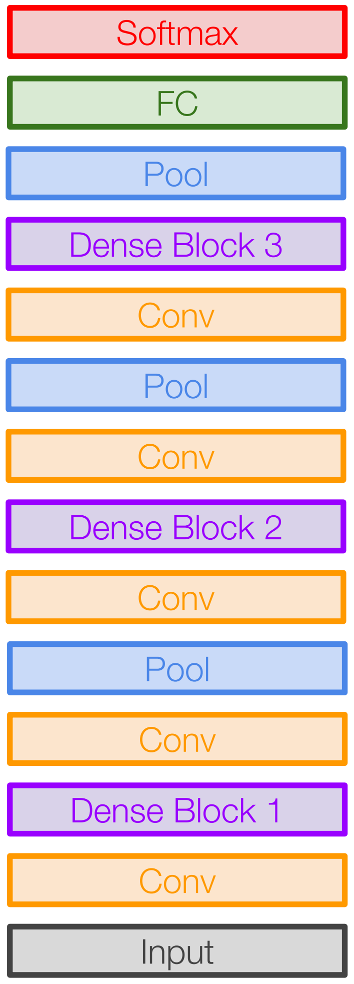
Densely Connected Convolutional Networks

[Huang et al. 2017]

- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse



Dense Block



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Efficient networks...

SqueezeNet: AlexNet-level Accuracy With 50x Fewer Parameters and <0.5Mb Model Size

[Iandola et al. 2017]

- Fire modules consisting of a 'squeeze' layer with 1x1 filters feeding an 'expand' layer with 1x1 and 3x3 filters
- AlexNet level accuracy on ImageNet with 50x fewer parameters
- Can compress to 510x smaller than AlexNet (0.5Mb)

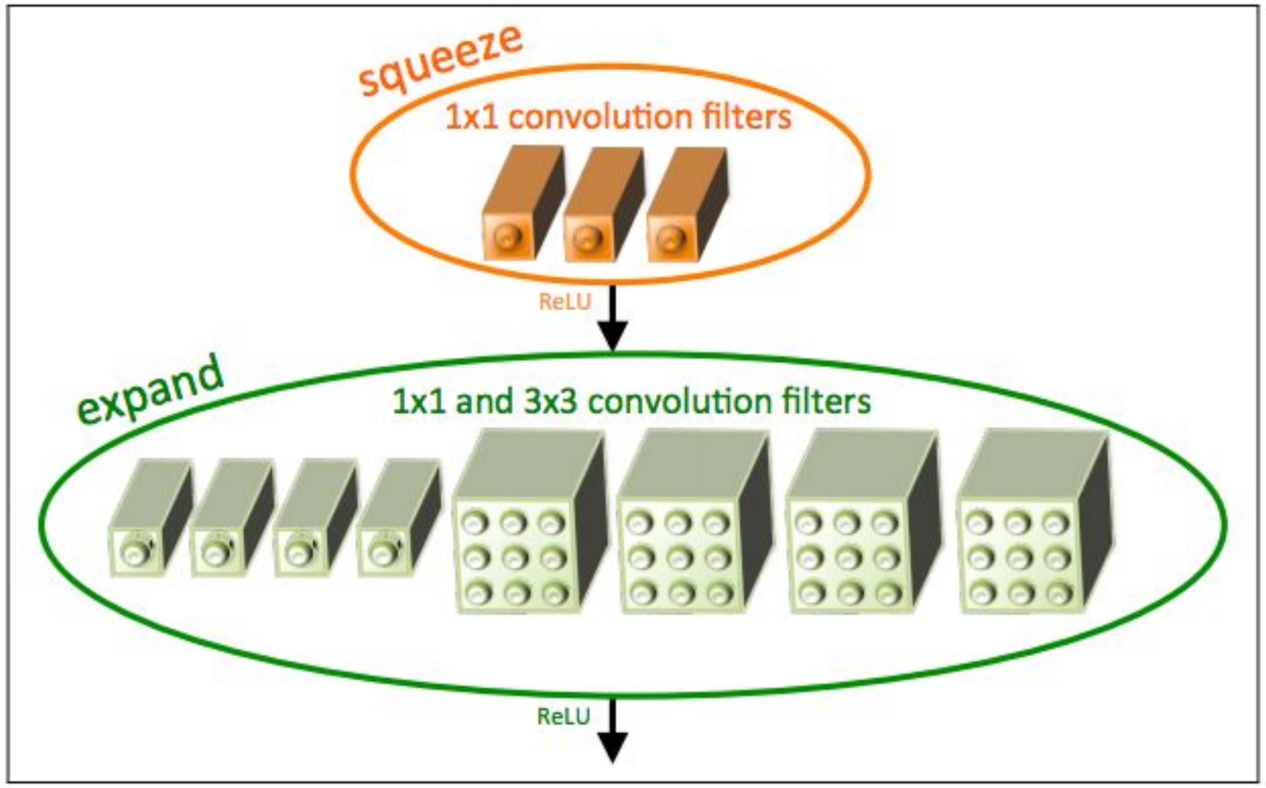


Figure copyright Iandola, Han, Moskewicz, Ashraf, Dally, Keutzer, 2017. Reproduced with permission.

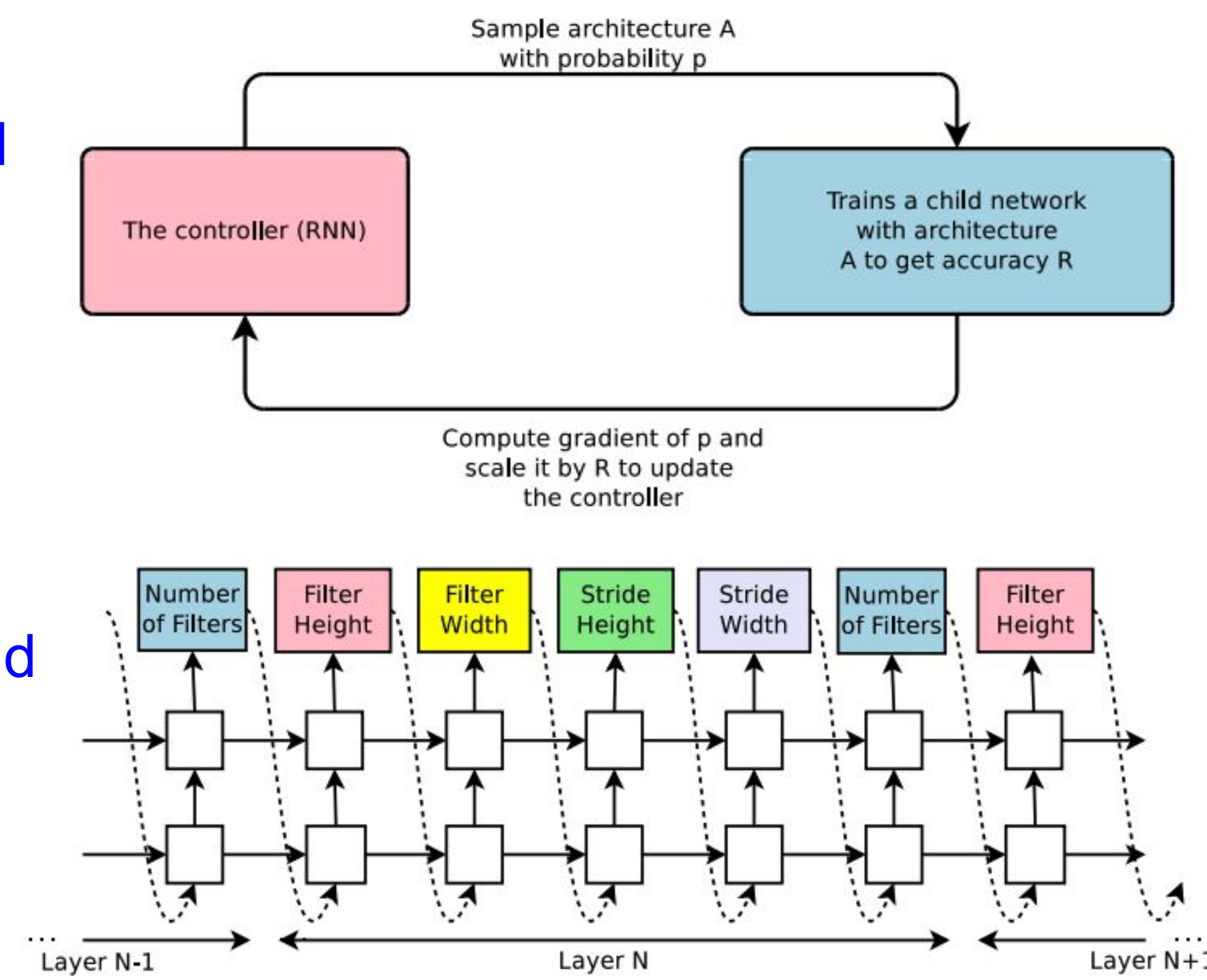
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Meta-learning: Learning to learn network architectures...

Neural Architecture Search with Reinforcement Learning (NAS)

[Zoph et al. 2016]

- "Controller" network that learns to design a good network architecture (output a string corresponding to network design)
- Iterate:
  1) Sample an architecture from search space
  2) Train the architecture to get a "reward" R corresponding to accuracy
  3) Compute gradient of sample probability, and scale by R to perform controller parameter update (i.e. increase likelihood of good architecture being sampled, decrease likelihood of bad architecture)



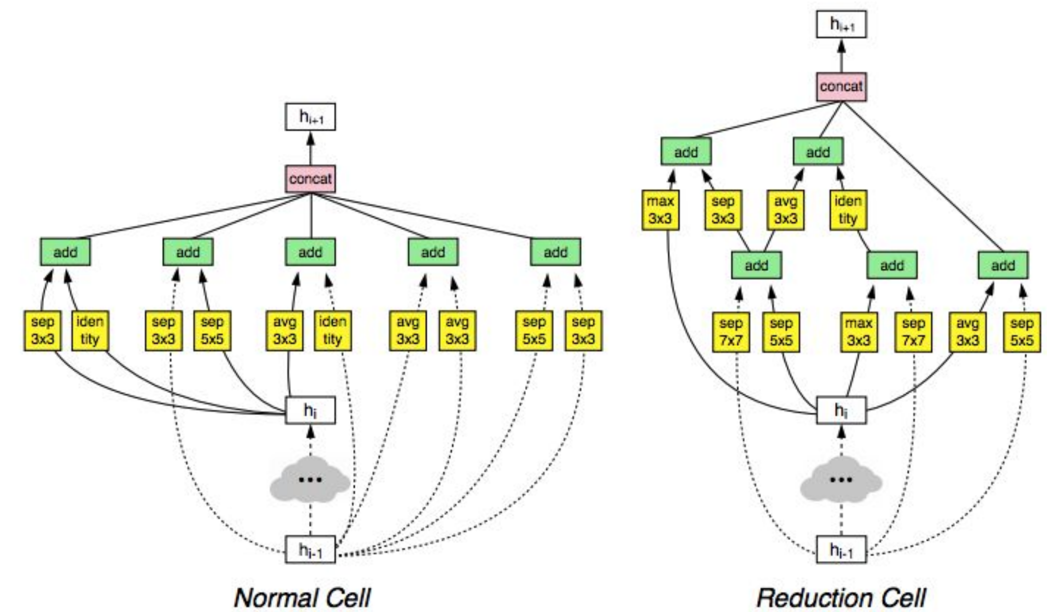
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

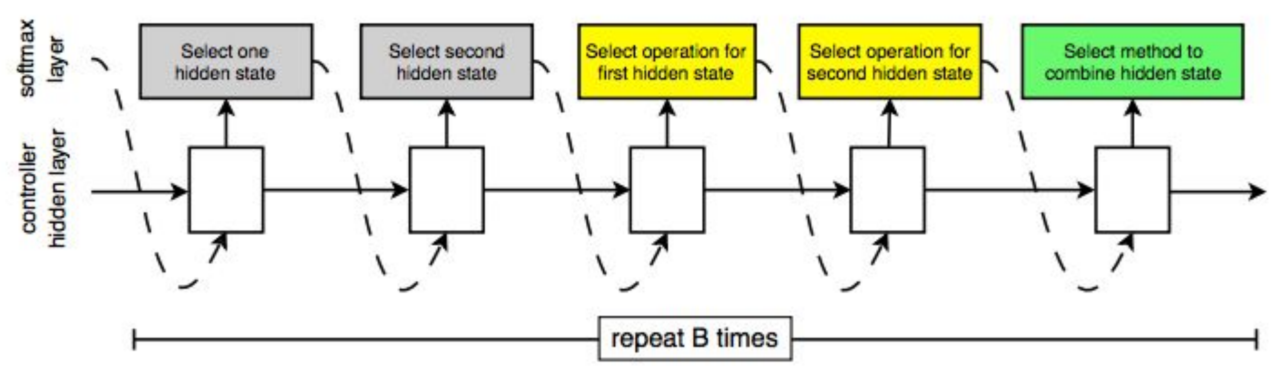
Meta-learning: Learning to learn network architectures...

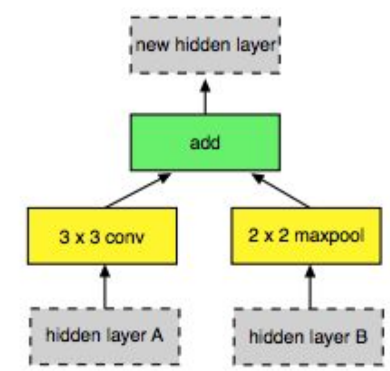
Learning Transferable Architectures for Scalable Image Recognition

[Zoph et al. 2017]

- Applying neural architecture search (NAS) to a large dataset like ImageNet is expensive
- Design a search space of building blocks ("cells") that can be flexibly stacked
- NASNet: Use NAS to find best cell structure on smaller CIFAR-10 dataset, then transfer architecture to ImageNet







slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Summary: CNN Architectures

Case Studies

- AlexNet
- VGG
- GoogLeNet
- ResNet

Also....

- NiN (Network in Network)
- Wide ResNet
- ResNeXT
- Stochastic Depth
- Squeeze-and-Excitation Network
- DenseNet
- FractalNet
- SqueezeNet
- NASNet

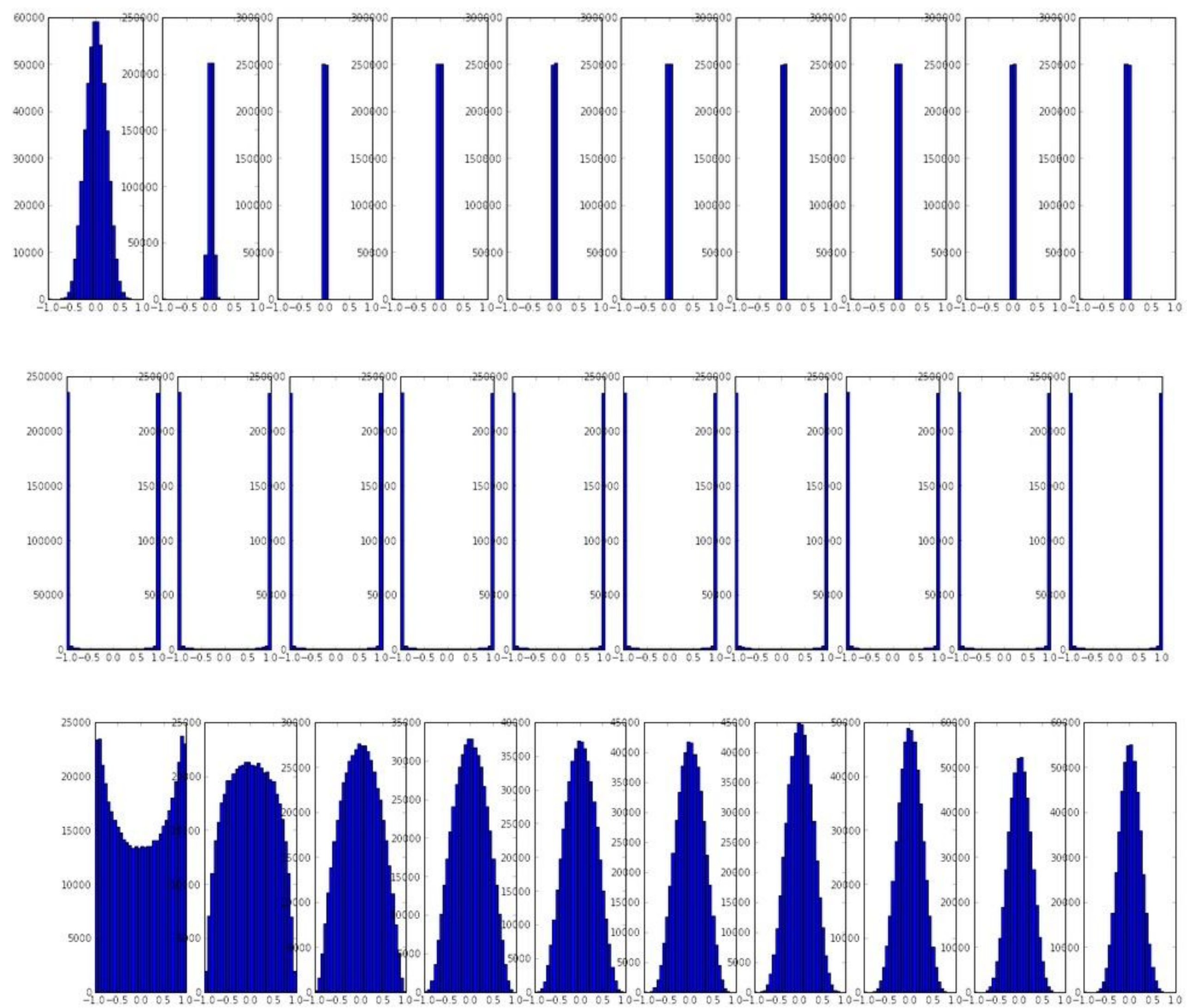
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Summary: CNN Architectures

- VGG, GoogLeNet, ResNet all in wide use, available in model zoos
- ResNet current best default, also consider SENet when available
- Trend towards extremely deep networks
- Significant research centers around design of layer / skip connections and improving gradient flow
- Efforts to investigate necessity of depth vs. width and residual connections
- Even more recent trend towards meta-learning

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Weight Initialization



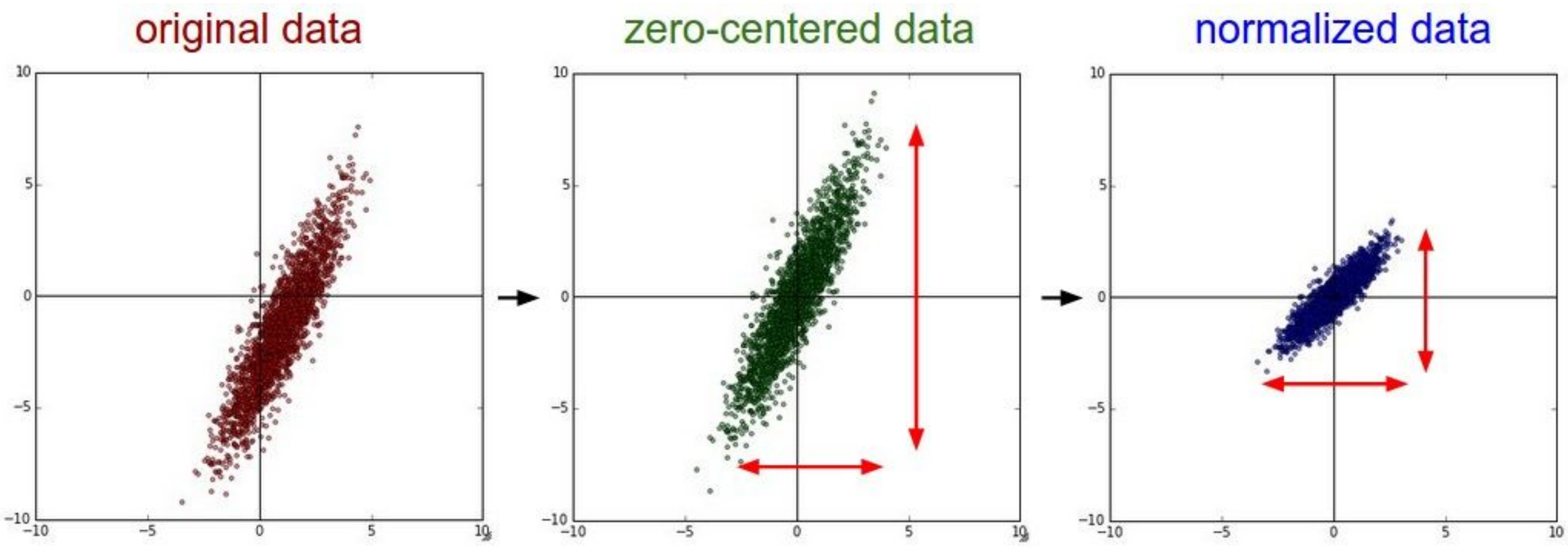
Initialization too small:
Activations go to zero, gradients also zero,
No learning

Initialization too big:
Activations saturate (for tanh),
Gradients zero, no learning

Initialization just right:
Nice distribution of activations at all layers,
Learning proceeds nicely

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

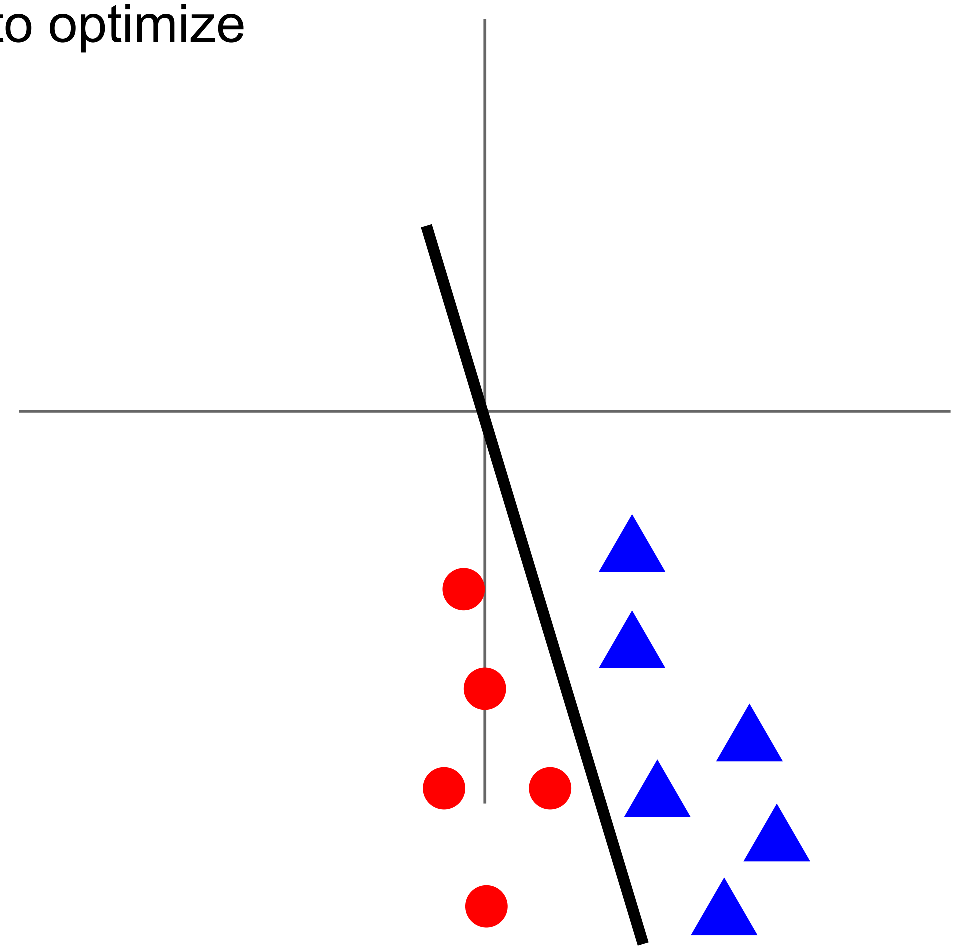
Data Preprocessing



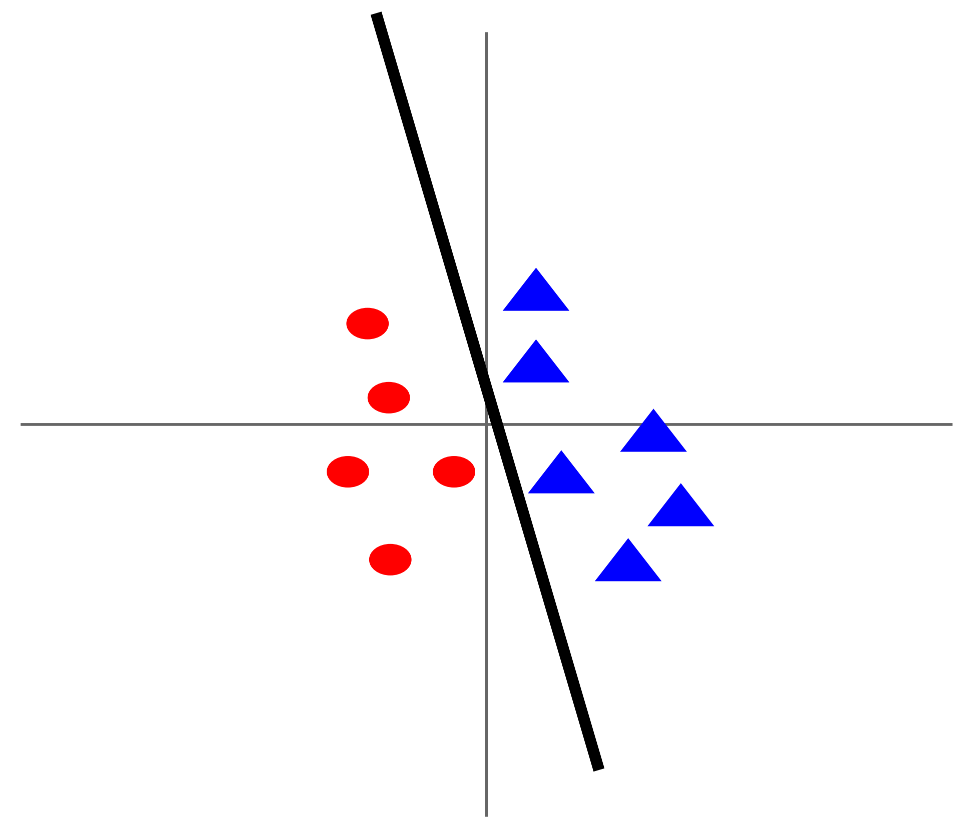
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Data Preprocessing

Before normalization: classification loss very sensitive to changes in weight matrix; hard to optimize

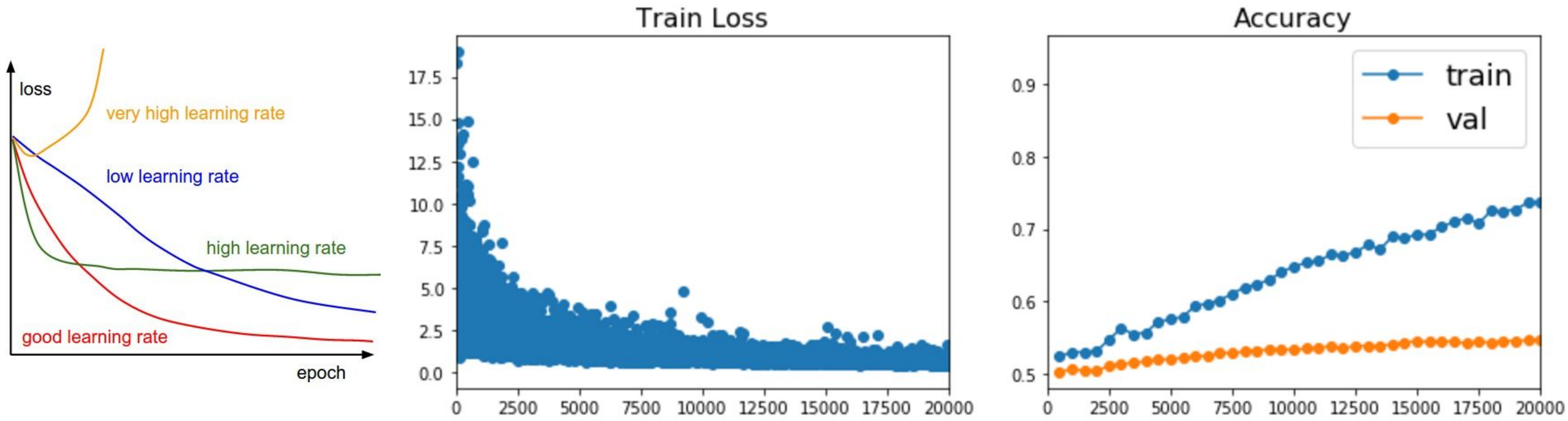


After normalization: less sensitive to small changes in weights; easier to optimize



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

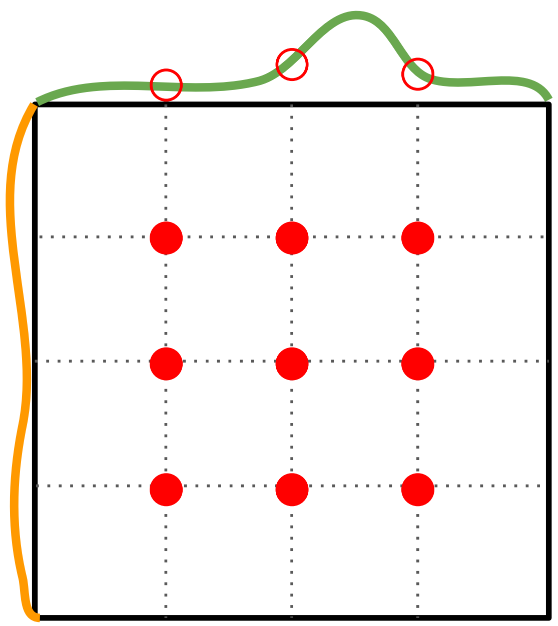
Babysitting Learning



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Hyperparameter Search

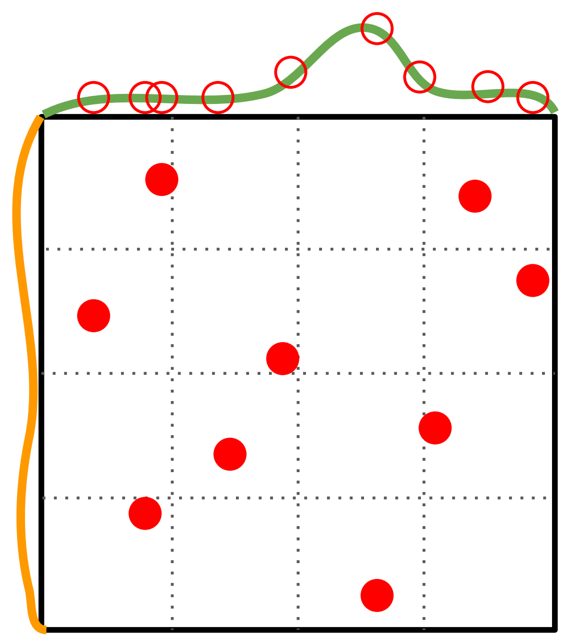
Grid Layout



Unimportant Parameter

Important Parameter

Random Layout



Unimportant Parameter

Important Parameter

Coarse to fine search

```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

- More normalization
- Fancier optimization
- Regularization
- Transfer Learning

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Batch Normalization

Input: $x : N \times D$

Learnable params:

$\gamma, \beta : D$

Intermediates: $\mu, \sigma : D$

$\hat{x} : N \times D$

Output: $y : N \times D$

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Batch Normalization

Estimate mean and variance from minibatch; Can't do this at test-time

Input: $x : N \times D$

Learnable params:

$\gamma, \beta : D$

Intermediates: $\mu, \sigma : D$

$\hat{x} : N \times D$

Output: $y : N \times D$

$$\boxed{\begin{aligned}\mu_j &= \frac{1}{N}\sum_{i=1}^{N} x_{i,j} \\ \sigma_j^2 &= \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2\end{aligned}}$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Batch Normalization: Test Time

Input: $x : N \times D$

Learnable params:

$\gamma, \beta : D$

Intermediates: $\mu, \sigma : D$
$\hat{x} : N \times D$

Output: $y : N \times D$

$\mu_j =$ (Running) average of values seen during training

$\sigma_j^2 =$ (Running) average of values seen during training

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Batch Normalization for ConvNets

Batch Normalization for **fully-connected** networks

$\mathbf{x}: N \times D$

Normalize

$\boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times D$

$\boldsymbol{\gamma}, \boldsymbol{\beta}: 1 \times D$

$\mathbf{y} = \gamma(\mathbf{x}-\boldsymbol{\mu})/\boldsymbol{\sigma}+\beta$

Batch Normalization for **convolutional** networks (Spatial Batchnorm, BatchNorm2D)

$\mathbf{x}: N \times C \times H \times W$

Normalize

$\boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times C \times 1 \times 1$

$\boldsymbol{\gamma}, \boldsymbol{\beta}: 1 \times C \times 1 \times 1$

$\mathbf{y} = \gamma(\mathbf{x}-\boldsymbol{\mu})/\boldsymbol{\sigma}+\beta$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Layer Normalization

Batch Normalization for
fully-connected networks

$$\mathbf{x}: N \times D$$

Normalize

$$\boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times D$$

$$\gamma, \beta: 1 \times D$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

Layer Normalization for
fully-connected networks
Same behavior at train and test!
Can be used in recurrent networks

$$\mathbf{x}: N \times D$$

Normalize

$$\boldsymbol{\mu}, \boldsymbol{\sigma}: N \times 1$$

$$\gamma, \beta: 1 \times D$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

Ba, Kiros, and Hinton, "Layer Normalization", arXiv 2016

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Instance Normalization

Batch Normalization for convolutional networks

$$\mathbf{x}: \; N \times C \times H \times W$$

Normalize

$$\boldsymbol{\mu}, \boldsymbol{\sigma}: \; 1 \times C \times 1 \times 1$$

$$\gamma, \beta: \; 1 \times C \times 1 \times 1$$

$$y = \gamma(x - \mu)/\sigma + \beta$$

Instance Normalization for convolutional networks
Same behavior at train / test!

$$\mathbf{x}: \; N \times C \times H \times W$$

Normalize

$$\boldsymbol{\mu}, \boldsymbol{\sigma}: \; N \times C \times 1 \times 1$$

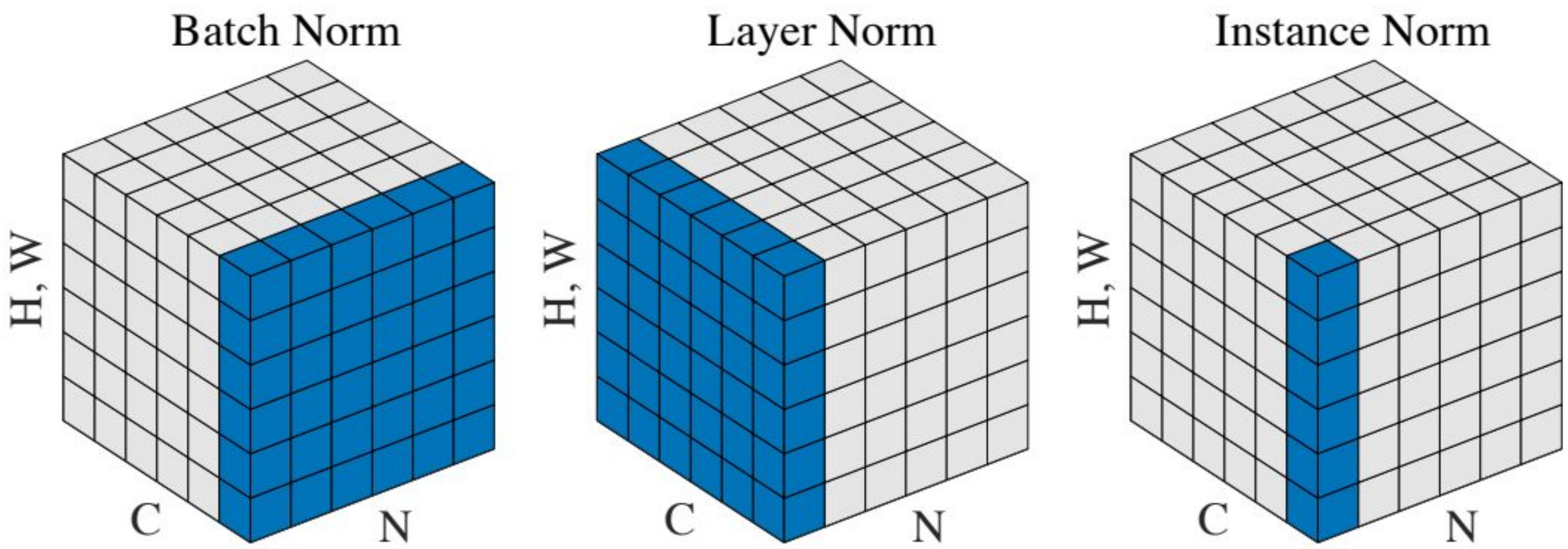
$$\gamma, \beta: \; 1 \times C \times 1 \times 1$$

$$y = \gamma(x - \mu)/\sigma + \beta$$

Ulyanov et al, Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, CVPR 2017

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

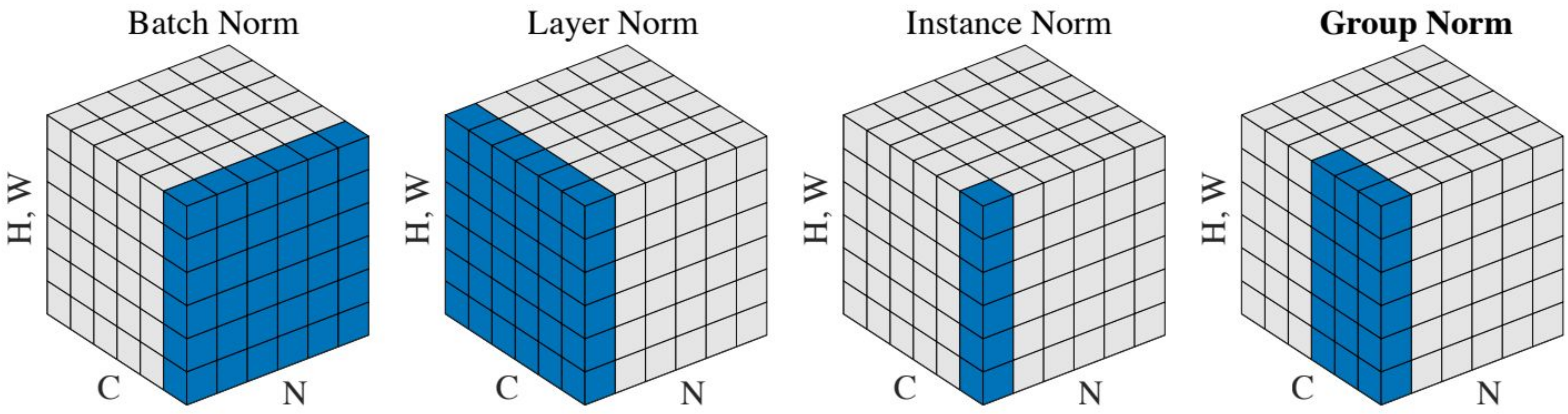
Comparison of Normalization Layers



Wu and He, “Group Normalization”, arXiv 2018

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Group Normalization

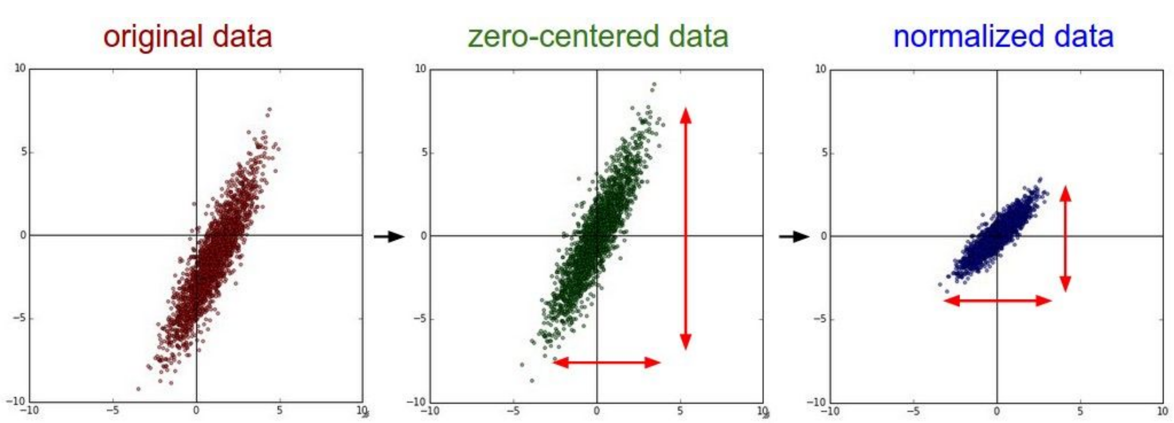


Wu and He, "Group Normalization", arXiv 2018 (Appeared 3/22/2018)

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Decorrelated Batch Normalization

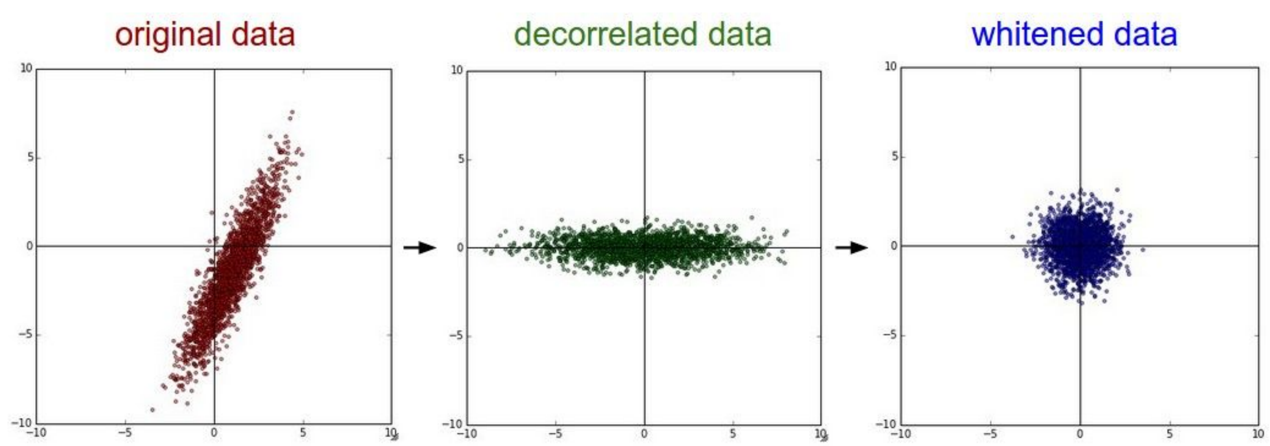
Batch Normalization



$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

BatchNorm normalizes the data, but cannot correct for correlations among the input features

Decorrelated Batch Normalization



$$\hat{x}_i = \Sigma^{-\frac{1}{2}}(x_i - \mu)$$

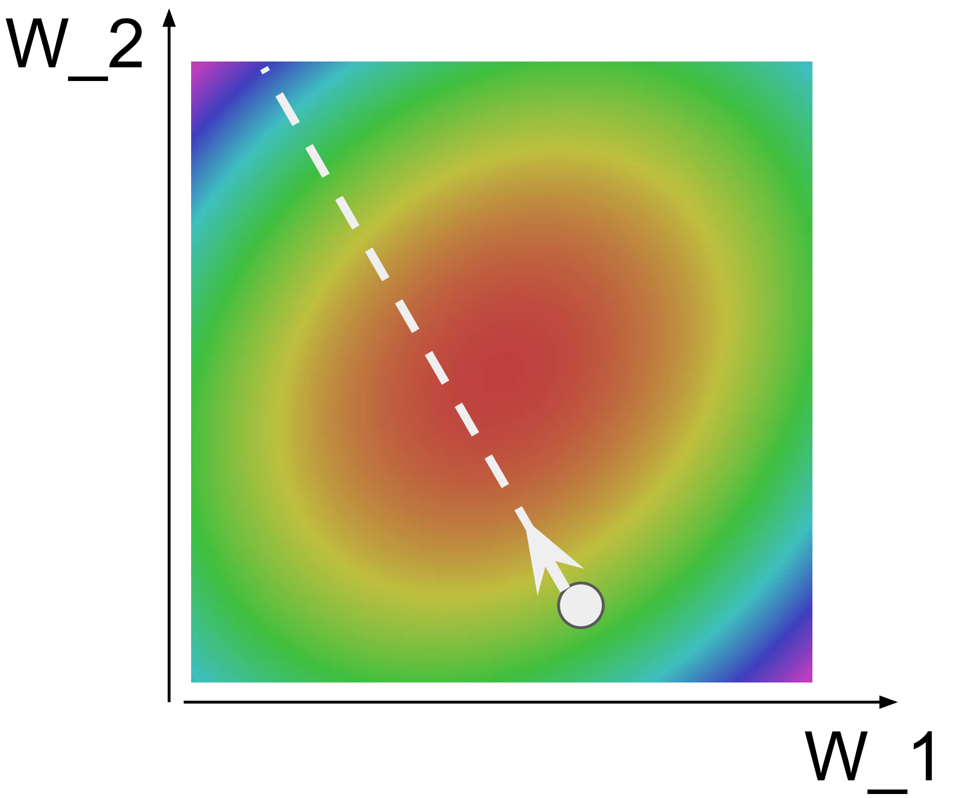
DBN **whitens** the data using the full covariance matrix of the minibatch; this corrects for correlations

Huang et al, “Decorrelated Batch Normalization”, arXiv 2018 (Appeared 4/23/2018)

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Optimization

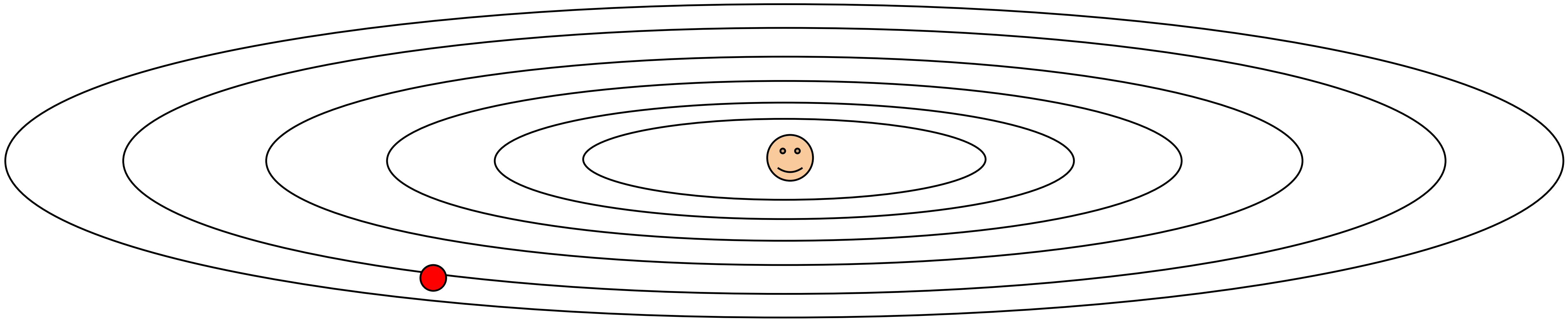
```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Optimization: Problems with SGD

What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?

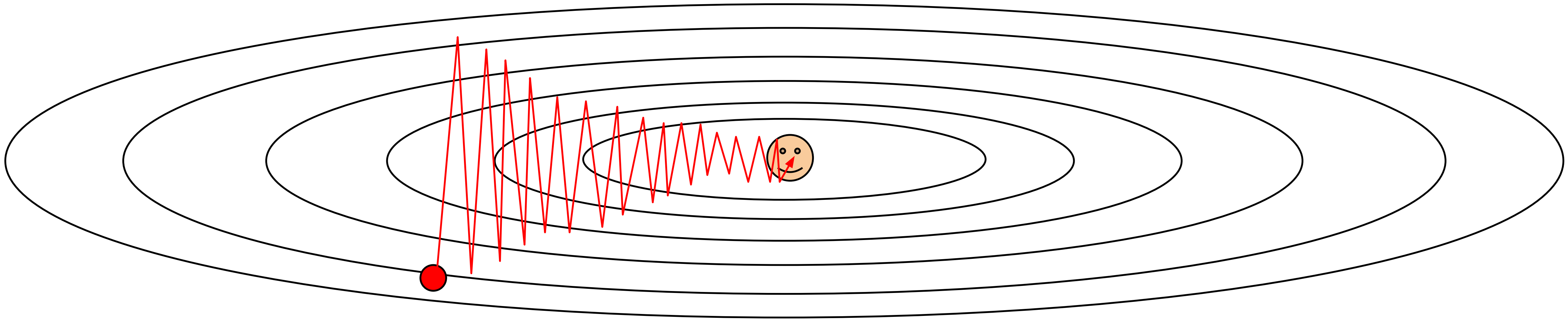


Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Optimization: Problems with SGD

What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?
Very slow progress along shallow dimension, jitter along steep direction

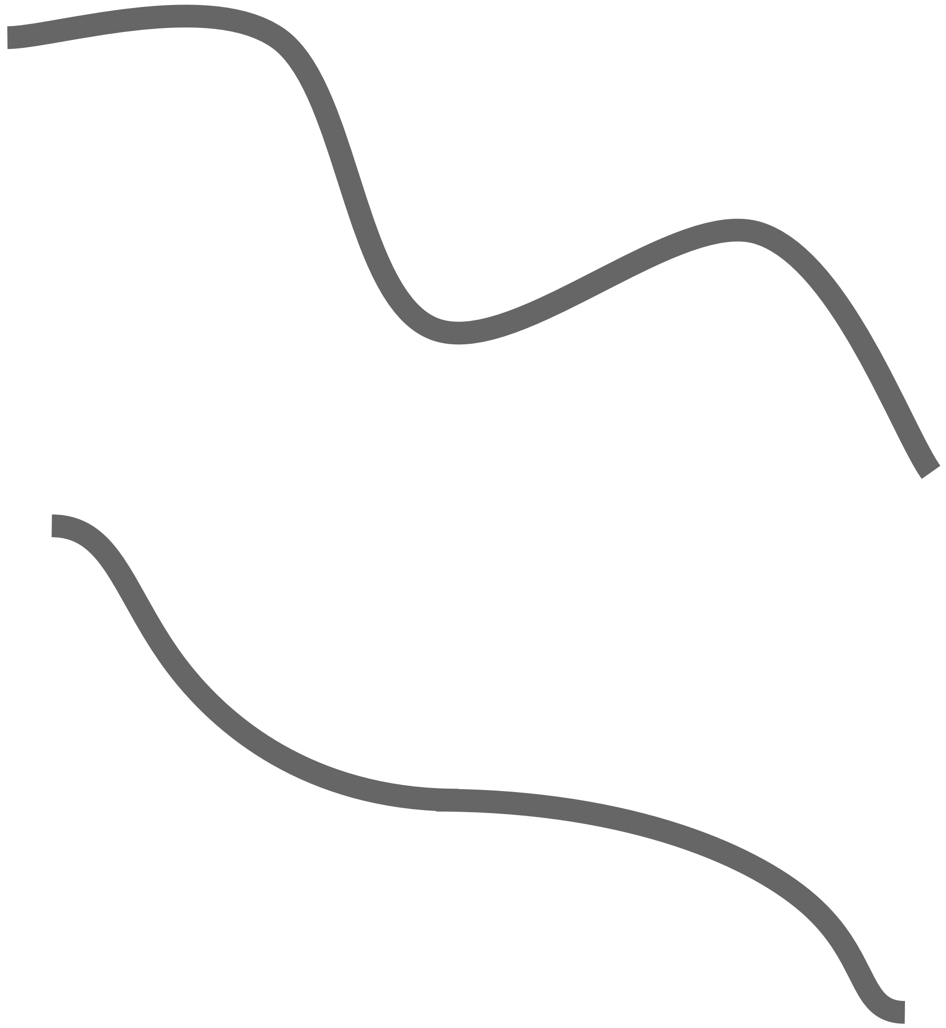


Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Optimization: Problems with SGD

What if the loss function has a **local minima** or **saddle point**?

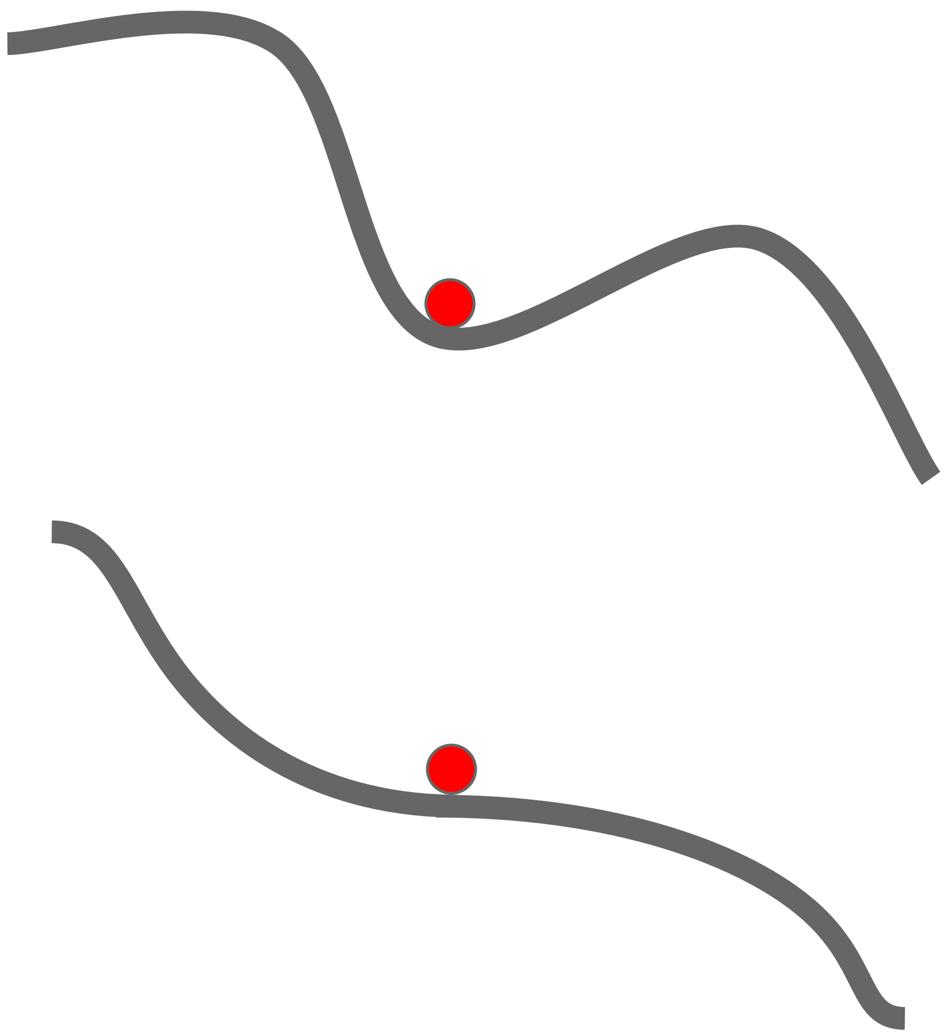


slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Optimization: Problems with SGD

What if the loss function has a **local minima** or **saddle point**?

Zero gradient, gradient descent gets stuck

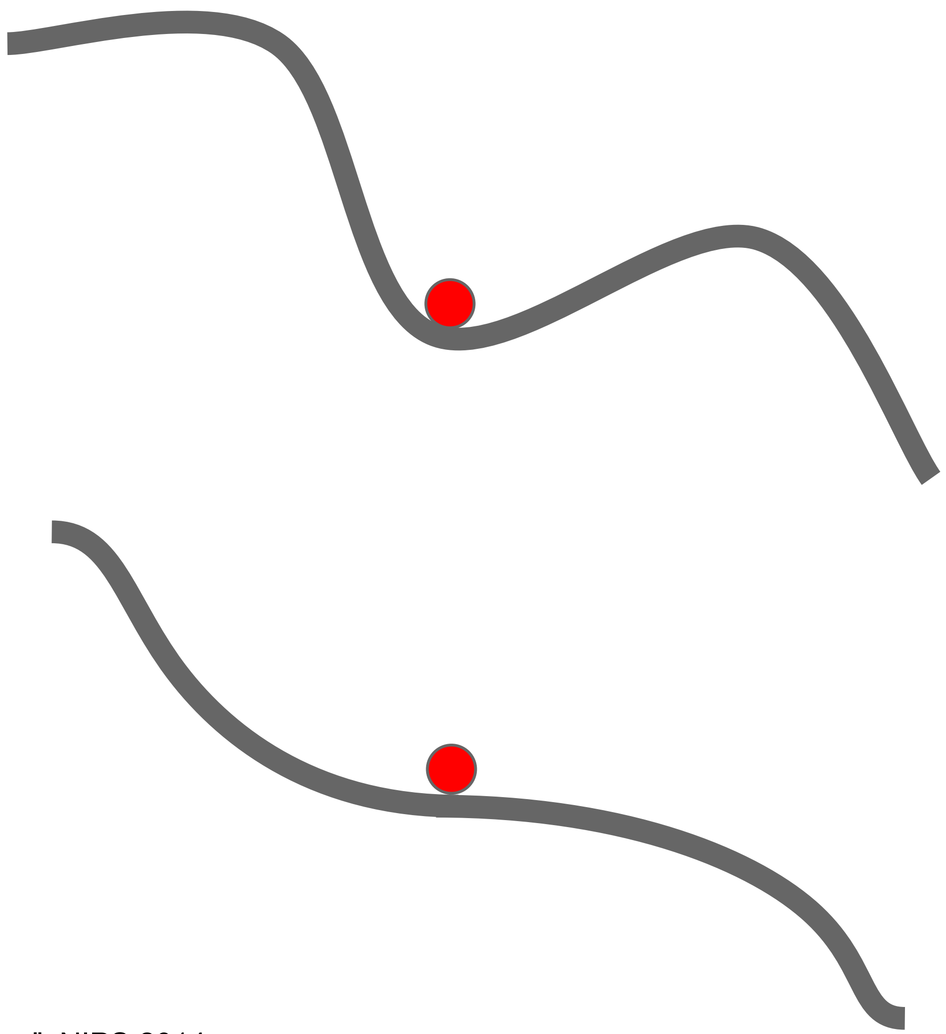


slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Optimization: Problems with SGD

What if the loss function has a **local minima** or **saddle point**?

Saddle points much more common in high dimension



Dauphin et al, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization", NIPS 2014

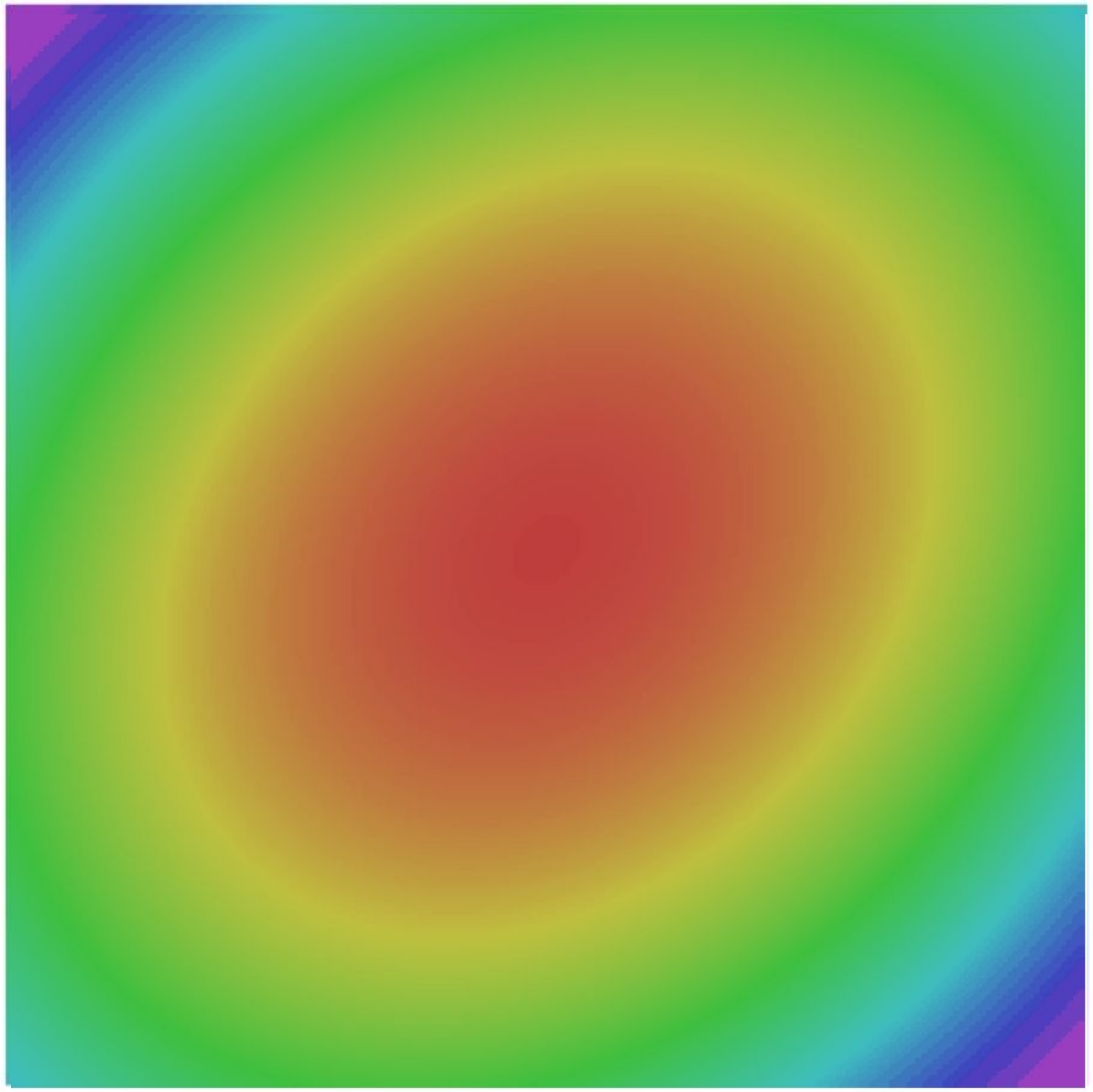
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Optimization: Problems with SGD

Our gradients come from minibatches so they can be noisy!

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

SGD + Momentum

SGD+Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx - learning_rate * dx
  x += vx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

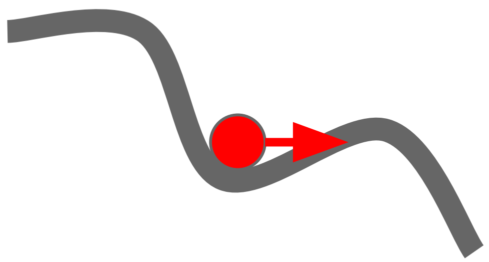
You may see SGD+Momentum formulated different ways, but they are equivalent - give same sequence of x

Sutskever et al, "On the importance of initialization and momentum in deep learning", ICML 2013

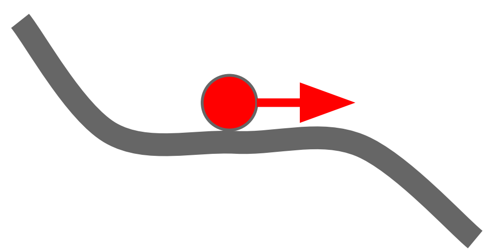
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

SGD + Momentum

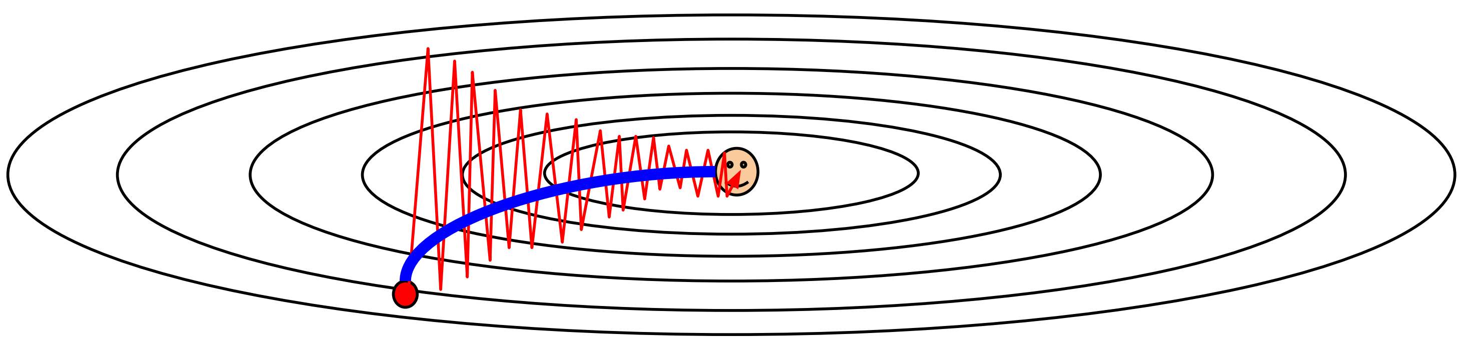
Local Minima



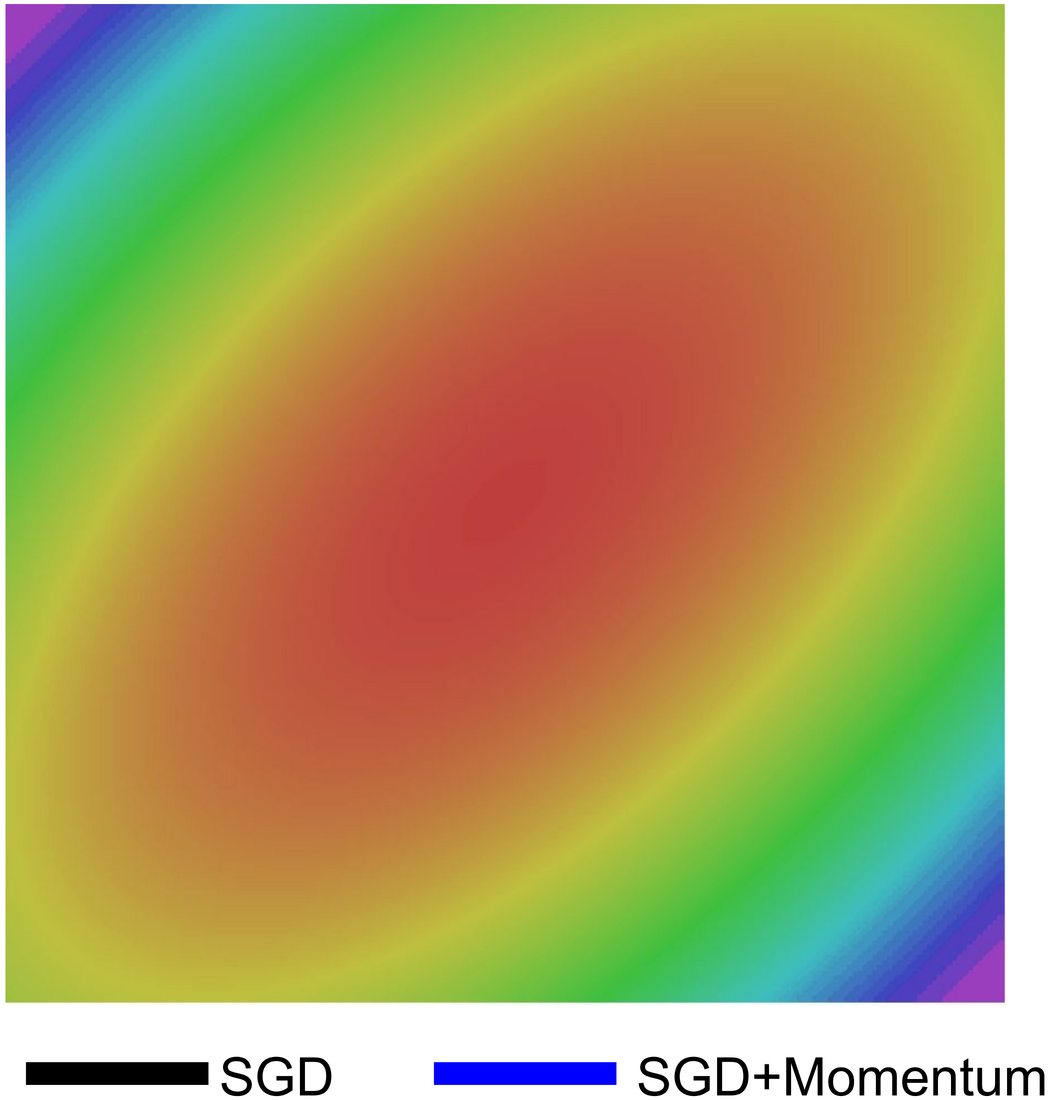
Saddle points



Poor Conditioning



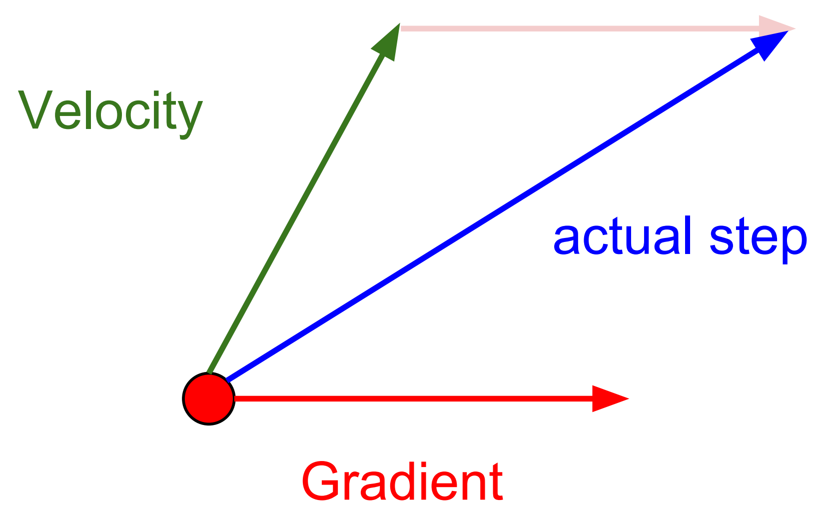
Gradient Noise



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

SGD+Momentum

Momentum update:



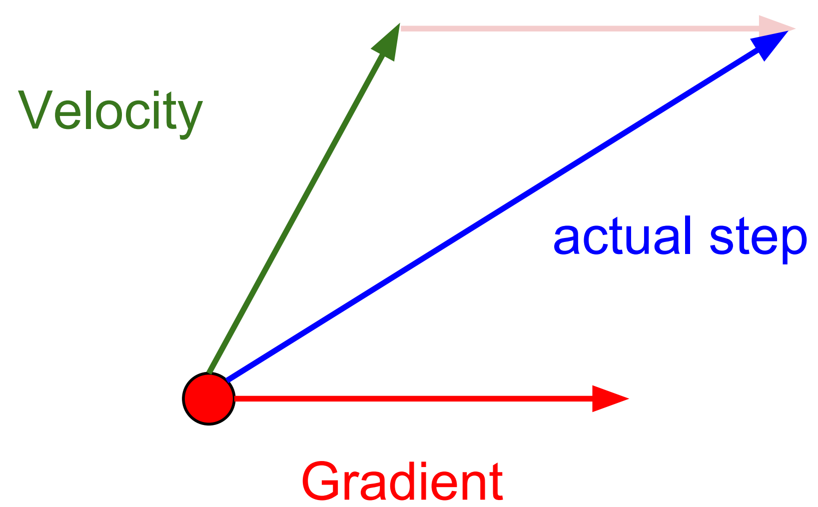
Combine gradient at current point with velocity to get step used to update weights

Nesterov, “A method of solving a convex programming problem with convergence rate O(1/k^2)”, 1983
Nesterov, “Introductory lectures on convex optimization: a basic course”, 2004
Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Nesterov Momentum

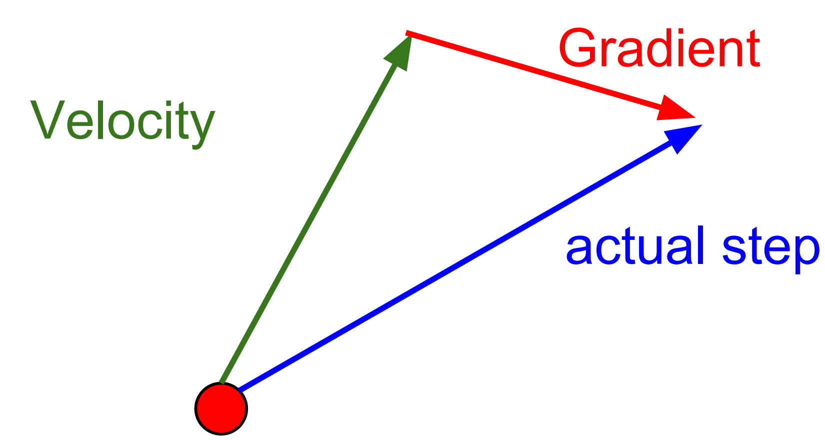
Momentum update:



Combine gradient at current point with velocity to get step used to update weights

Nesterov, "A method of solving a convex programming problem with convergence rate O(1/k^2)", 1983
Nesterov, "Introductory lectures on convex optimization: a basic course", 2004
Sutskever et al, "On the importance of initialization and momentum in deep learning", ICML 2013

Nesterov Momentum

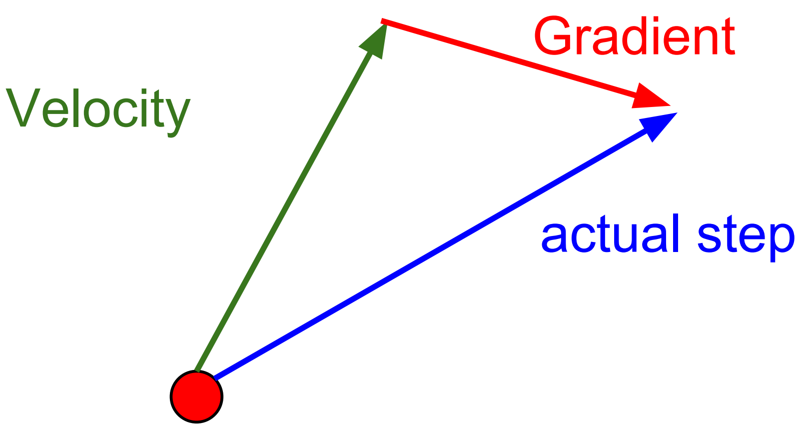


"Look ahead" to the point where updating using velocity would take us; compute gradient there and mix it with velocity to get actual update direction

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$
$$x_{t+1} = x_t + v_{t+1}$$



“Look ahead” to the point where updating using velocity would take us; compute gradient there and mix it with velocity to get actual update direction

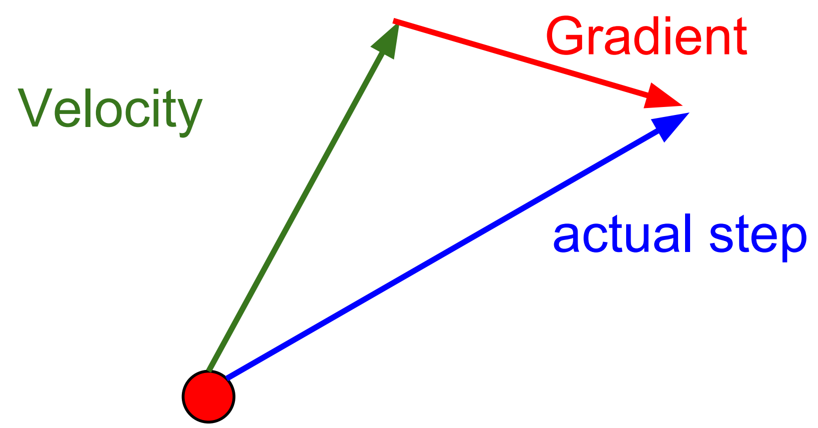
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(\boxed{x_t + \rho v_t})$$

$$x_{t+1} = x_t + v_{t+1}$$

Annoying, usually we want update in terms of $x_t, \nabla f(x_t)$



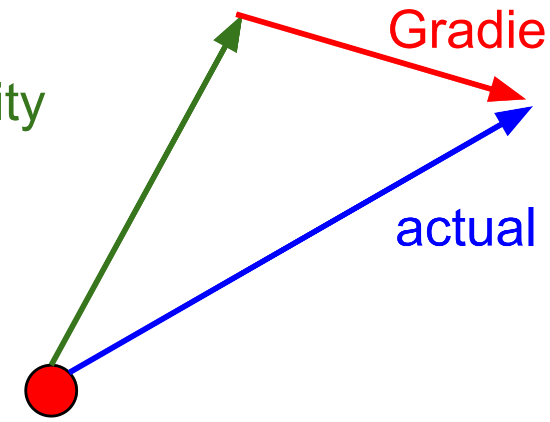
"Look ahead" to the point where updating using velocity would take us; compute gradient there and mix it with velocity to get actual update direction

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(\boxed{x_t + \rho v_t})$$
$$x_{t+1} = x_t + v_{t+1}$$

Annoying, usually we want update in terms of $x_t, \nabla f(x_t)$



Change of variables $\tilde{x}_t = x_t + \rho v_t$ and rearrange:

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$
$$\begin{aligned} \tilde{x}_{t+1} &= \tilde{x}_t - \rho v_t + (1 + \rho) v_{t+1} \\ &= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t) \end{aligned}$$

"Look ahead" to the point where updating using velocity would take us; compute gradient there and mix it with velocity to get actual update direction

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(\boxed{x_t + \rho v_t})$$
$$x_{t+1} = x_t + v_{t+1}$$

Annoying, usually we want update in terms of $x_t, \nabla f(x_t)$

Change of variables $\tilde{x}_t = x_t + \rho v_t$ and rearrange:

$$\begin{aligned} v_{t+1} &= \rho v_t - \alpha \nabla f(\tilde{x}_t) \\ \tilde{x}_{t+1} &= \tilde{x}_t - \rho v_t + (1 + \rho) v_{t+1} \\ &= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t) \end{aligned}$$

```
dx = compute_gradient(x)
old_v = v
v = rho * v - learning_rate * dx
x += -rho * old_v + (1 + rho) * v
```

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Nesterov Momentum

SGD

SGD+Momentum

Nesterov

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Added element-wise scaling of the gradient based on the historical sum of squares in each dimension

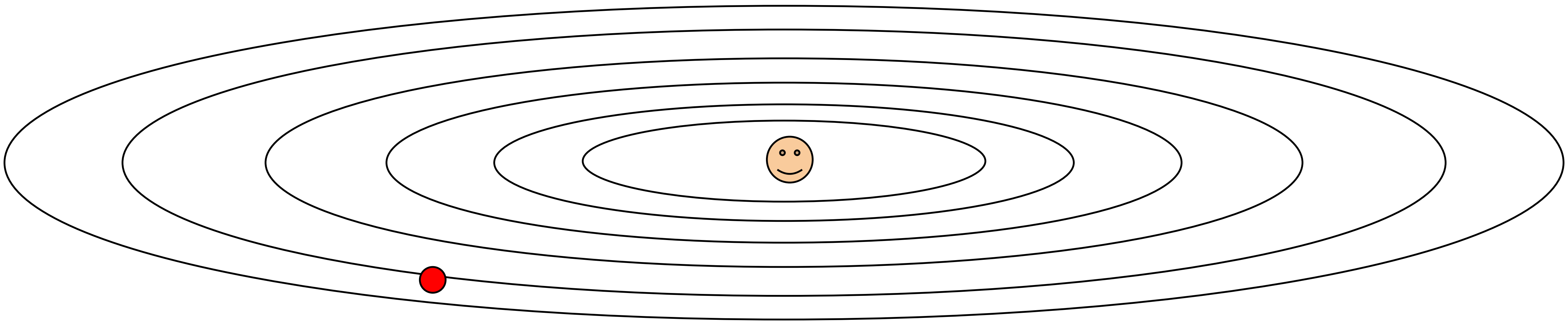
"Per-parameter learning rates"
or "adaptive learning rates"

Duchi et al, "Adaptive subgradient methods for online learning and stochastic optimization", JMLR 2011

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

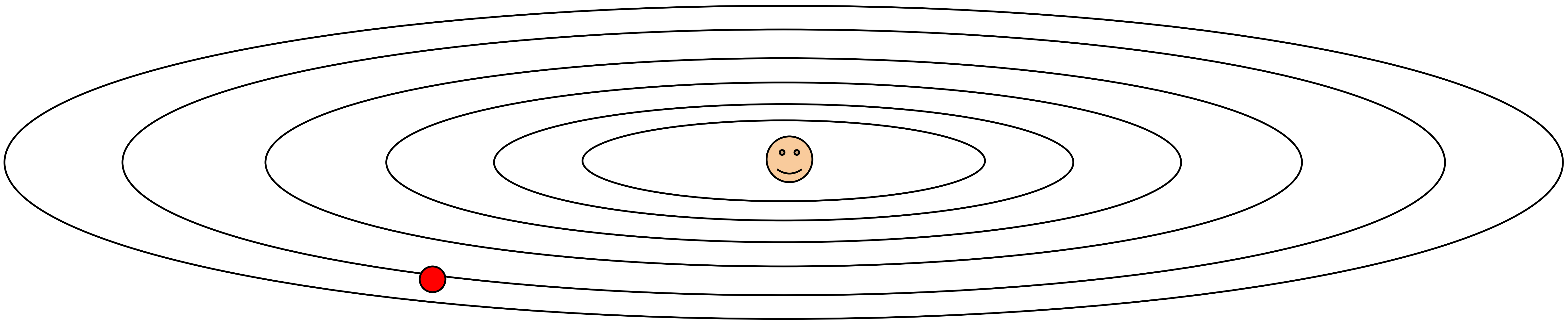


Q: What happens with AdaGrad?

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



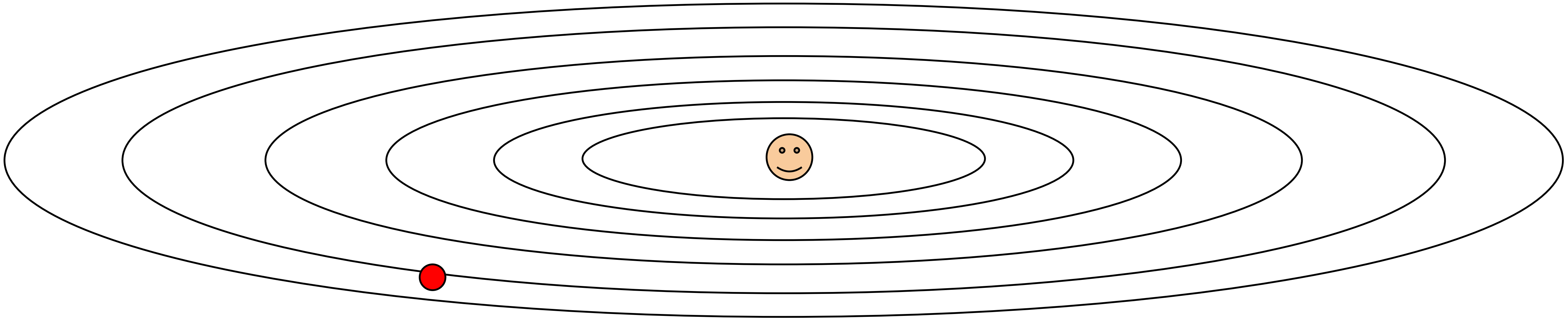
Q: What happens with AdaGrad?

Progress along "steep" directions is damped; progress along "flat" directions is accelerated

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

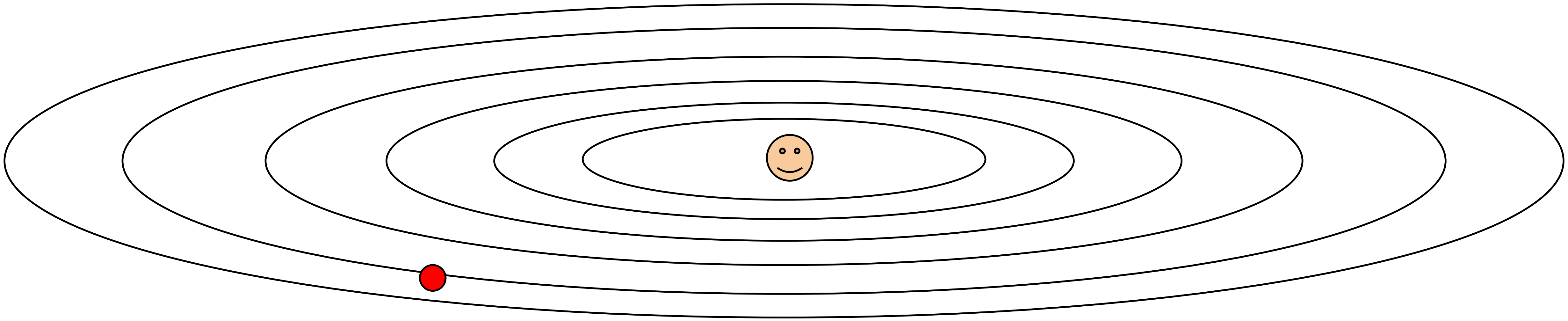


Q2: What happens to the step size over long time?

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



Q2: What happens to the step size over long time? Decays to zero

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

RMSProp

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

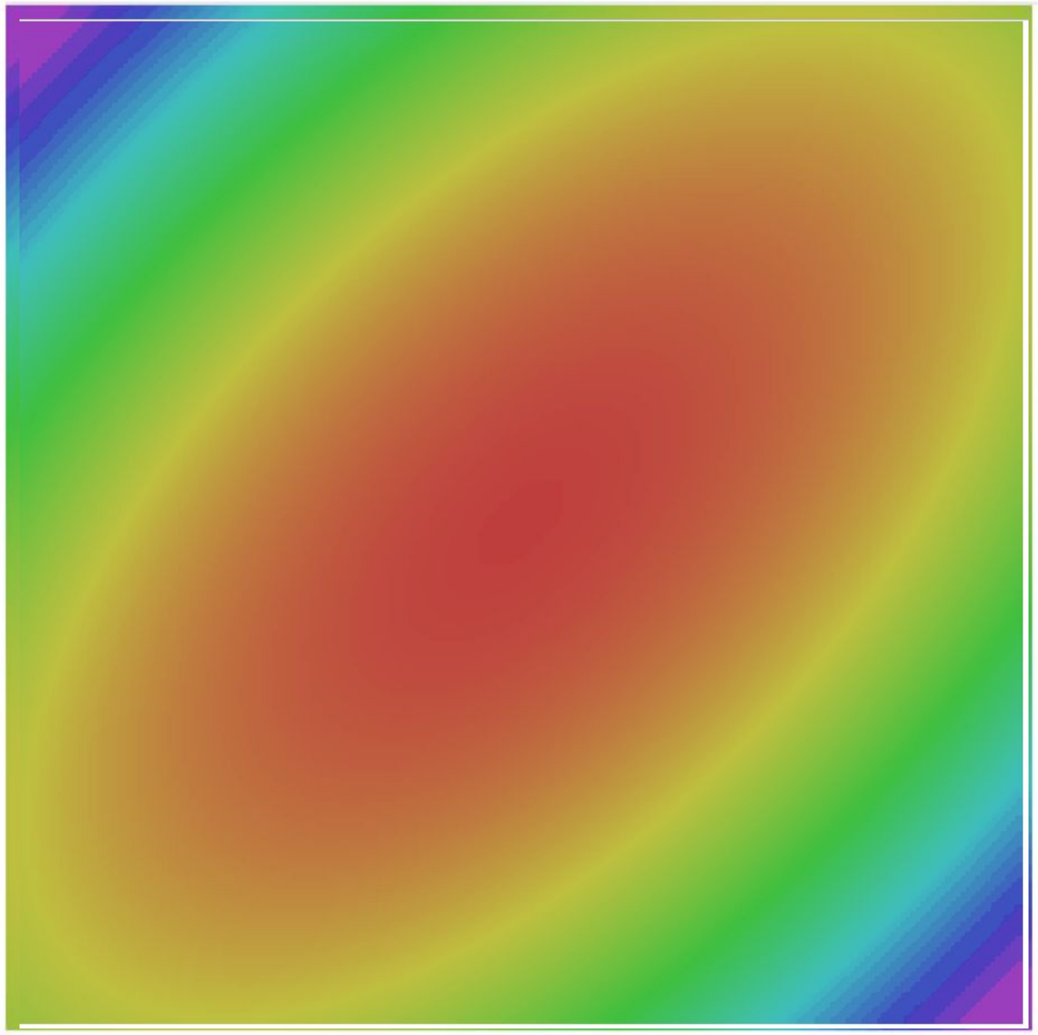
RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Tieleman and Hinton, 2012

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

RMSProp



SGD

SGD+Momentum

RMSProp

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Adam (almost)

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Adam (almost)

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Momentum

AdaGrad / RMSProp

Sort of like RMSProp with momentum

Q: What happens at first timestep?

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Adam (full form)

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

Bias correction for the fact that
first and second moment
estimates start at zero

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Adam (full form)

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

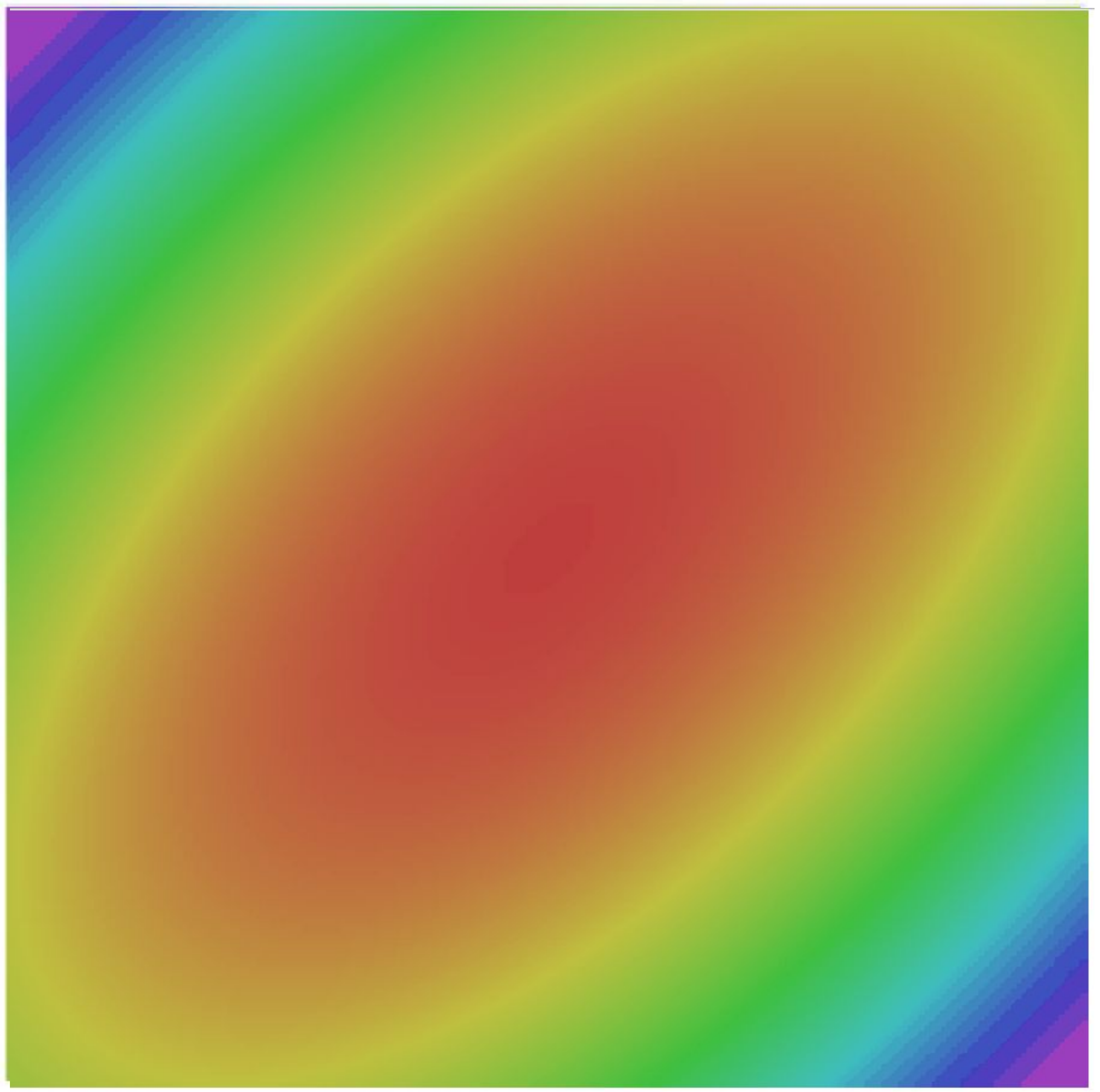
Bias correction for the fact that first and second moment estimates start at zero

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Adam



SGD

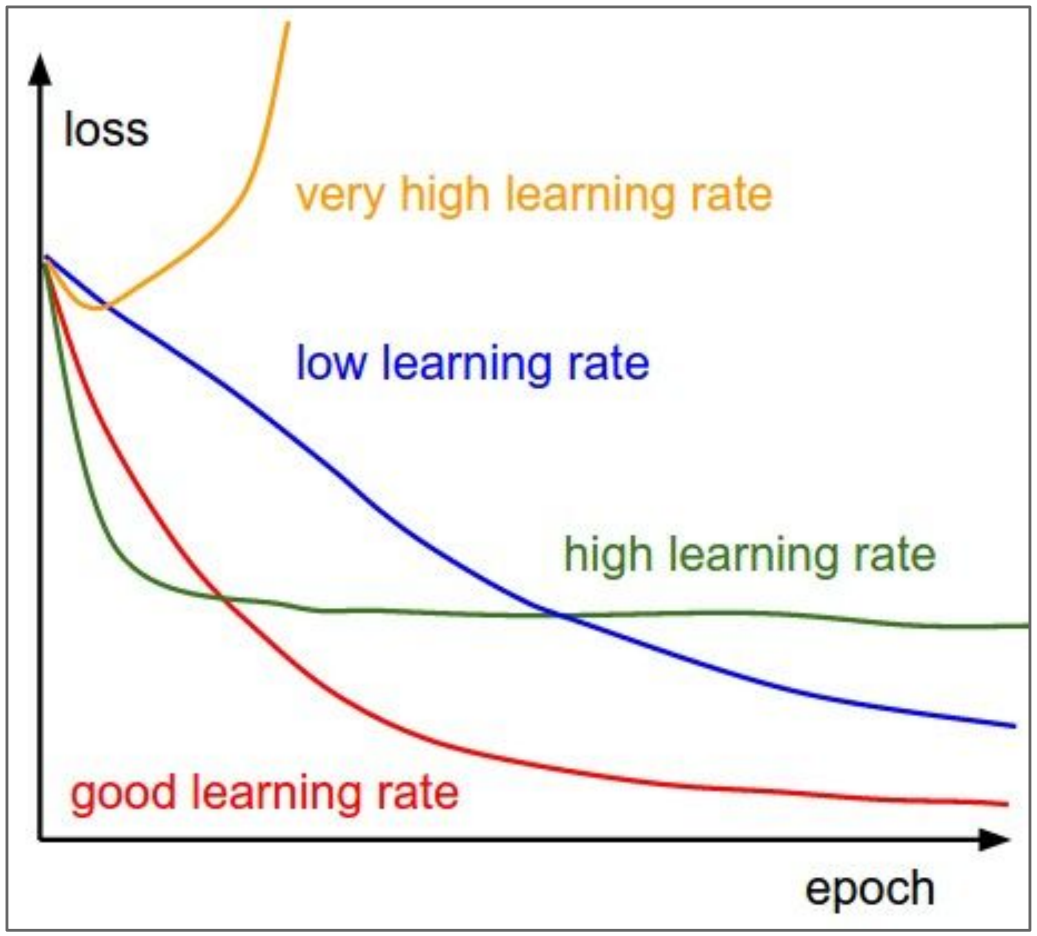
SGD+Momentum

RMSProp

Adam

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

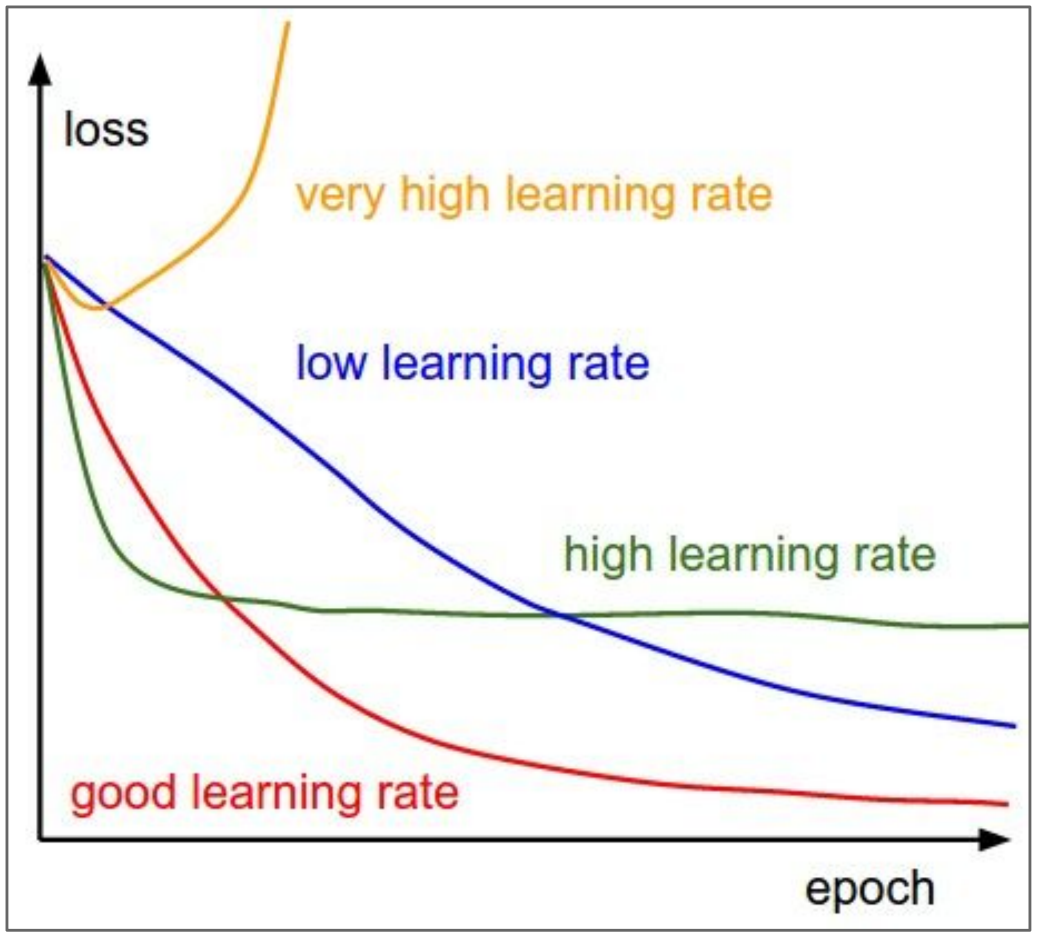
SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



Q: Which one of these learning rates is best to use?

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



=> Learning rate decay over time!

step decay:
e.g. decay learning rate by half every few epochs.

exponential decay:

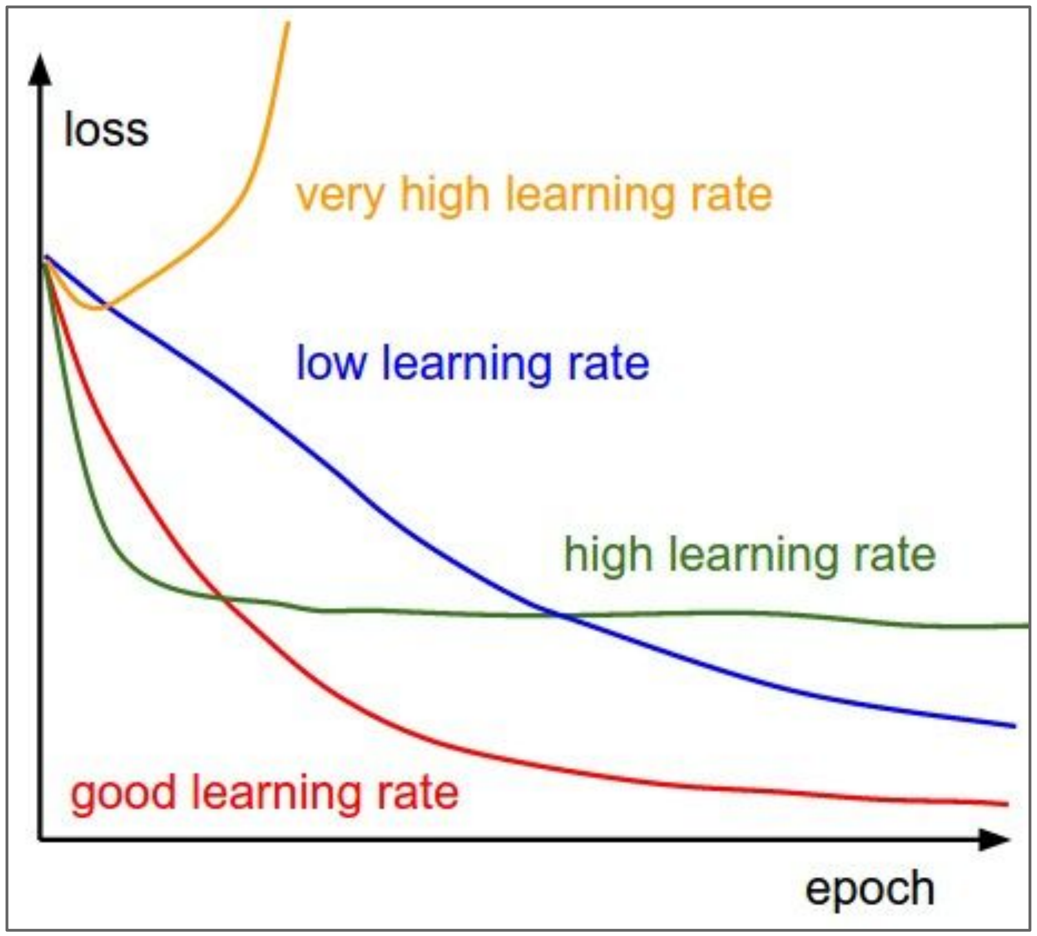
$$\alpha = \alpha_0 e^{-kt}$$

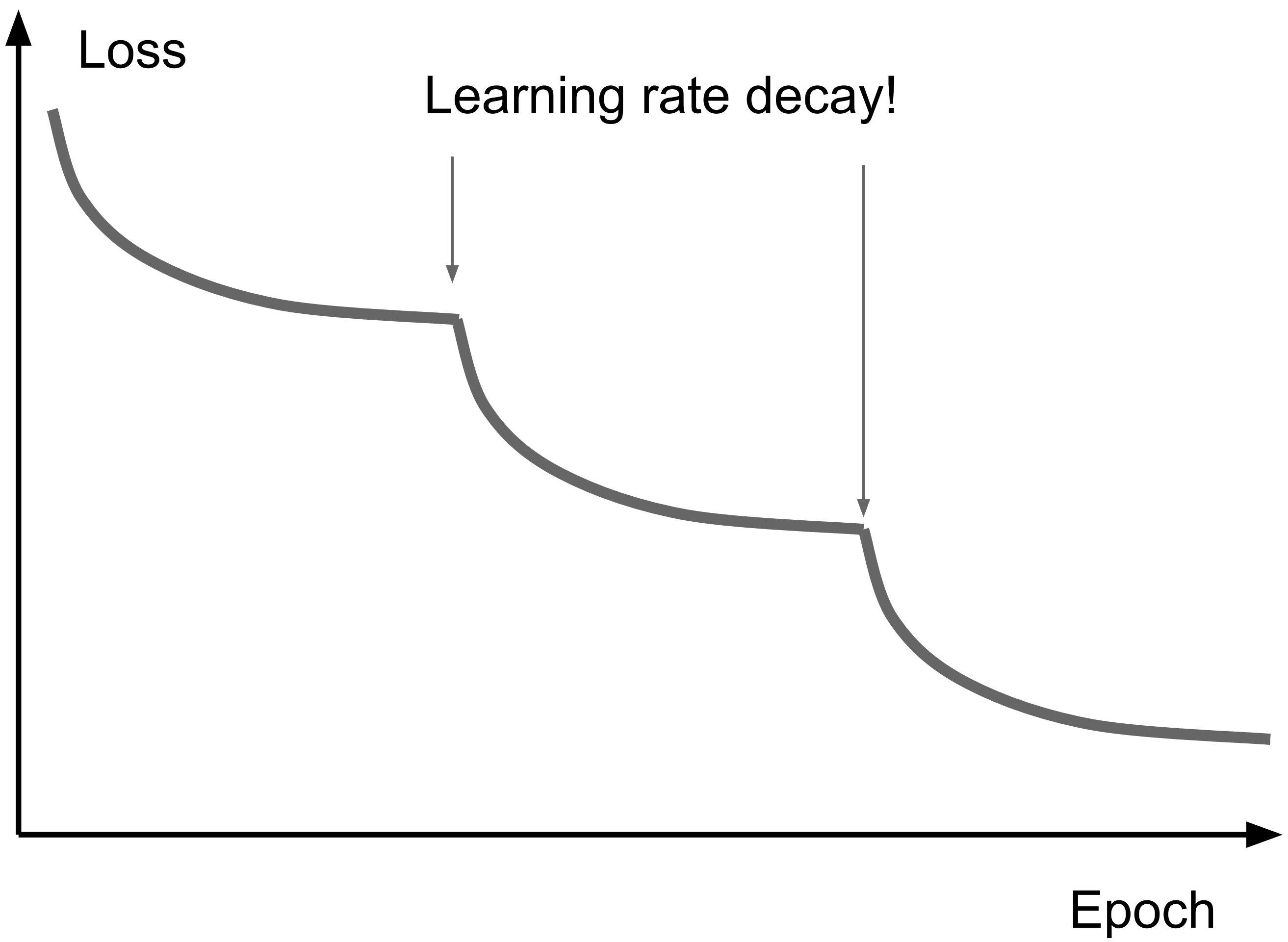
1/t decay:

$$\alpha = \alpha_0 / (1 + kt)$$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

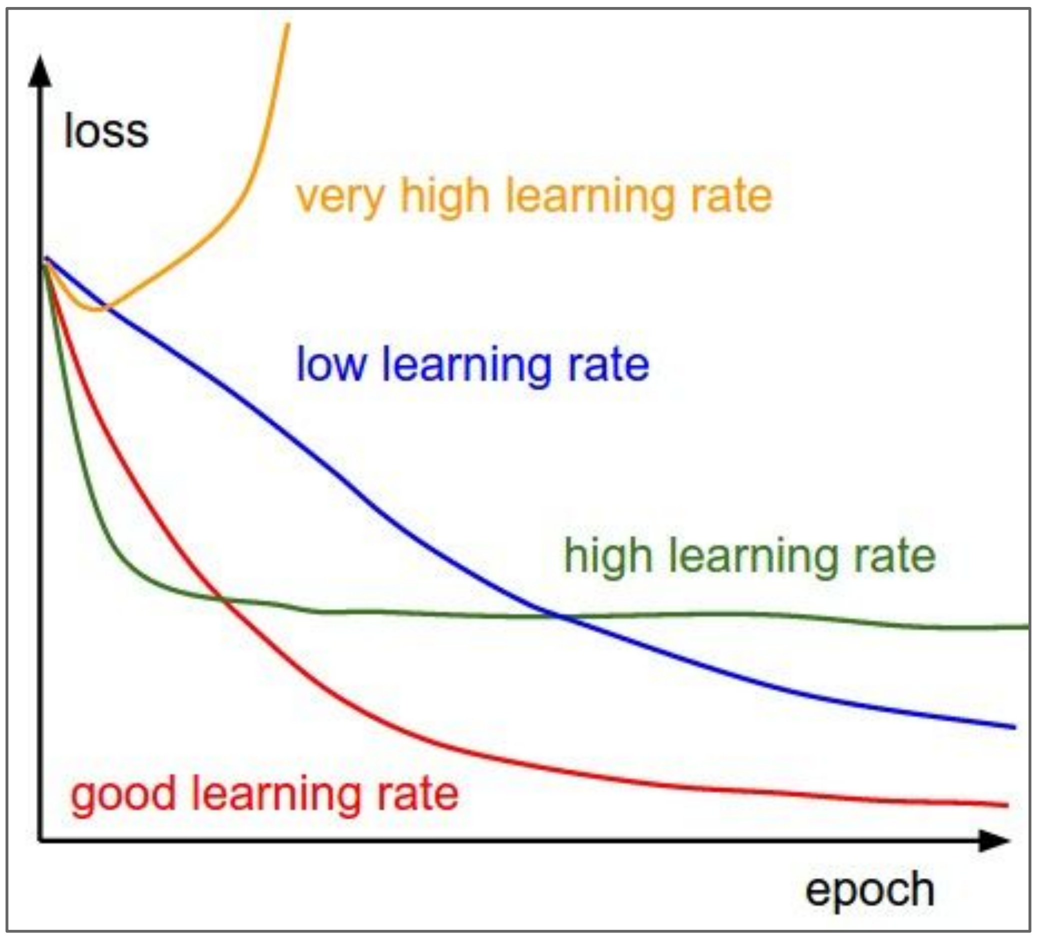
SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.

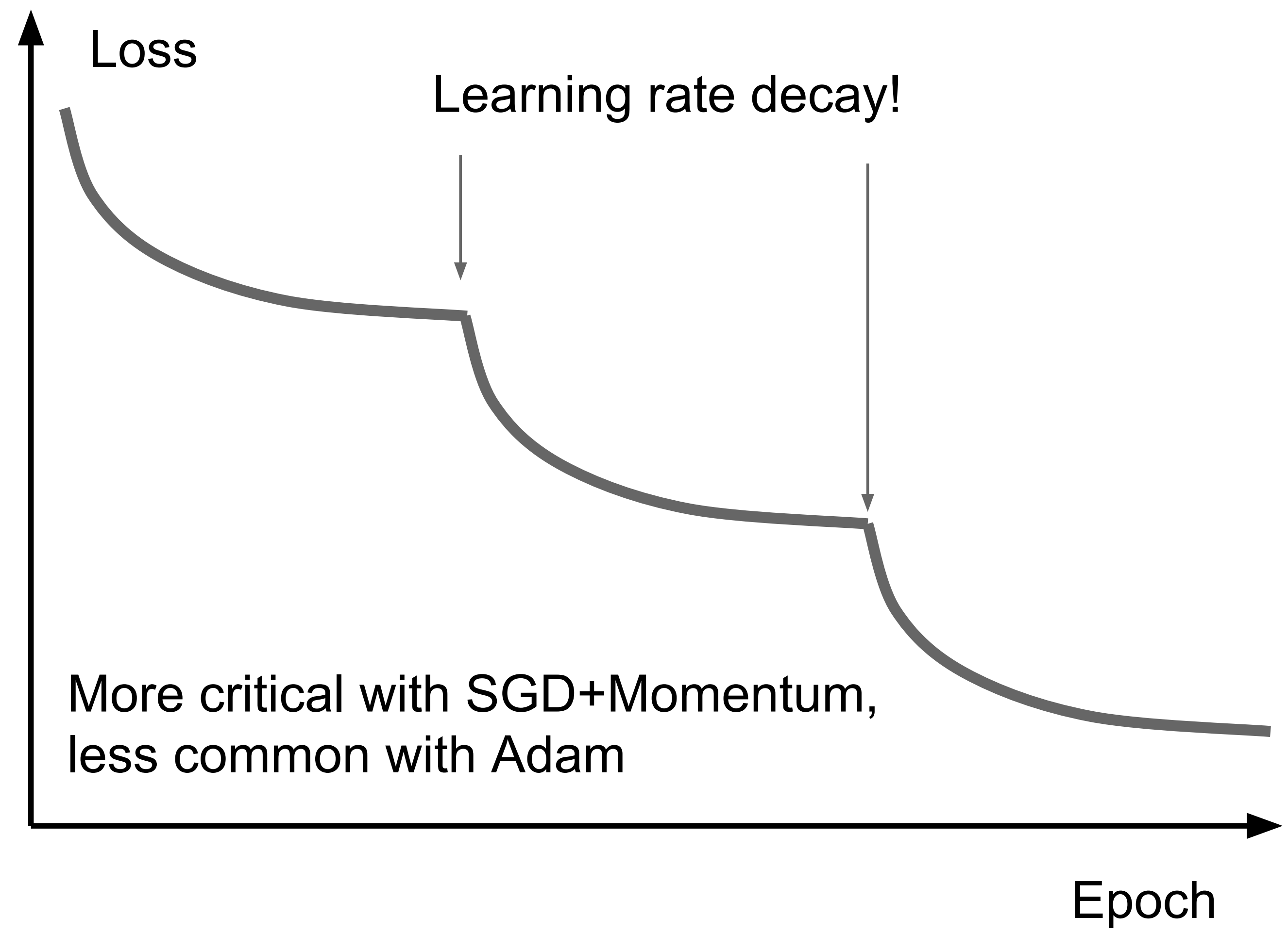




slide credit: Fei-Fei, Justin Johnson, Serena Yeung

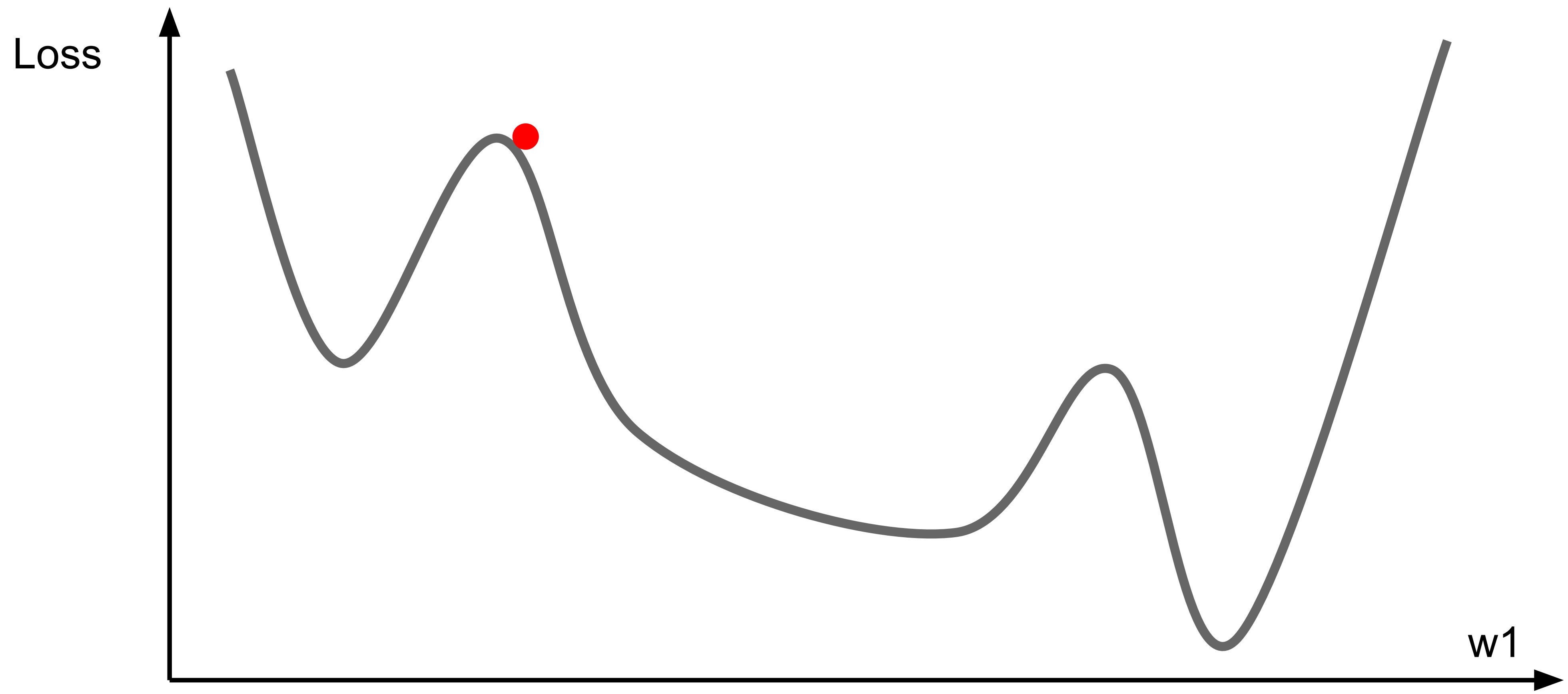
SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.





slide credit: Fei-Fei, Justin Johnson, Serena Yeung

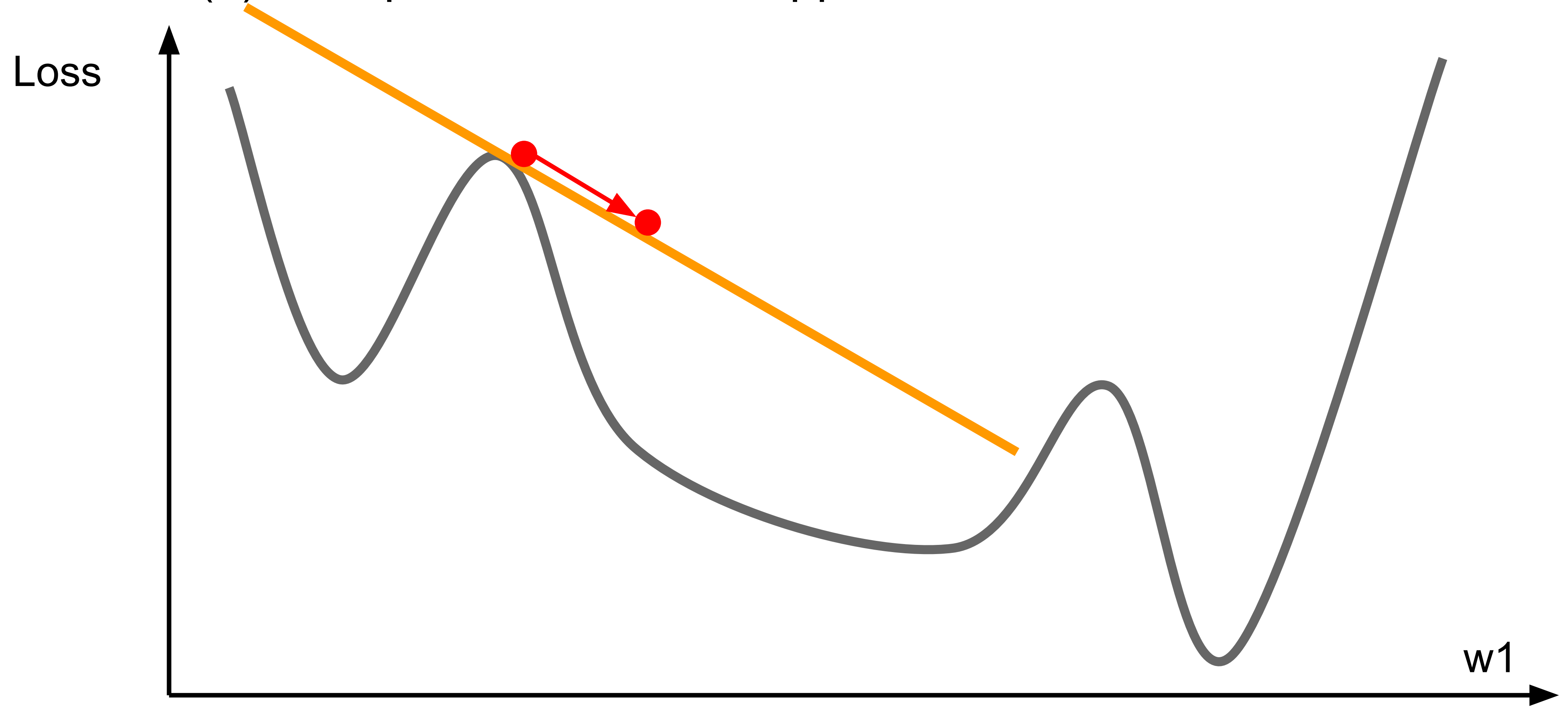
First-Order Optimization



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

First-Order Optimization

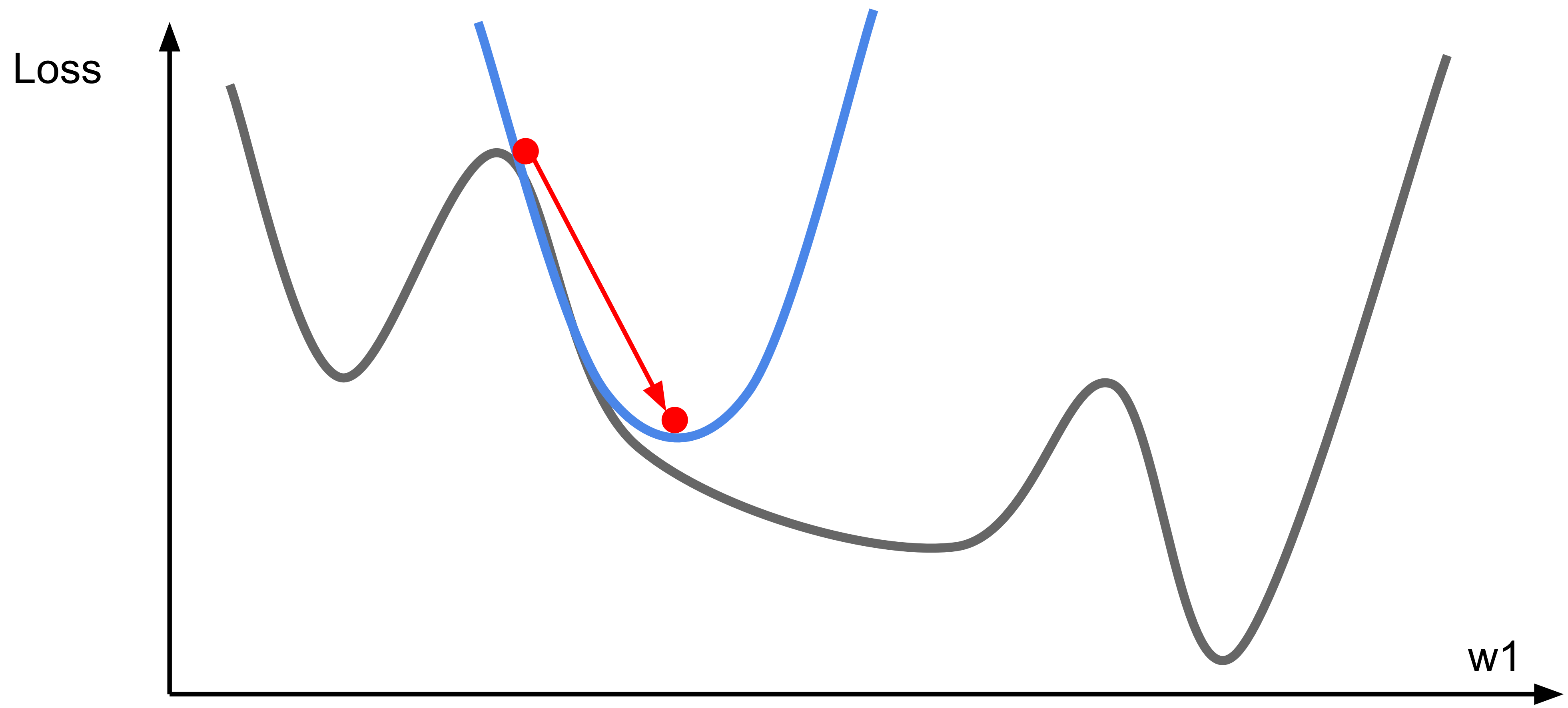
(1) Use gradient form linear approximation
(2) Step to minimize the approximation



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Second-Order Optimization

(1) Use gradient **and Hessian** to form **quadratic** approximation
(2) Step to the **minima** of the approximation



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Second-Order Optimization

second-order Taylor expansion:

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

Solving for the critical point we obtain the Newton parameter update:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

Q: What is nice about this update?

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Second-Order Optimization

second-order Taylor expansion:

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

Solving for the critical point we obtain the Newton parameter update:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

No hyperparameters!
No learning rate!
(Though you might use one in practice)

Q: What is nice about this update?

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Second-Order Optimization

second-order Taylor expansion:

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

Solving for the critical point we obtain the Newton parameter update:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

Q2: Why is this bad for deep learning?

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Second-Order Optimization

second-order Taylor expansion:

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

Solving for the critical point we obtain the Newton parameter update:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

Hessian has O(N^2) elements
Inverting takes O(N^3)
N = (Tens or Hundreds of) Millions

Q2: Why is this bad for deep learning?

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Second-Order Optimization

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

- Quasi-Newton methods (**BGFS** most popular):
  instead of inverting the Hessian (O(n^3)), approximate inverse Hessian with rank 1 updates over time (O(n^2) each).

- **L-BFGS** (Limited memory BFGS):
  Does not form/store the full inverse Hessian.

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

L-BFGS

- **Usually works very well in full batch, deterministic mode** i.e. if you have a single, deterministic f(x) then L-BFGS will probably work very nicely

- **Does not transfer very well to mini-batch setting**. Gives bad results. Adapting second-order methods to large-scale, stochastic setting is an active area of research.

Le et al, “On optimization methods for deep learning, ICML 2011”
Ba et al, “Distributed second-order optimization using Kronecker-factored approximations”, ICLR 2017

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

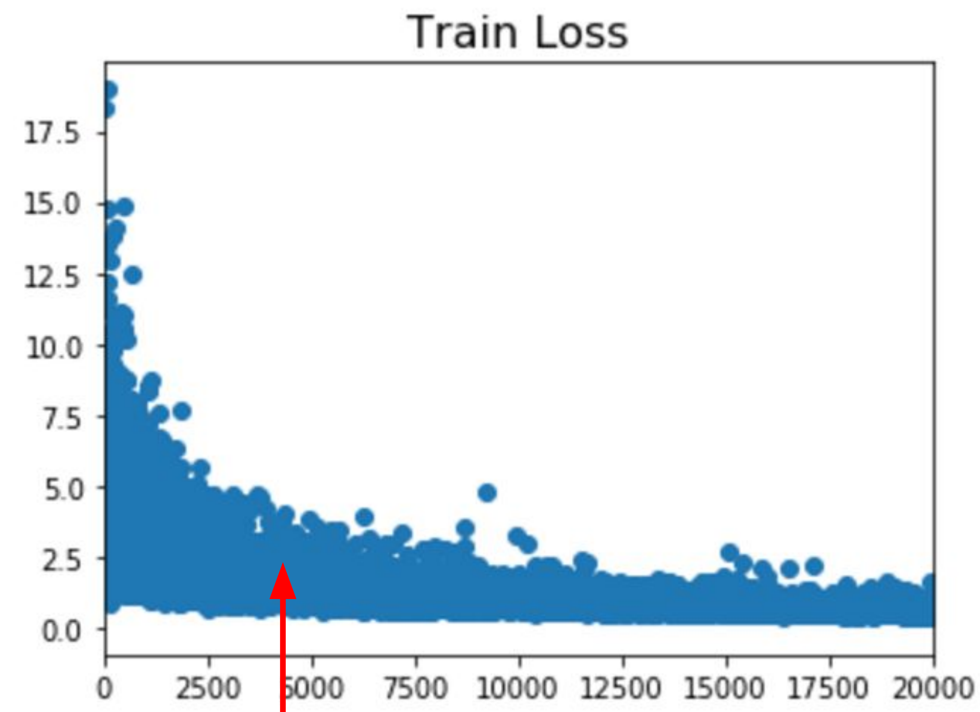
In practice:

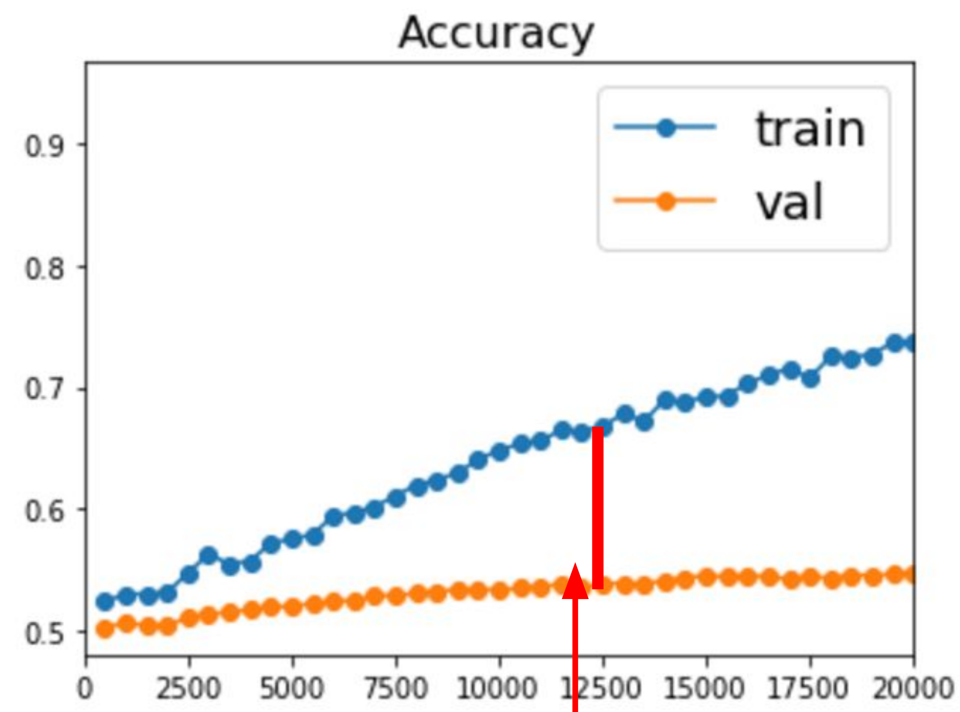
- **Adam** is a good default choice in many cases
- **SGD+Momentum** with learning rate decay often outperforms Adam by a bit, but requires more tuning

- If you can afford to do full batch updates then try out **L-BFGS** (and don't forget to disable all sources of noise)

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Beyond Training Error



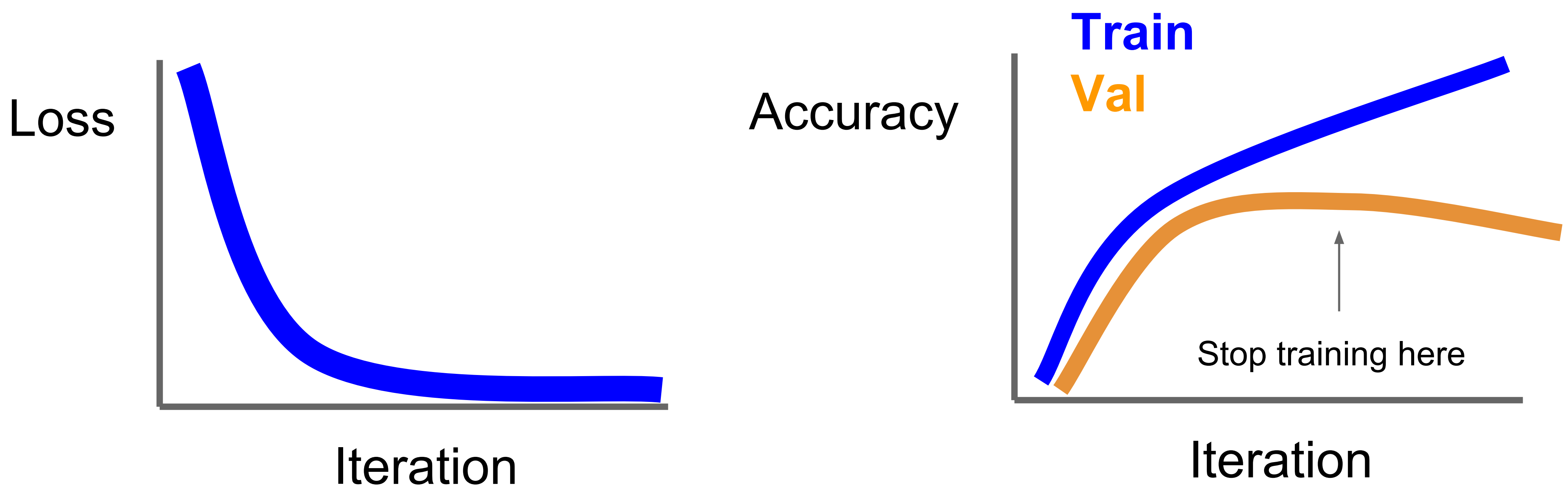


Better optimization algorithms
help reduce training loss

But we really care about error on new
data - how to reduce the gap?

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Early Stopping



Stop training the model when accuracy on the validation set decreases
Or train for a long time, but always keep track of the model snapshot that worked best on val

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Model Ensembles

1. Train multiple independent models
2. At test time average their results

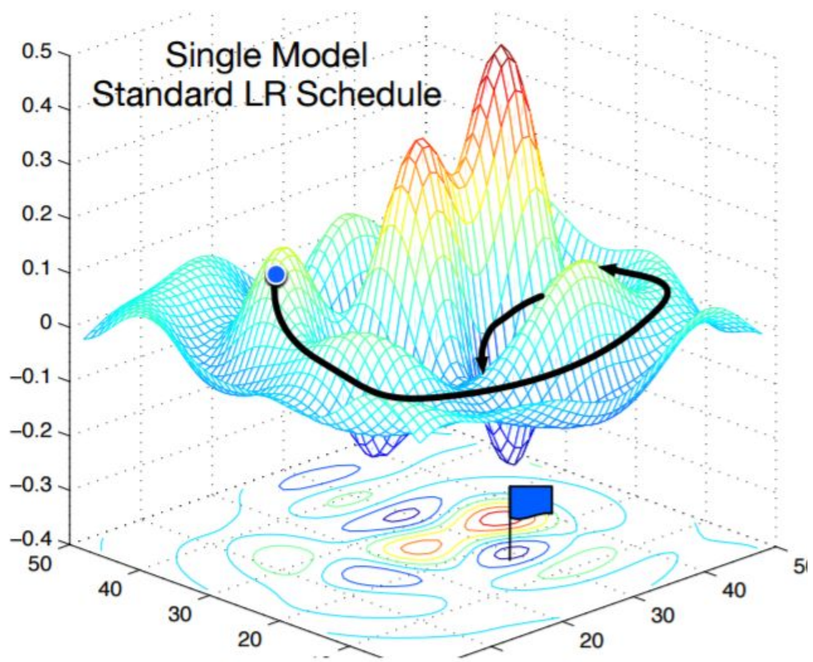
   (Take average of predicted probability distributions, then choose argmax)

Enjoy 2% extra performance

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Model Ensembles: Tips and Tricks

Instead of training independent models, use multiple snapshots of a single model during training!

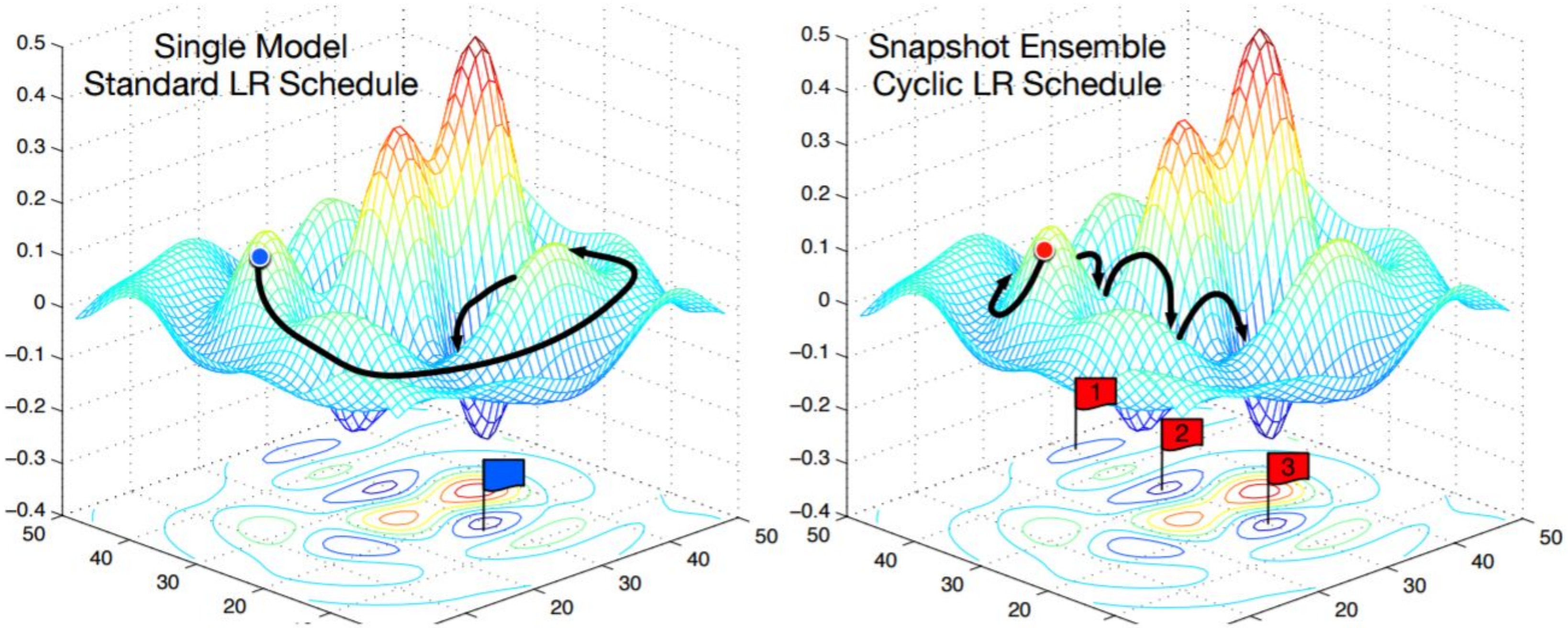


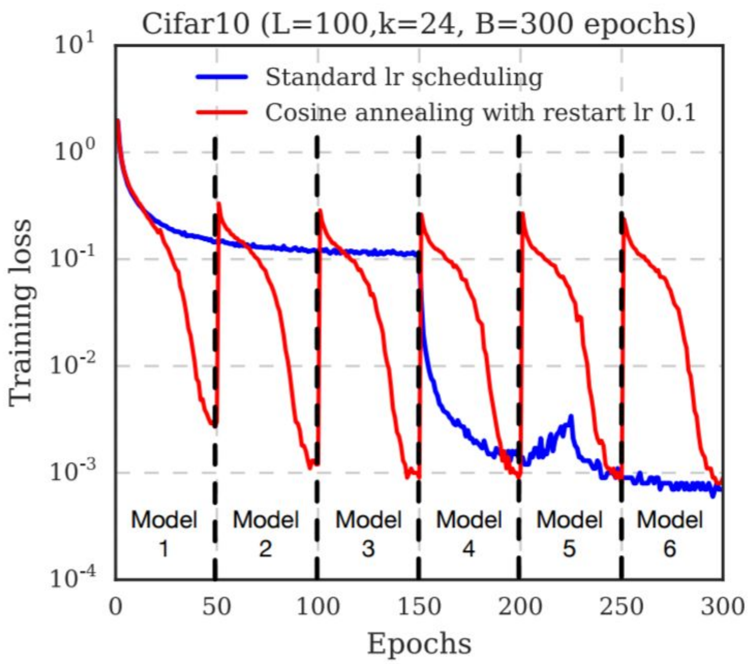
Loshchilov and Hutter, "SGDR: Stochastic gradient descent with restarts", arXiv 2016
Huang et al, "Snapshot ensembles: train 1, get M for free", ICLR 2017
Figures copyright Yixuan Li and Geoff Pleiss, 2017. Reproduced with permission.

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Model Ensembles: Tips and Tricks

Instead of training independent models, use multiple snapshots of a single model during training!





Loshchilov and Hutter, "SGDR: Stochastic gradient descent with restarts", arXiv 2016
Huang et al, "Snapshot ensembles: train 1, get M for free", ICLR 2017
Figures copyright Yixuan Li and Geoff Pleiss, 2017. Reproduced with permission.

Cyclic learning rate schedules can make this work even better!

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Model Ensembles: Tips and Tricks

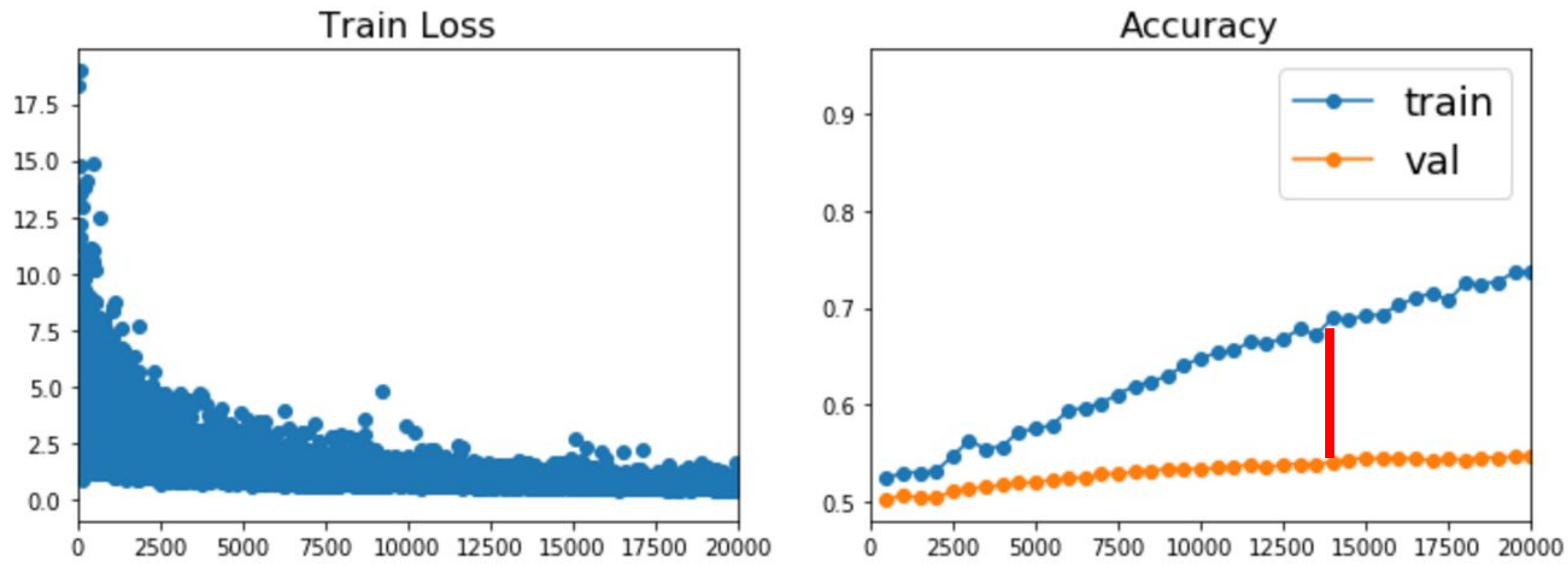
Instead of using actual parameter vector, keep a moving average of the parameter vector and use that at test time (Polyak averaging)

```
while True:
  data_batch = dataset.sample_data_batch()
  loss = network.forward(data_batch)
  dx = network.backward()
  x += - learning_rate * dx
  x_test = 0.995*x_test + 0.005*x # use for test set
```

Polyak and Juditsky, “Acceleration of stochastic approximation by averaging”, SIAM Journal on Control and Optimization, 1992.

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

How to improve single-model performance?



Regularization

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Regularization: Add term to loss

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

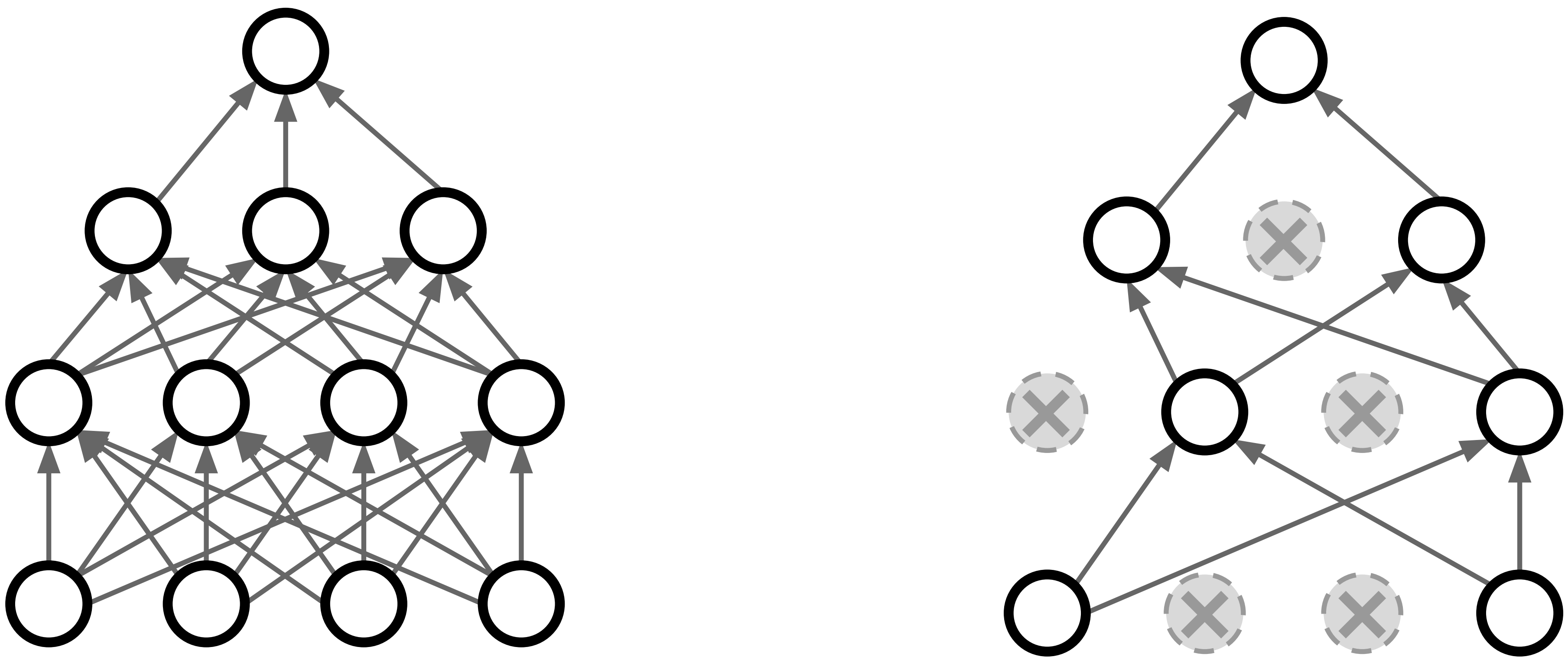
L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Regularization: Dropout

In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common



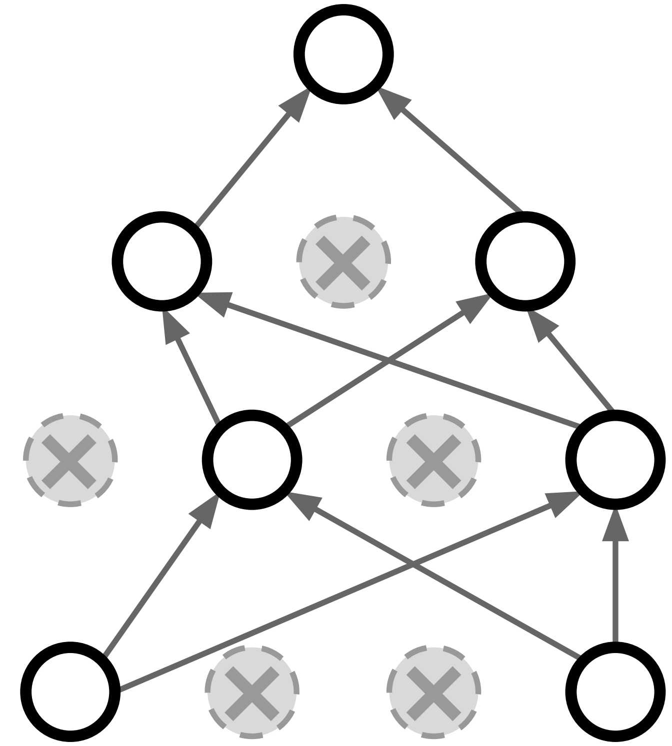
Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Regularization: Dropout

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

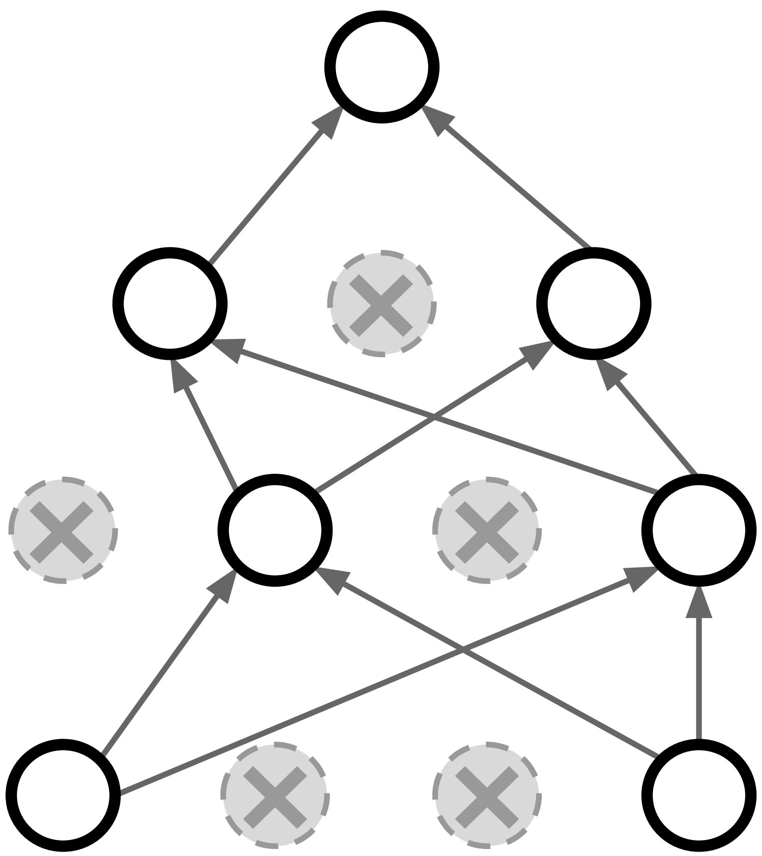
Example forward pass with a 3-layer network using dropout



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Regularization: Dropout

How can this possibly be a good idea?

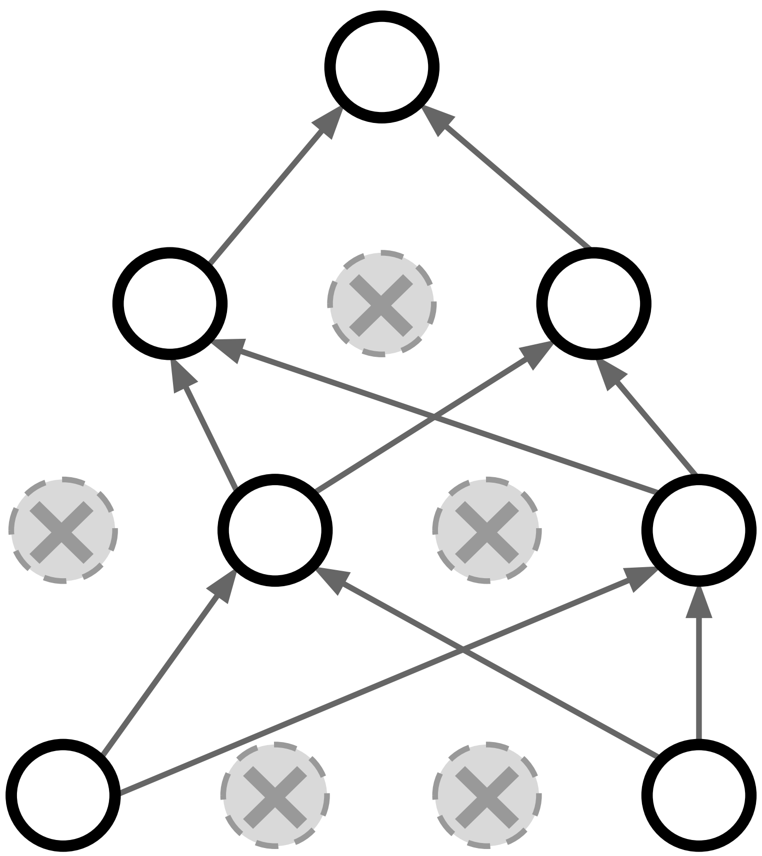


Forces the network to have a redundant representation;
Prevents co-adaptation of features

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Regularization: Dropout

How can this possibly be a good idea?



Another interpretation:

Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

An FC layer with 4096 units has $2^{4096} \sim 10^{1233}$ possible masks!
Only $\sim 10^{82}$ atoms in the universe...

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Dropout: Test time

Dropout makes our output random!

Output (label) Input (image)

$$\boxed{y} = f_W(\boxed{x}, \boxed{z})$$

Random mask

Want to "average out" the randomness at test-time

$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

But this integral seems hard …

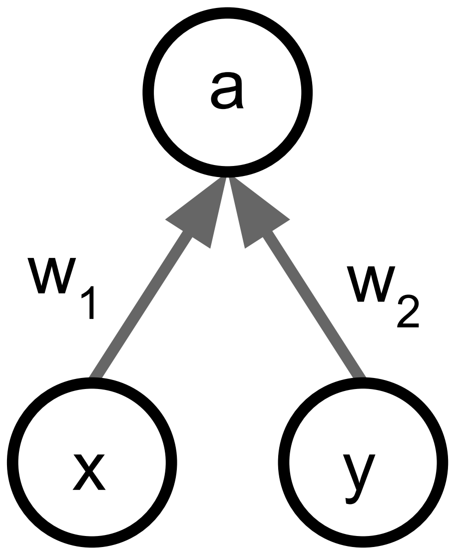
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z)f(x,z)dz$$

Consider a single neuron.

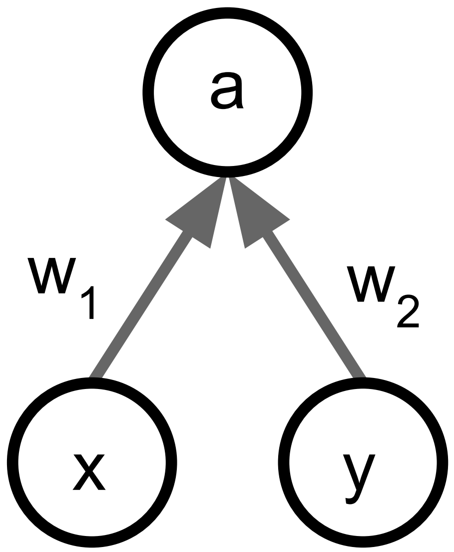


slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$



Consider a single neuron.

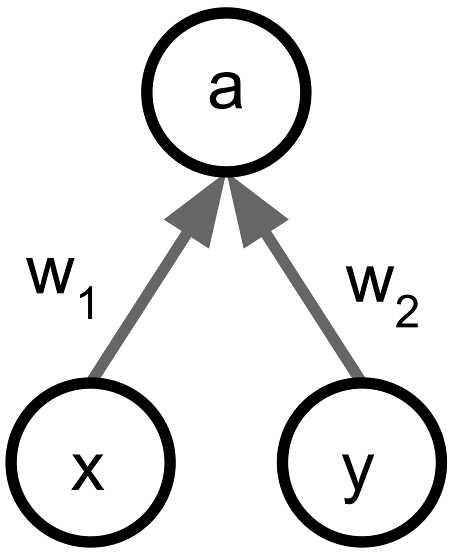
At test time we have: $E\big[a\big] = w_1 x + w_2 y$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z)f(x,z)dz$$



Consider a single neuron.

At test time we have: $E\big[a\big] = w_1x + w_2y$

During training we have:

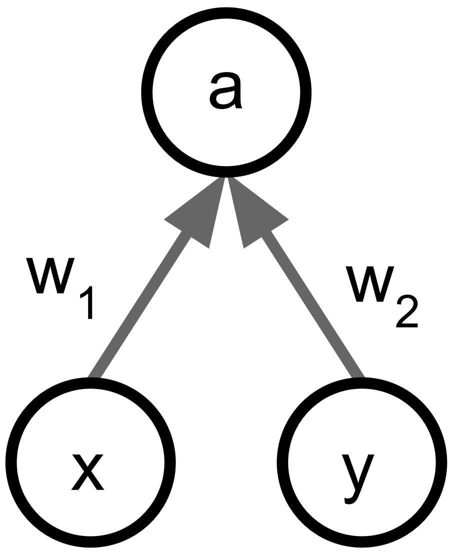
$$\begin{aligned} E\big[a\big] =& \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) \\ =& \frac{1}{2}(w_1x + w_2y) \end{aligned}$$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z\left[f(x, z)\right] = \int p(z) f(x, z) dz$$



Consider a single neuron.

At test time we have: $E\left[a\right] = w_1 x + w_2 y$

During training we have:

$$\begin{aligned} E\left[a\right] =& \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ =& \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

At test time, **multiply** by dropout probability

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Dropout: Test time

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

At test time all neurons are active always
=> We must scale the activations so that for each neuron:
<u>output at test time</u> = <u>expected output at training time</u>

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Dropout Summary

```python
""" Vanilla Dropout: Not recommended implementation (see notes below) """

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

drop in forward pass

scale at test time

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

More common: “Inverted dropout”

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

test time is unchanged!

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Regularization: A common pattern

Training: Add some kind of randomness

$$y = f_W(x, z)$$

Testing: Average out randomness (sometimes approximate)

$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Regularization: A common pattern

Training: Add some kind of randomness

$$y = f_W(x, z)$$

Testing: Average out randomness (sometimes approximate)

$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

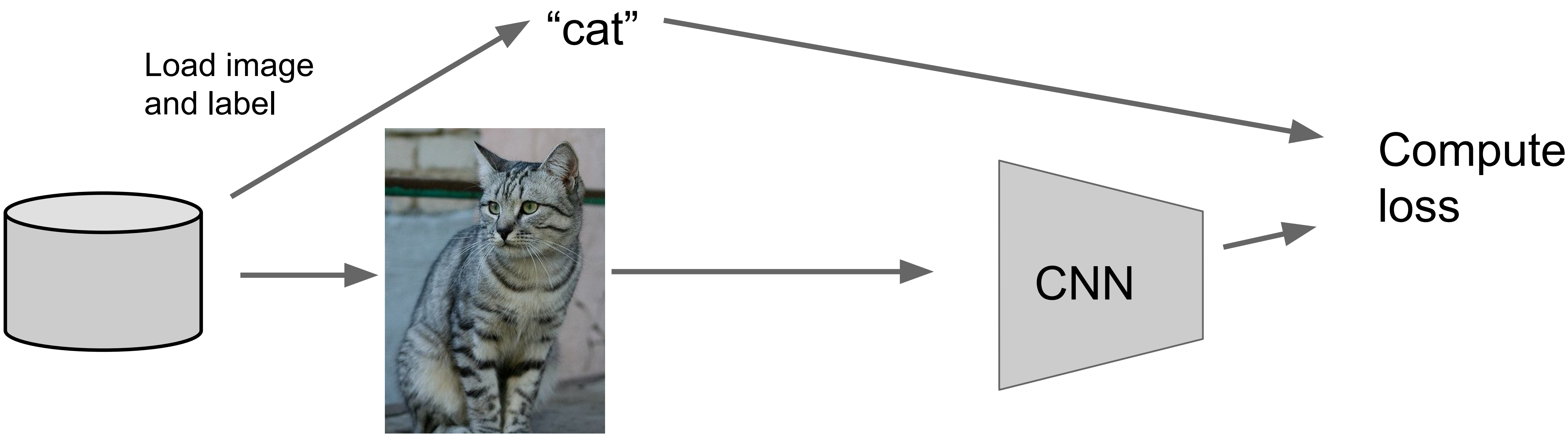
Example: Batch Normalization

Training: Normalize using stats from random minibatches

Testing: Use fixed stats to normalize

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

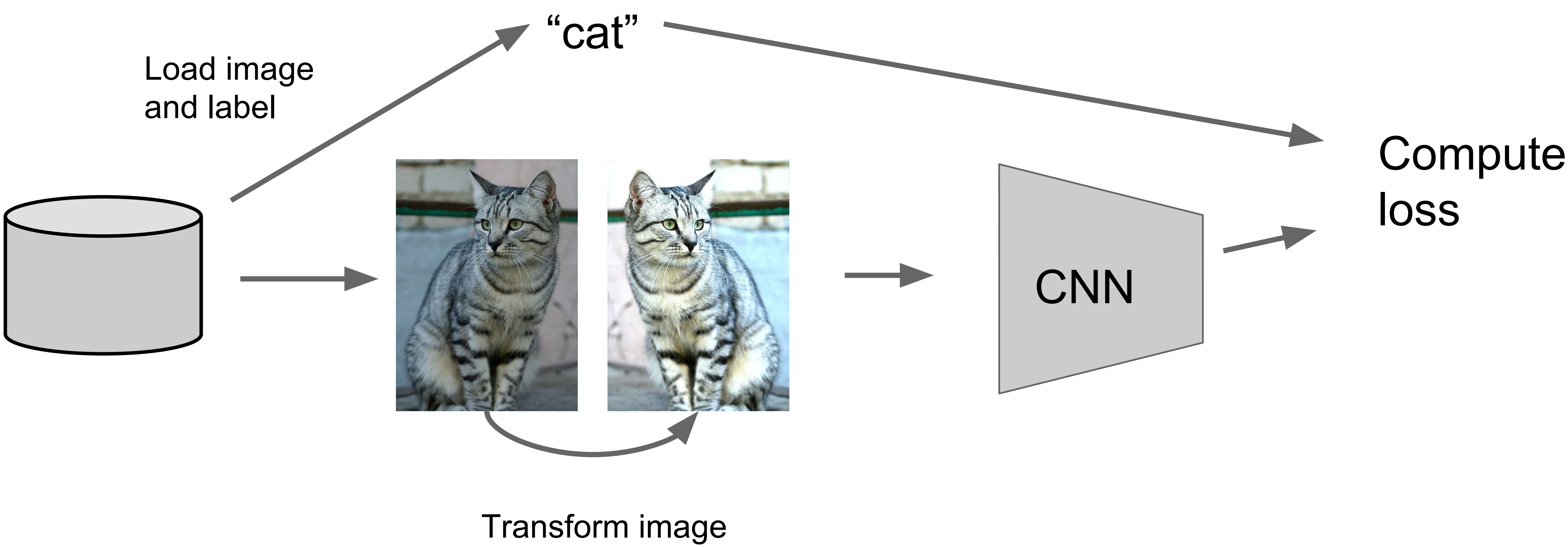
Regularization: Data Augmentation



This image by Nikita is
licensed under CC-BY 2.0

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Regularization: Data Augmentation



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Data Augmentation

Horizontal Flips

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

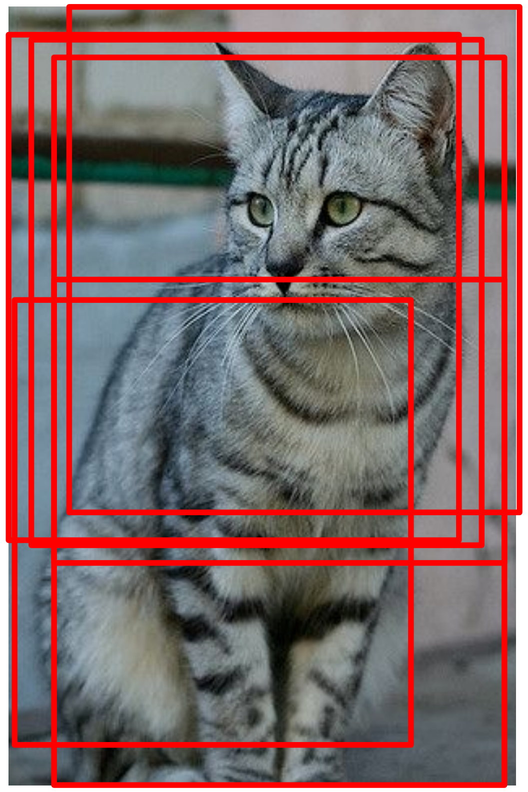
Data Augmentation

Random crops and scales

Training: sample random crops / scales

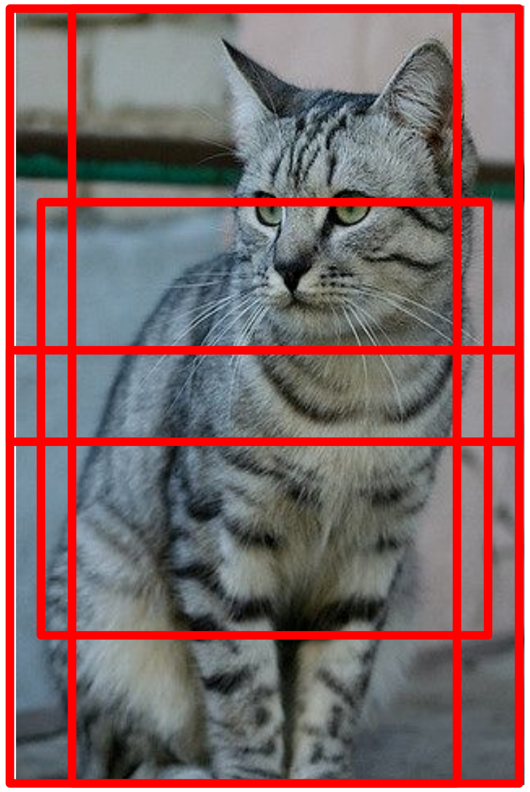
ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Data Augmentation
Random crops and scales



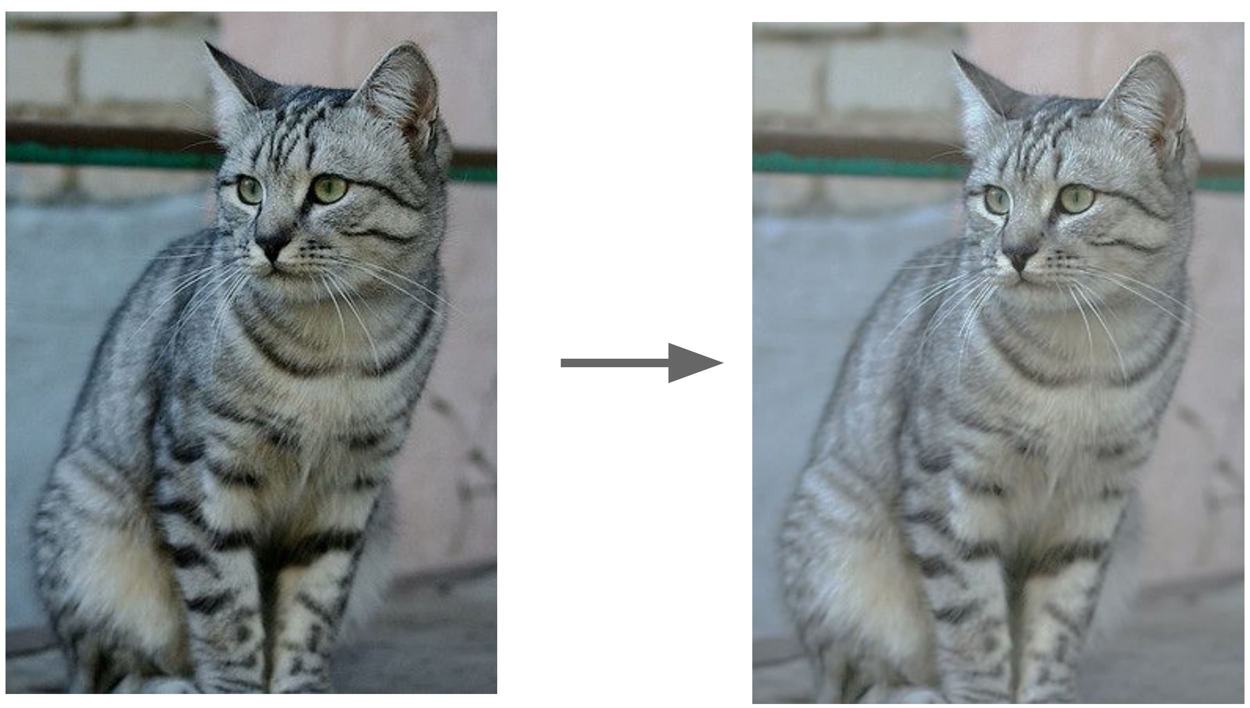
Training: sample random crops / scales
ResNet:
1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

Testing: average a fixed set of crops
ResNet:
1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224 x 224 crops: 4 corners + center, + flips

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Data Augmentation
Color Jitter

Simple: Randomize
contrast and brightness



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Data Augmentation
Color Jitter

Simple: Randomize contrast and brightness

More Complex:

1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a "color offset" along principal component directions
3. Add offset to all pixels of a training image

(As seen in *[Krizhevsky et al. 2012]*, ResNet, etc)

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Data Augmentation

Get creative for your problem!

Random mix/combinations of :
- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Regularization: A common pattern

Training: Add random noise
Testing: Marginalize over the noise

Examples:
Dropout
Batch Normalization
Data Augmentation

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Regularization: A common pattern

Training: Add random noise

Testing: Marginalize over the noise

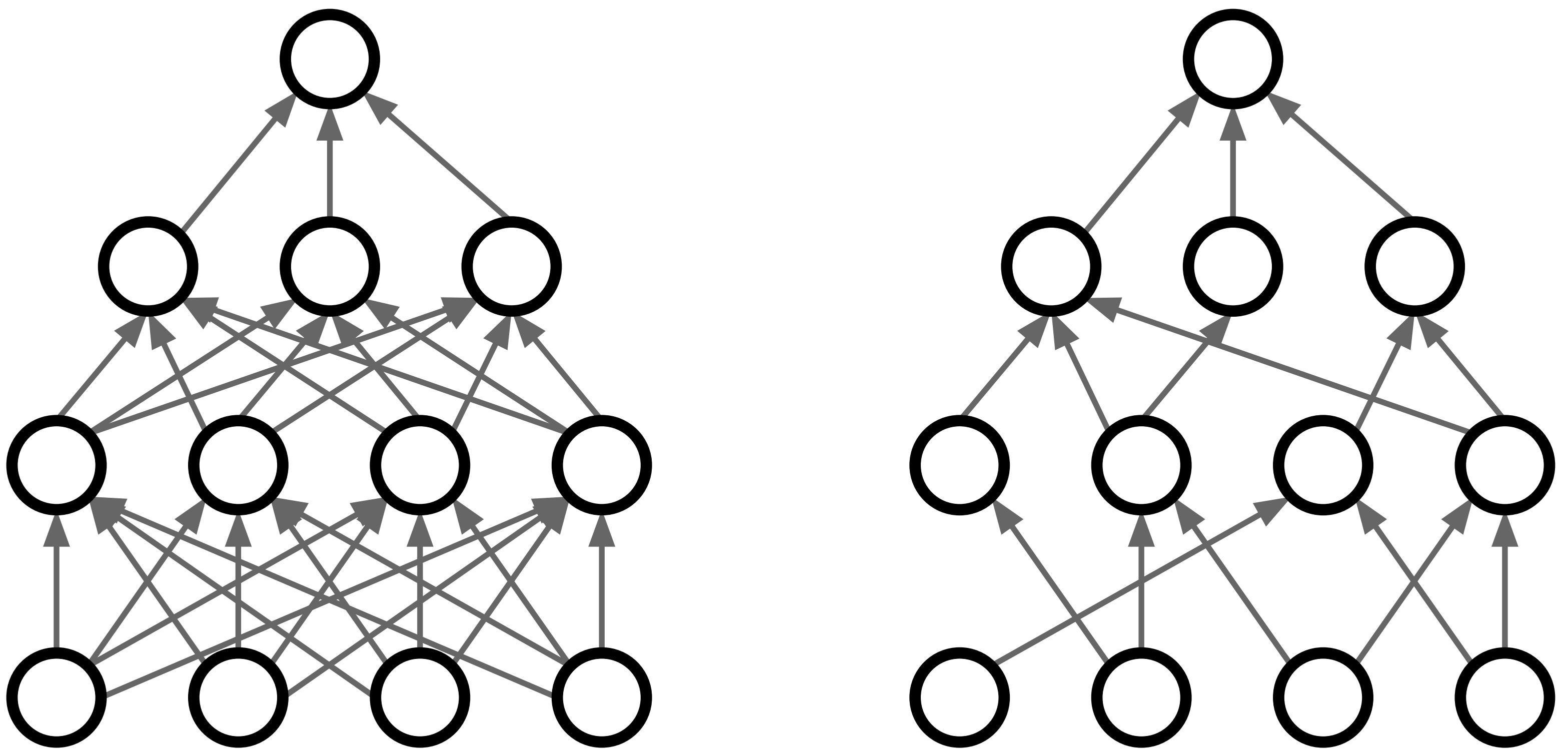
Examples:

Dropout

Batch Normalization

Data Augmentation

DropConnect



Wan et al, "Regularization of Neural Networks using DropConnect", ICML 2013

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Regularization: A common pattern

Training: Add random noise

Testing: Marginalize over the noise

Examples:

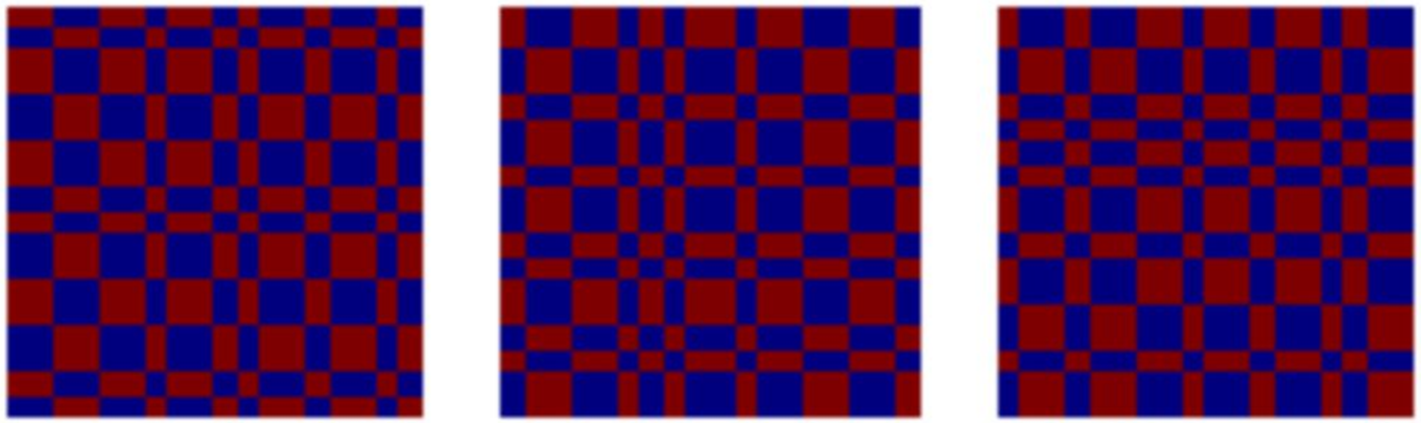
Dropout

Batch Normalization

Data Augmentation

DropConnect

Fractional Max Pooling

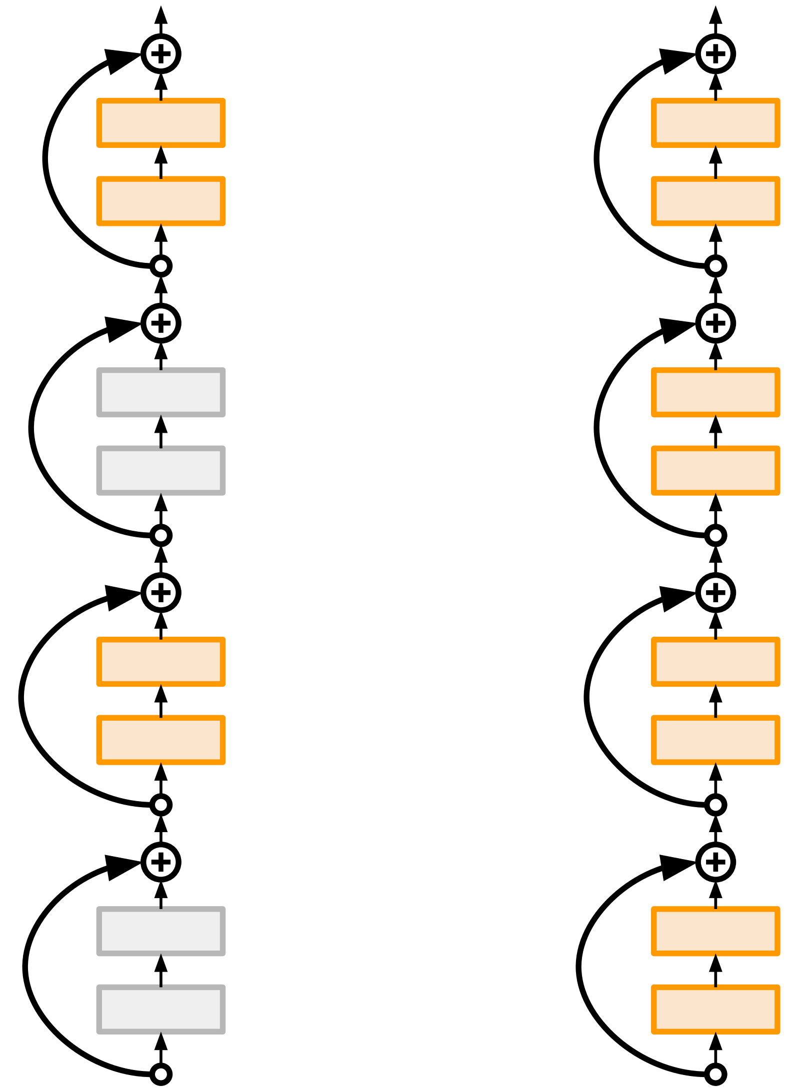


Graham, “Fractional Max Pooling”, arXiv 2014

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Regularization: A common pattern

Training: Add random noise
Testing: Marginalize over the noise



Examples:
Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth

Huang et al, “Deep Networks with Stochastic Depth”, ECCV 2016

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Transfer Learning

"You need a lot of a data if you want to train/use CNNs"

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Transfer Learning

"You need a lot of a data if you want to train/use CNNs"

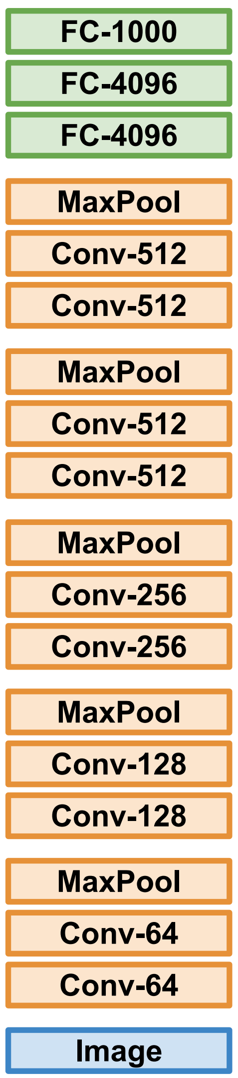
BUSTED

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

1. Train on Imagenet



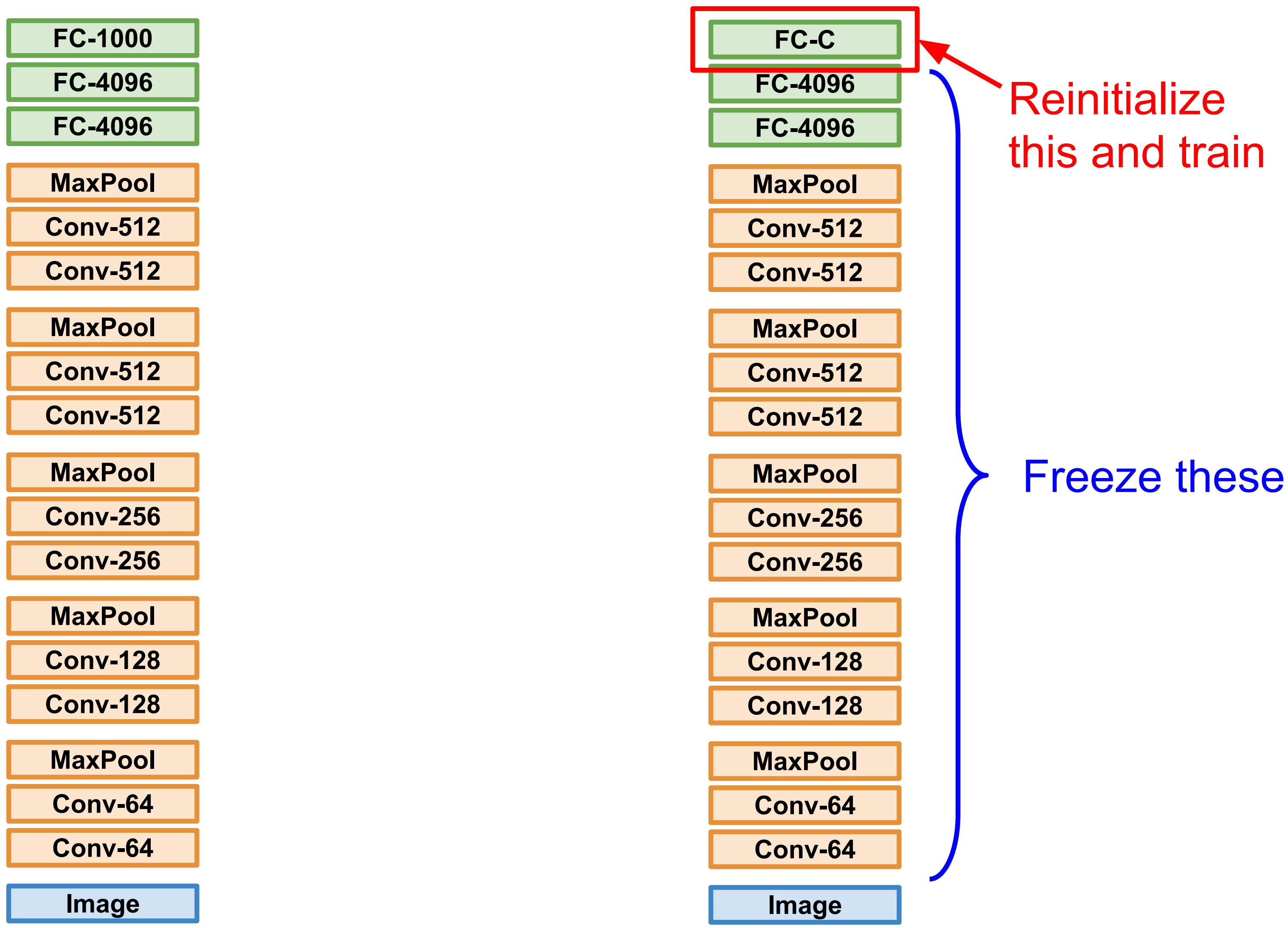
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014
Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

Transfer Learning with CNNs

1. Train on Imagenet

2. Small Dataset (C classes)



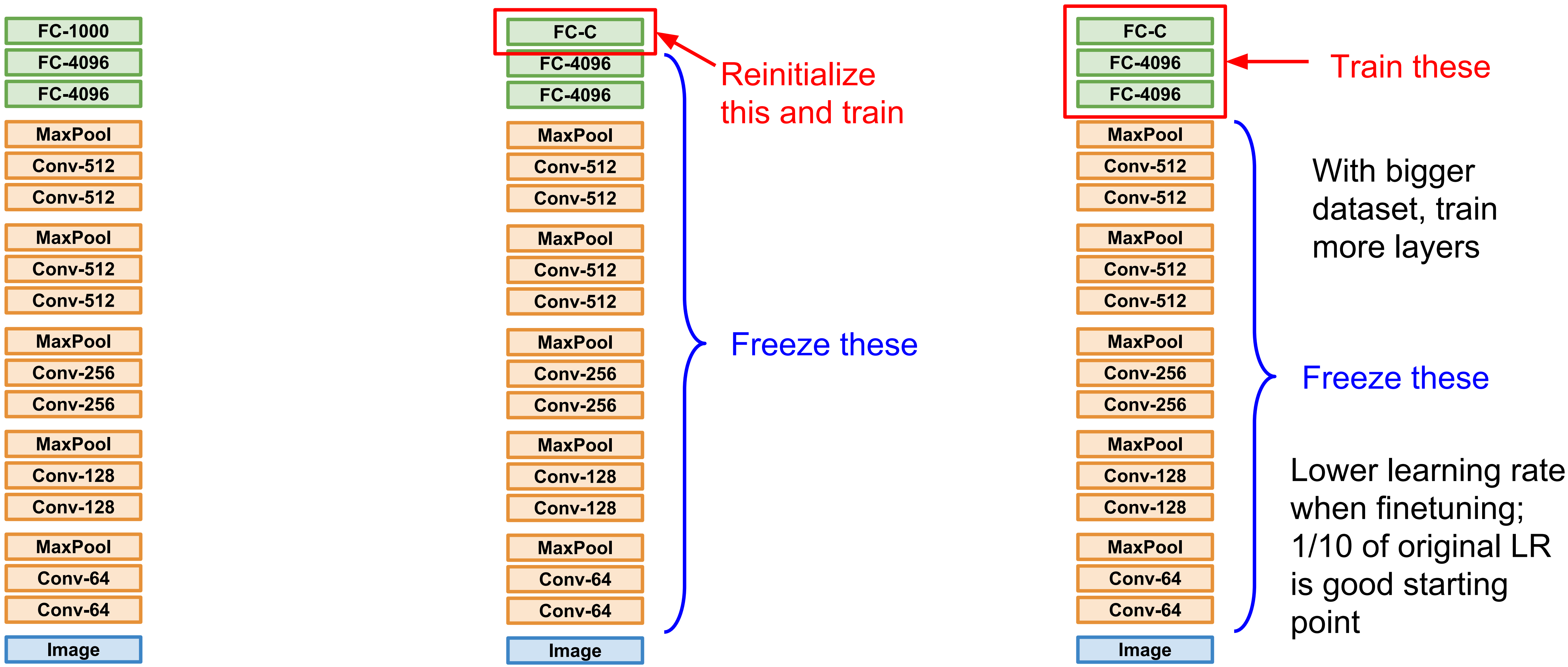
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Transfer Learning with CNNs

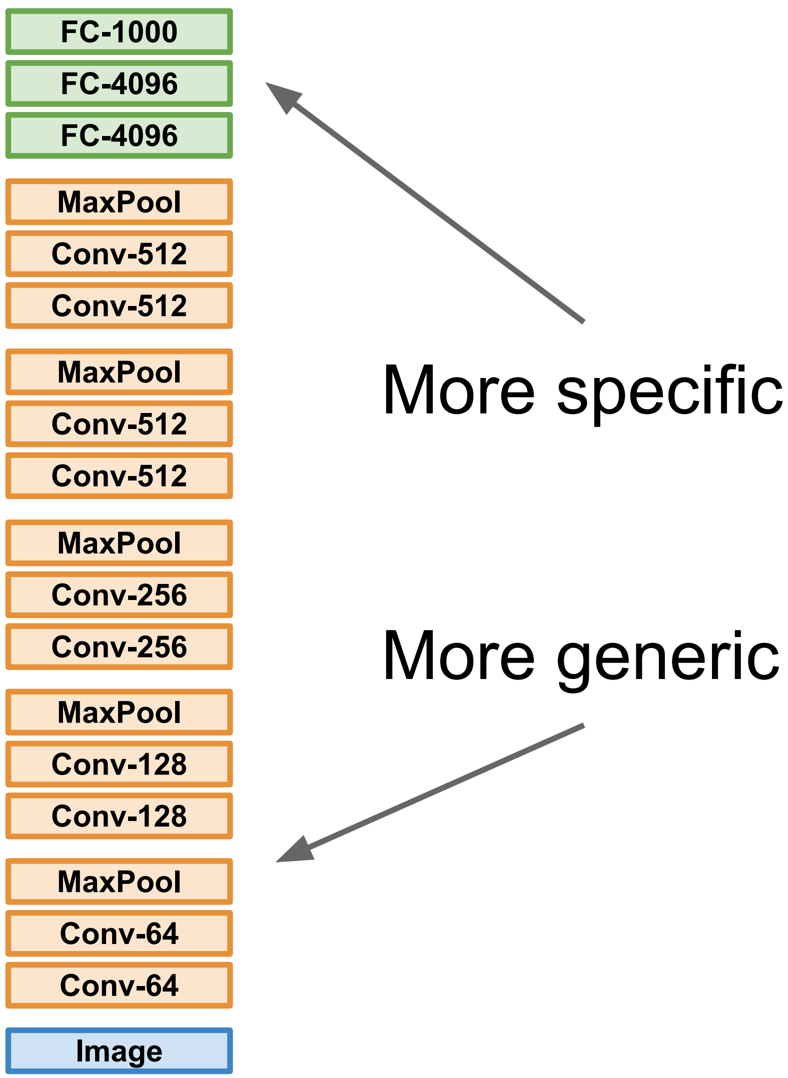
1. Train on Imagenet

2. Small Dataset (C classes)

3. Bigger dataset

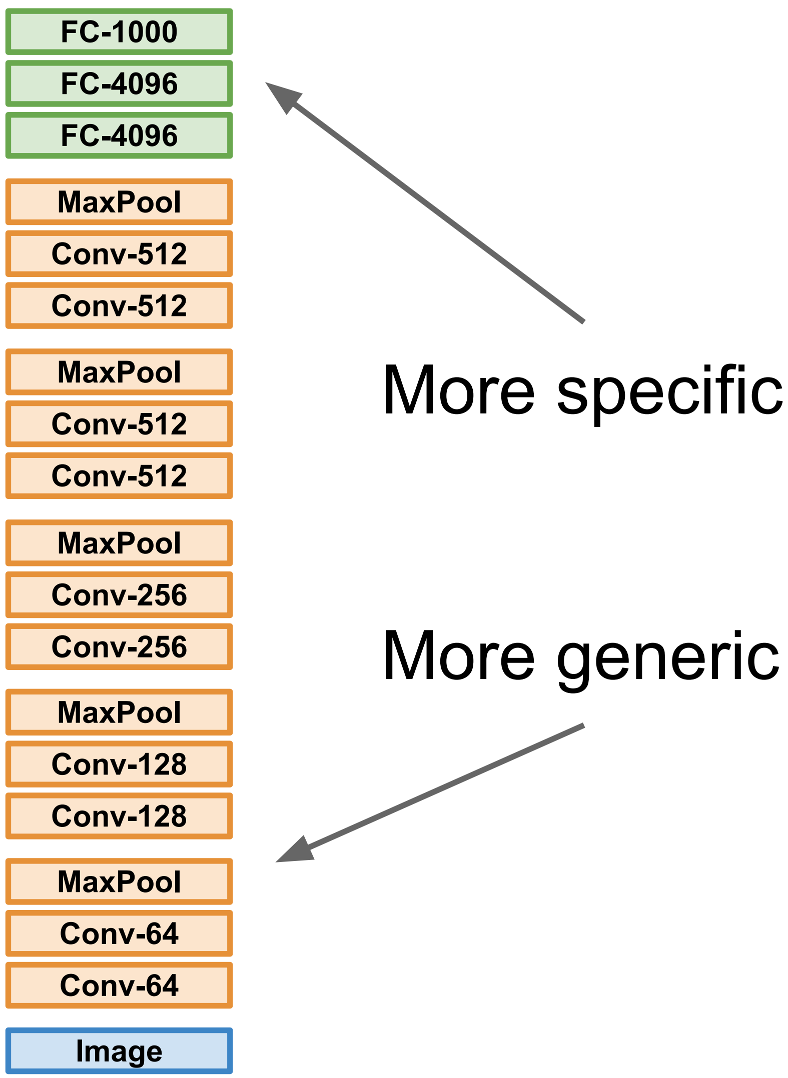


slide credit: Fei-Fei, Justin Johnson, Serena Yeung



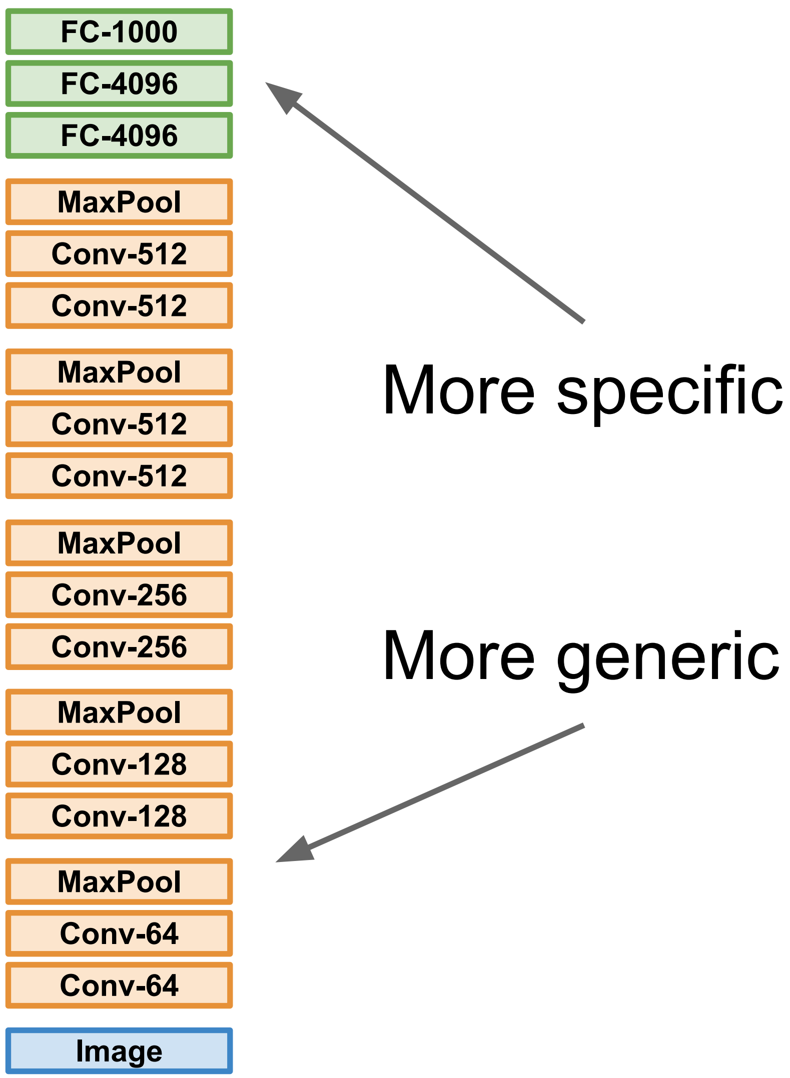
	very similar dataset	**very different dataset**
very little data	?	?
quite a lot of data	?	?

slide credit: Fei-Fei, Justin Johnson, Serena Yeung



	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	?
quite a lot of data	Finetune a few layers	?

slide credit: Fei-Fei, Justin Johnson, Serena Yeung



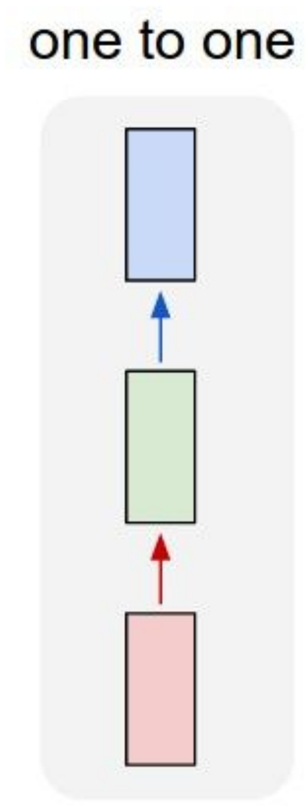
	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You’re in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Recurrent Neural Networks

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

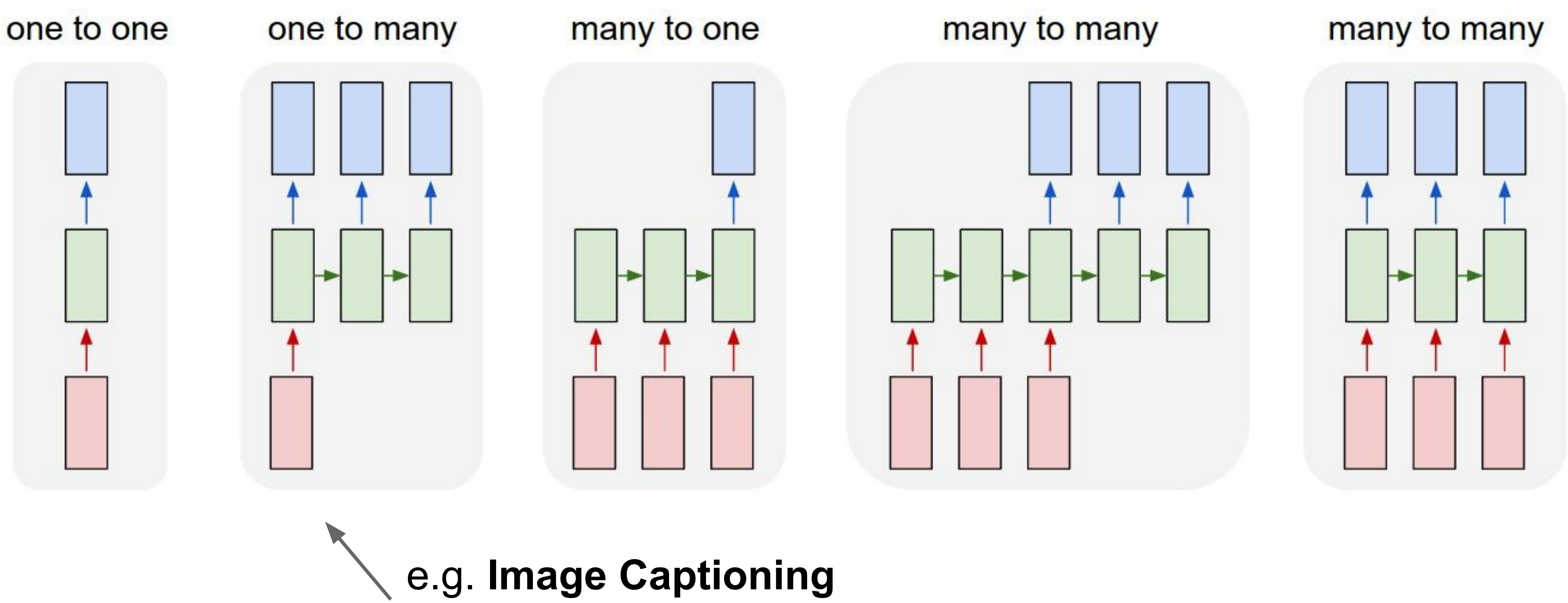
“Vanilla” Neural Network



Vanilla Neural Networks

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

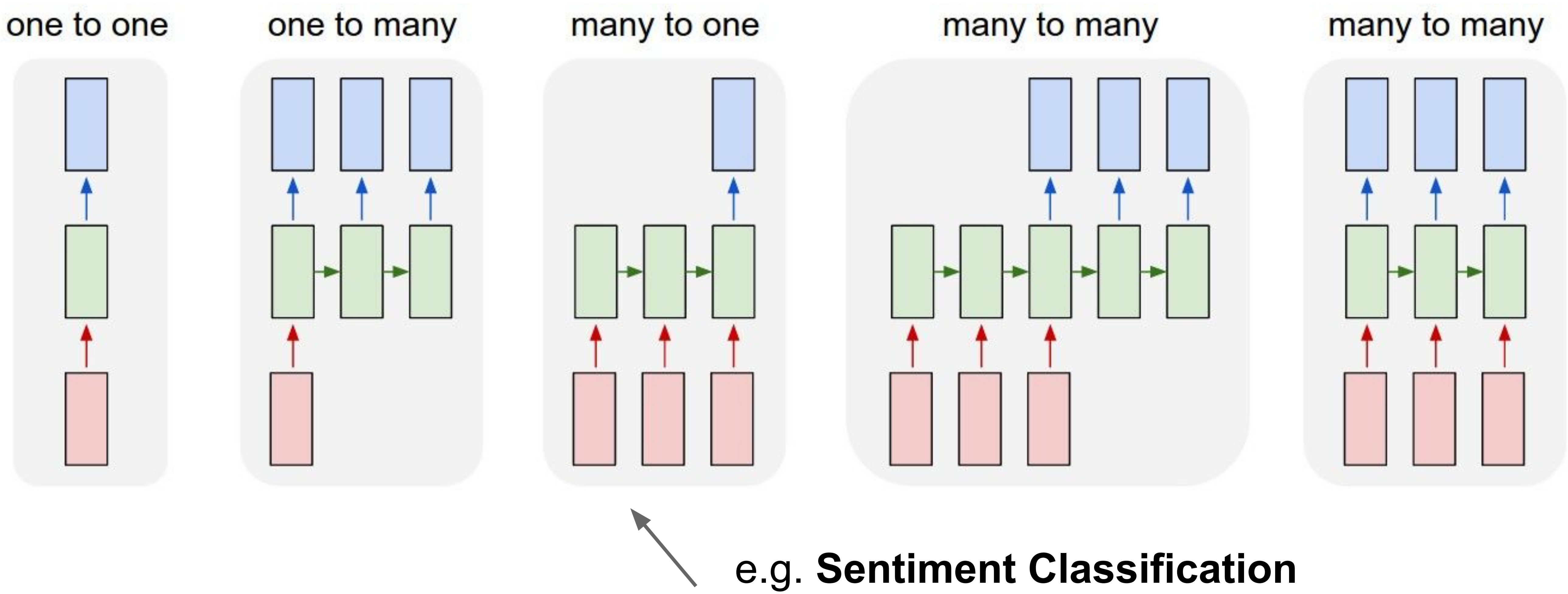
Recurrent Neural Networks: Process Sequences



e.g. **Image Captioning**
image -> sequence of words

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

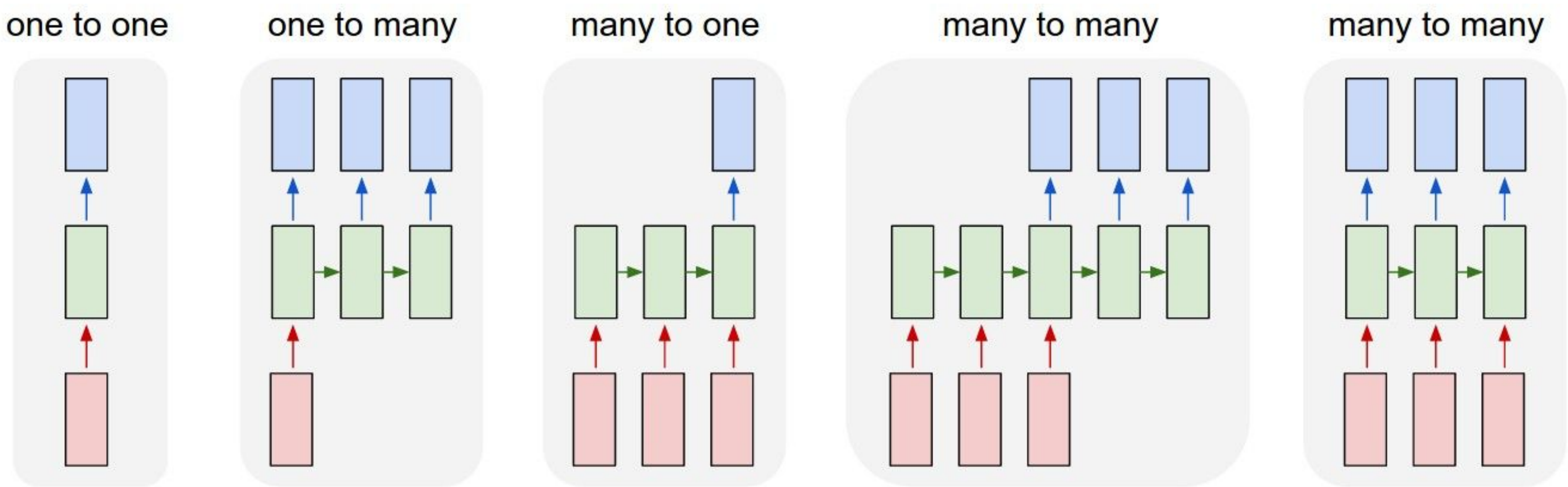
Recurrent Neural Networks: Process Sequences



e.g. **Sentiment Classification**
sequence of words -> sentiment

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

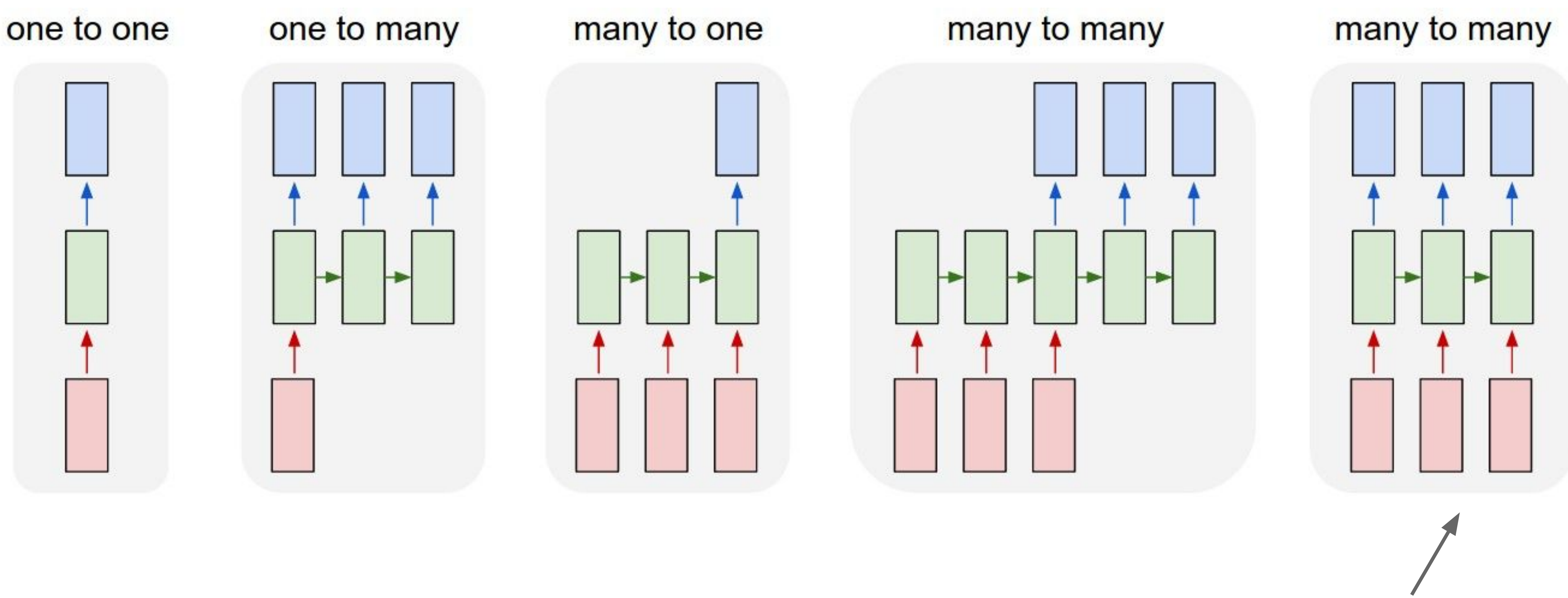
Recurrent Neural Networks: Process Sequences



e.g. **Machine Translation**
seq of words -> seq of words

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Recurrent Neural Networks: Process Sequences



e.g. **Video classification on frame level**

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Sequential Processing of Non-Sequence Data

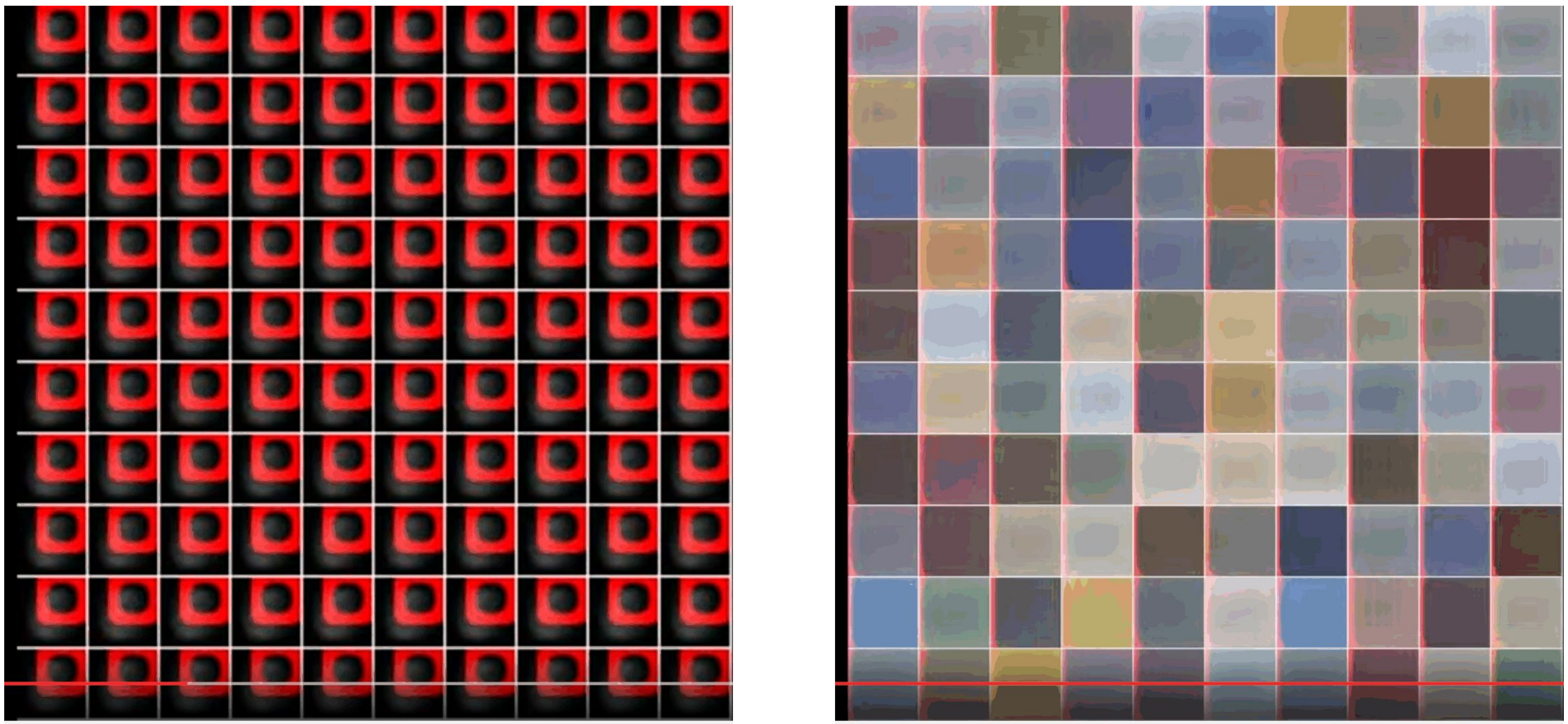
Classify images by taking a series of “glimpses”

Ba, Mnih, and Kavukcuoglu, “Multiple Object Recognition with Visual Attention”, ICLR 2015.
Gregor et al, “DRAW: A Recurrent Neural Network For Image Generation”, ICML 2015
Figure copyright Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra, 2015. Reproduced with permission.

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Sequential Processing of Non-Sequence Data

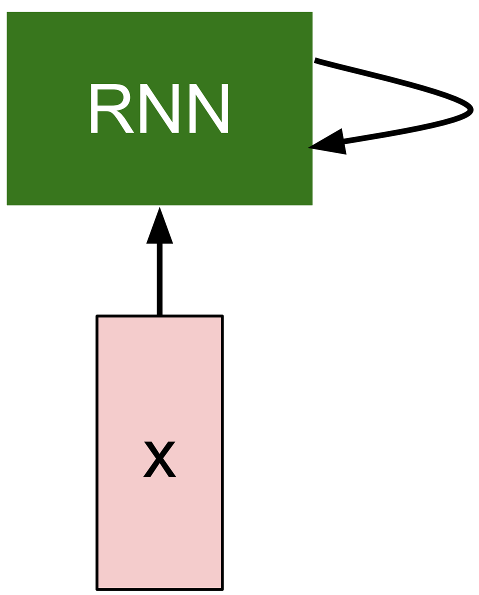
Generate images one piece at a time!



Gregor et al, "DRAW: A Recurrent Neural Network For Image Generation", ICML 2015
Figure copyright Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra, 2015. Reproduced with permission.

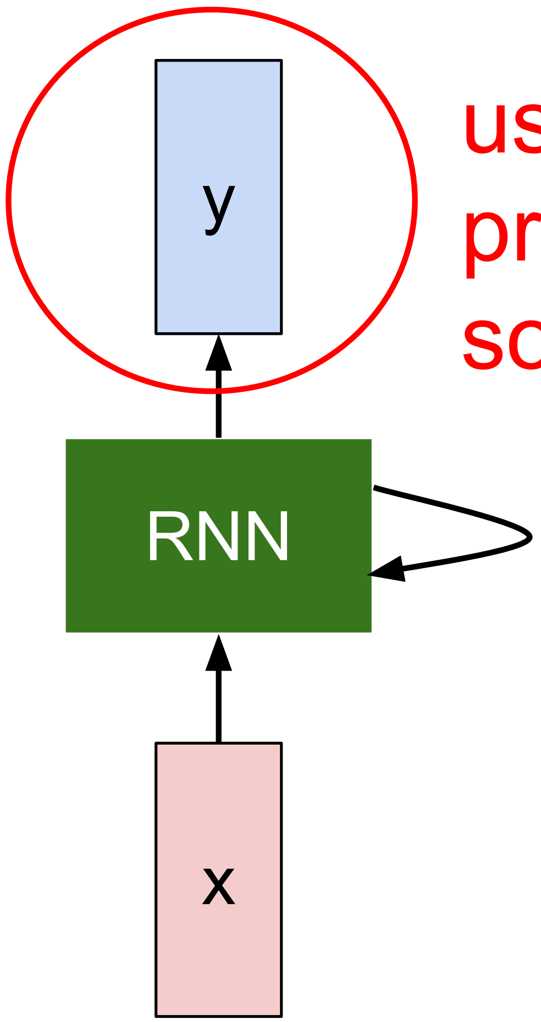
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Recurrent Neural Network



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Recurrent Neural Network



usually want to predict a vector at some time steps

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

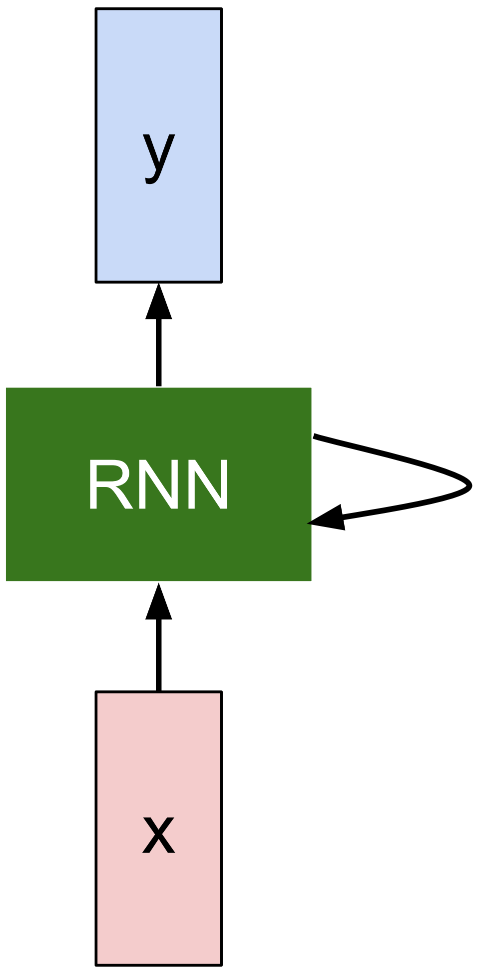
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

some function with parameters W

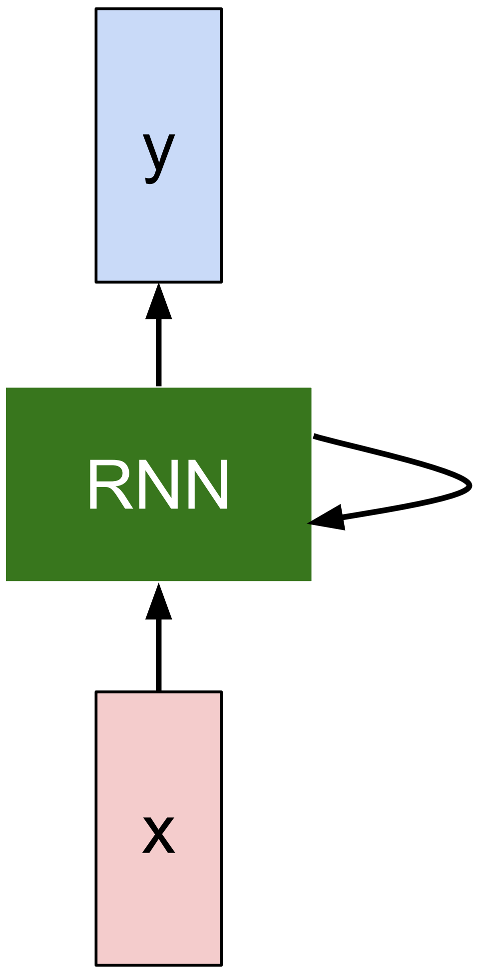


slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

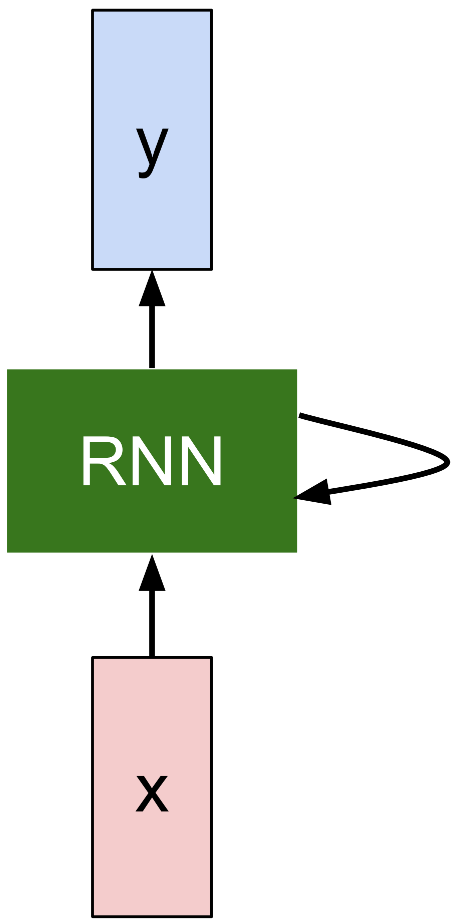


Notice: the same function and the same set of parameters are used at every time step.

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

(Simple) Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:



$$h_t = f_W(h_{t-1}, x_t)$$

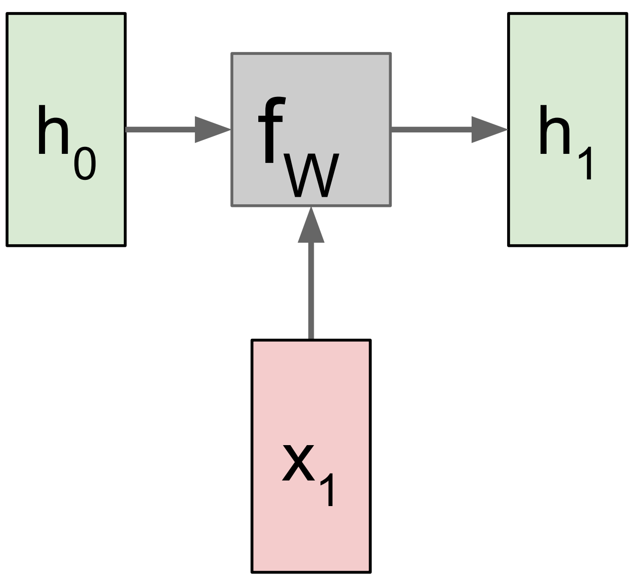
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

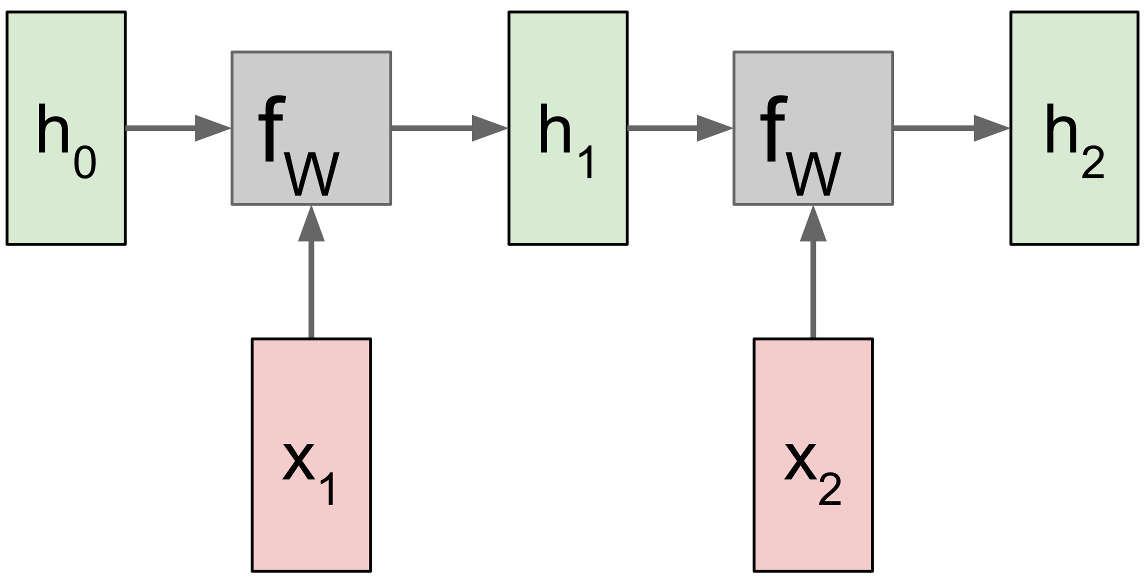
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

RNN: Computational Graph



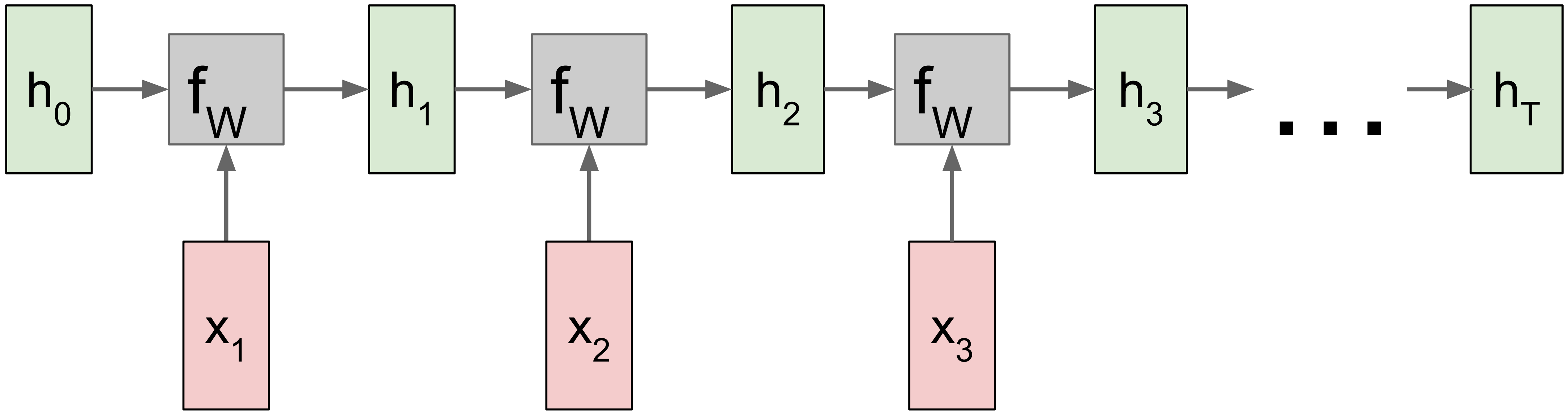
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

RNN: Computational Graph



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

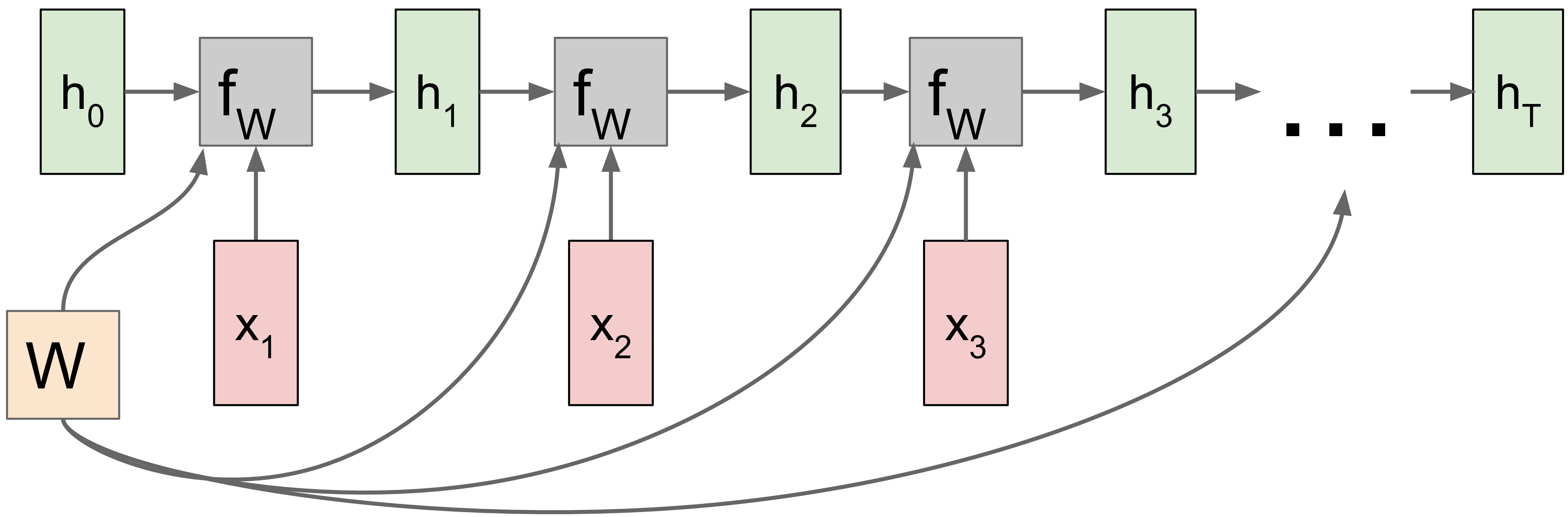
RNN: Computational Graph



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

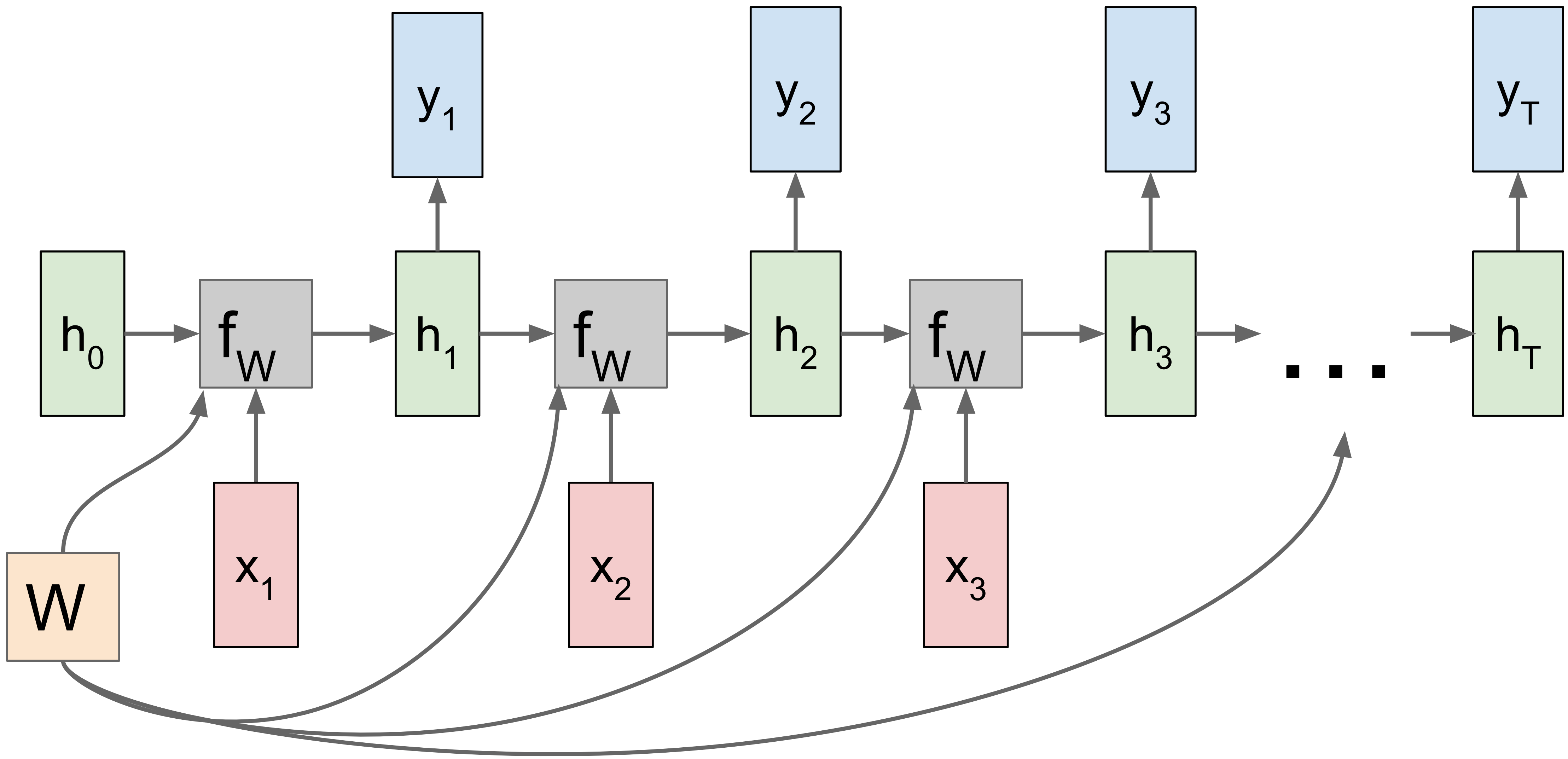
RNN: Computational Graph

Re-use the same weight matrix at every time-step



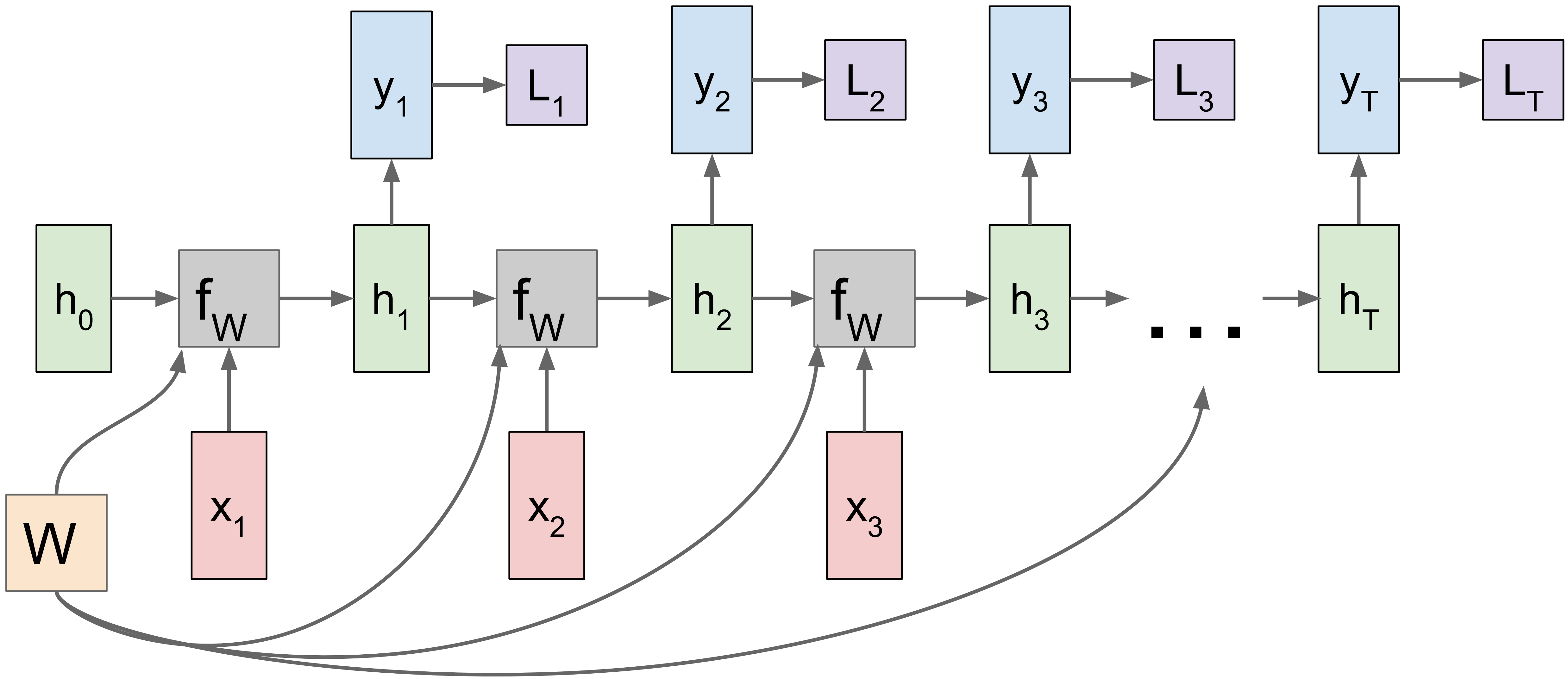
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

RNN: Computational Graph: Many to Many



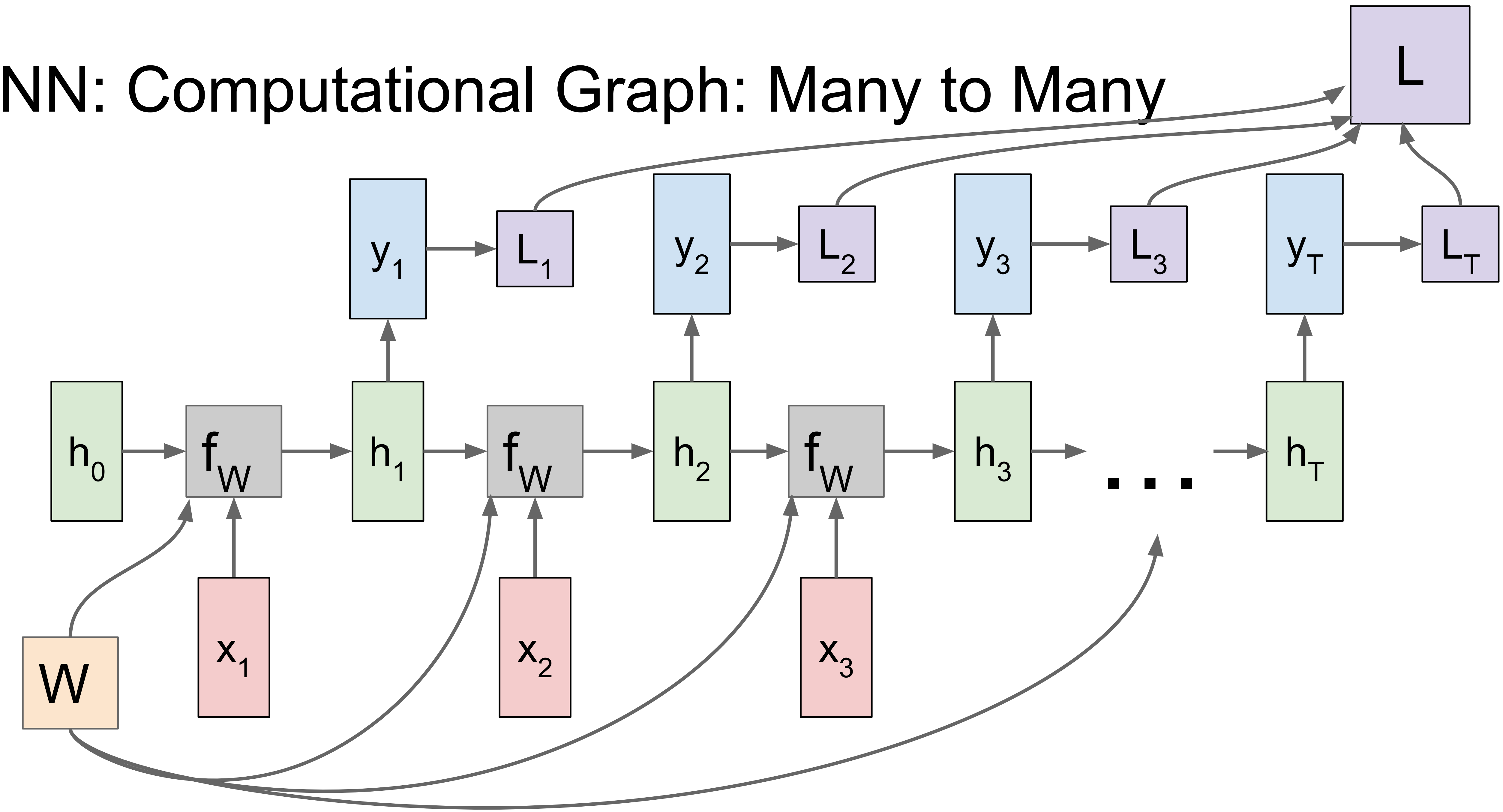
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

RNN: Computational Graph: Many to Many



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

RNN: Computational Graph: Many to Many



slide credit: Fei-Fei, Justin Johnson, Serena Yeung