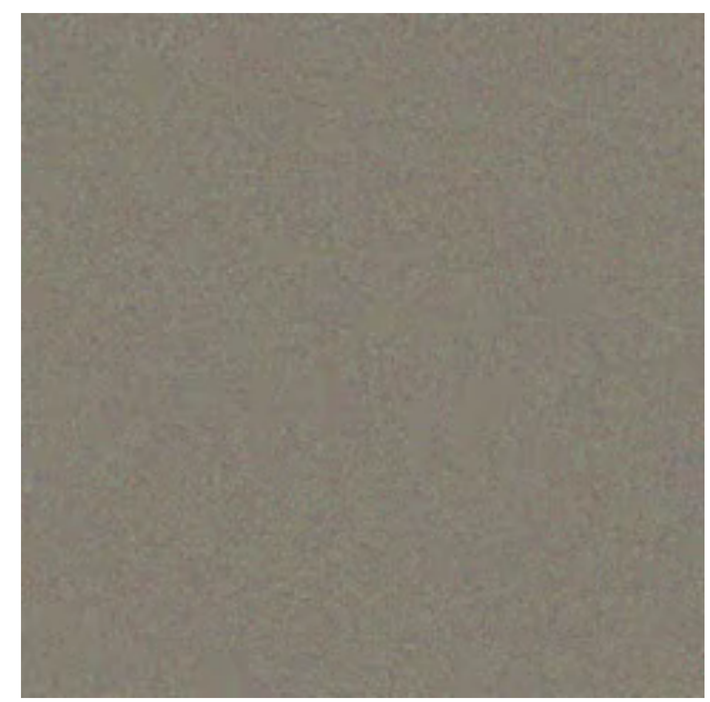
Lecture 39: Training Neural Networks (Cont'd)

CS 4670/5670
Sean Bell

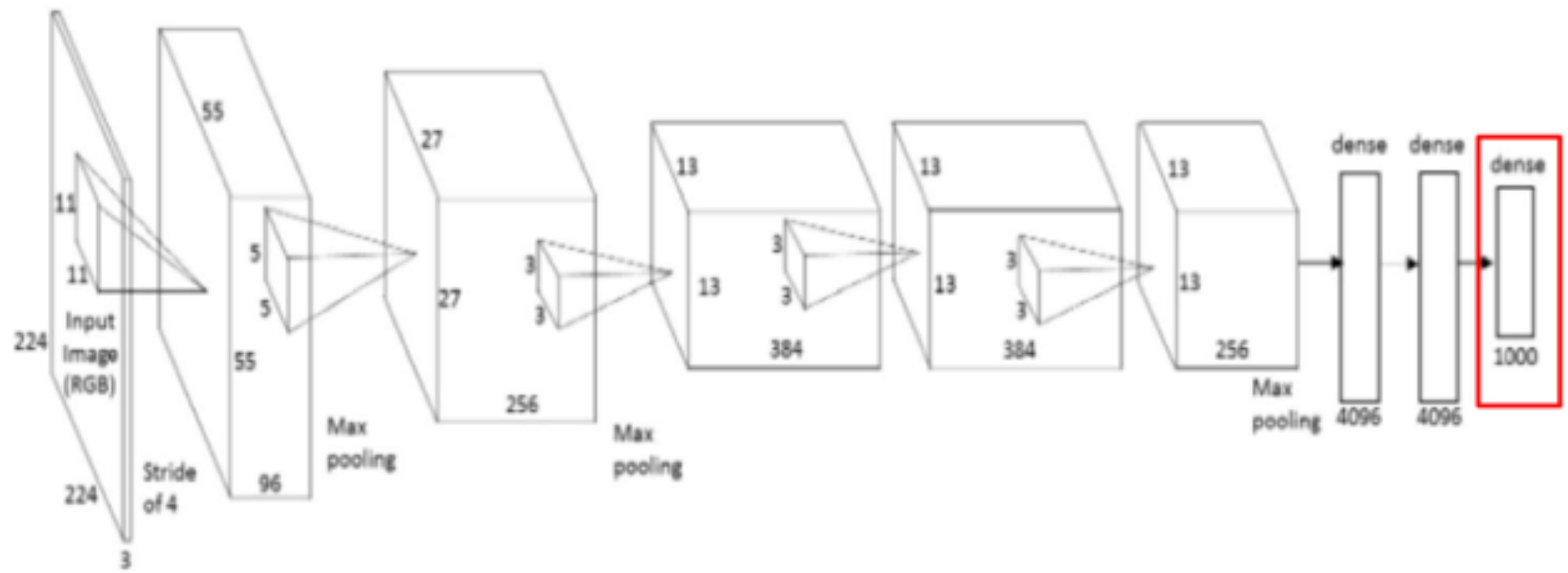
Strawberry

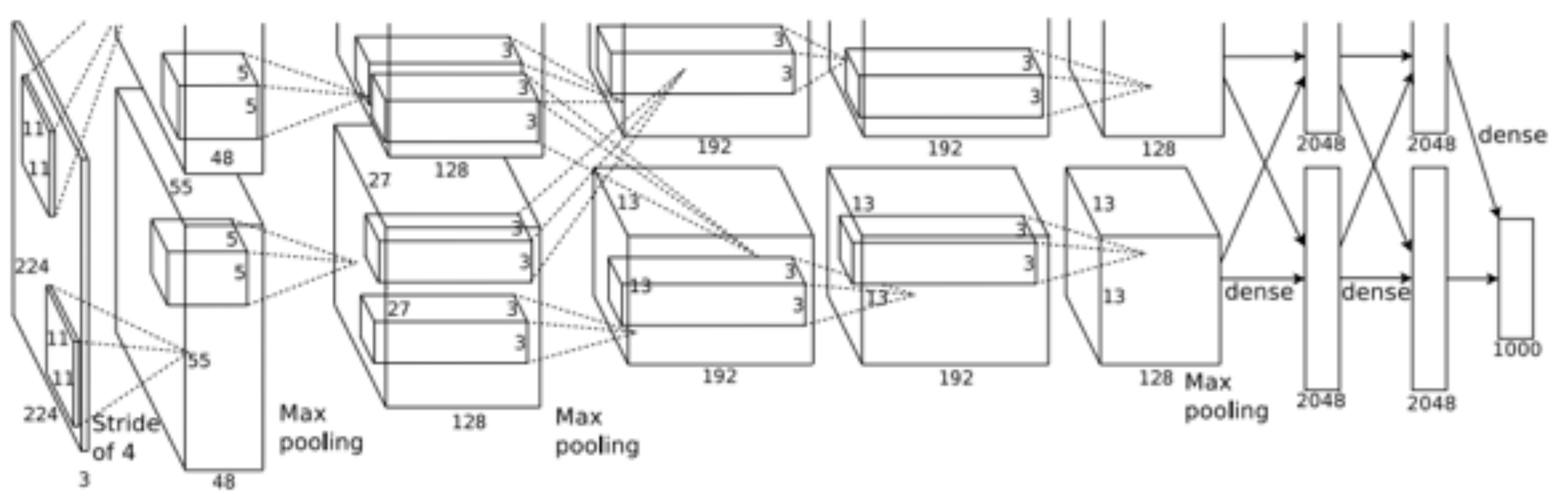
Goblet



Throne

(Side Note for PA5) AlexNet: 1 vs 2 parts





Caffe represents caffe like the above image, but computes as if it were the bottom image using 2 "groups"

(Recall) Each iteration of training

(1) Forward Propagation:

$$x \to \boxed{\text{Function}} \to h \to \cdots \to \boxed{\text{Function}} \to s \to L$$

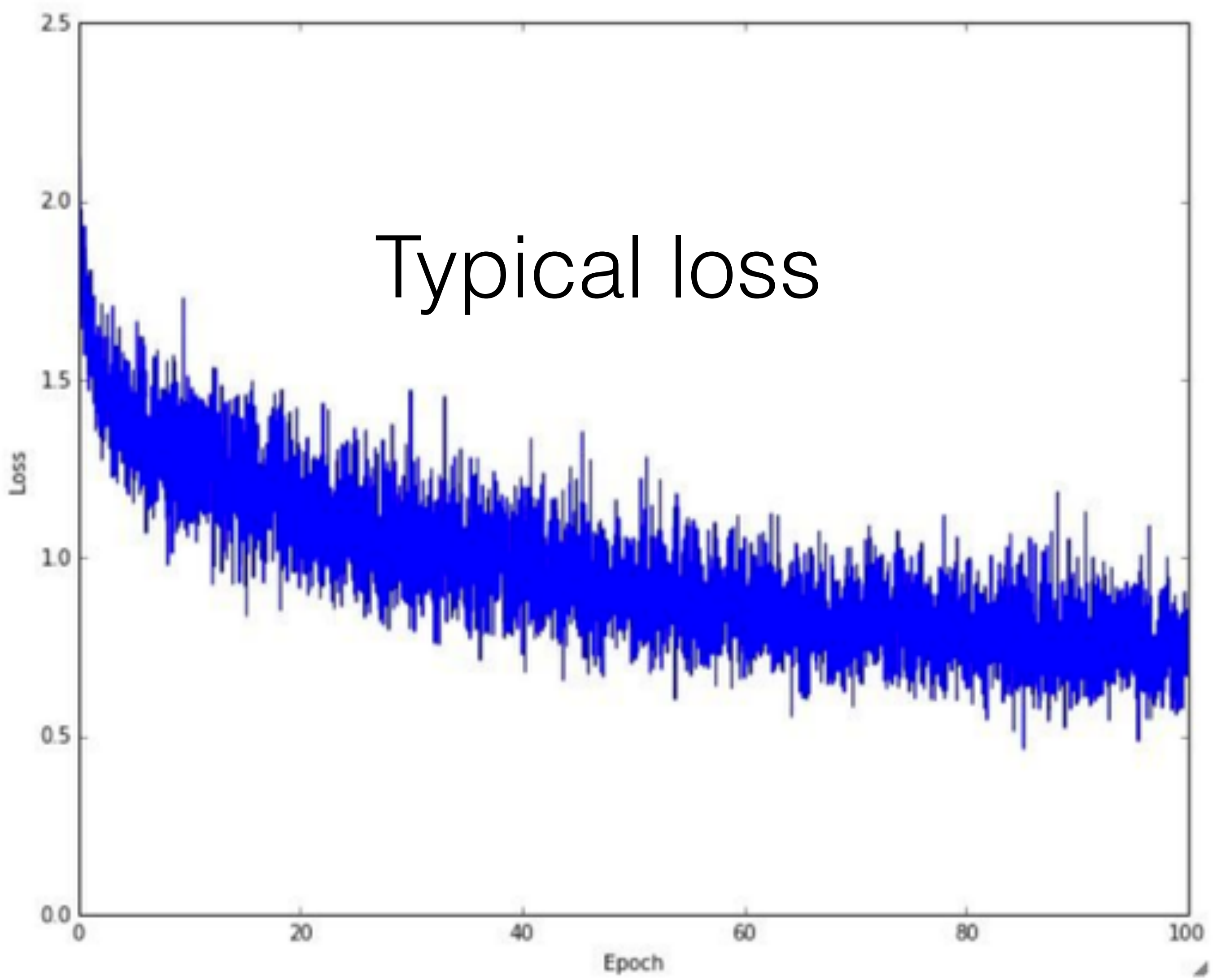
(2) Backward Propagation:

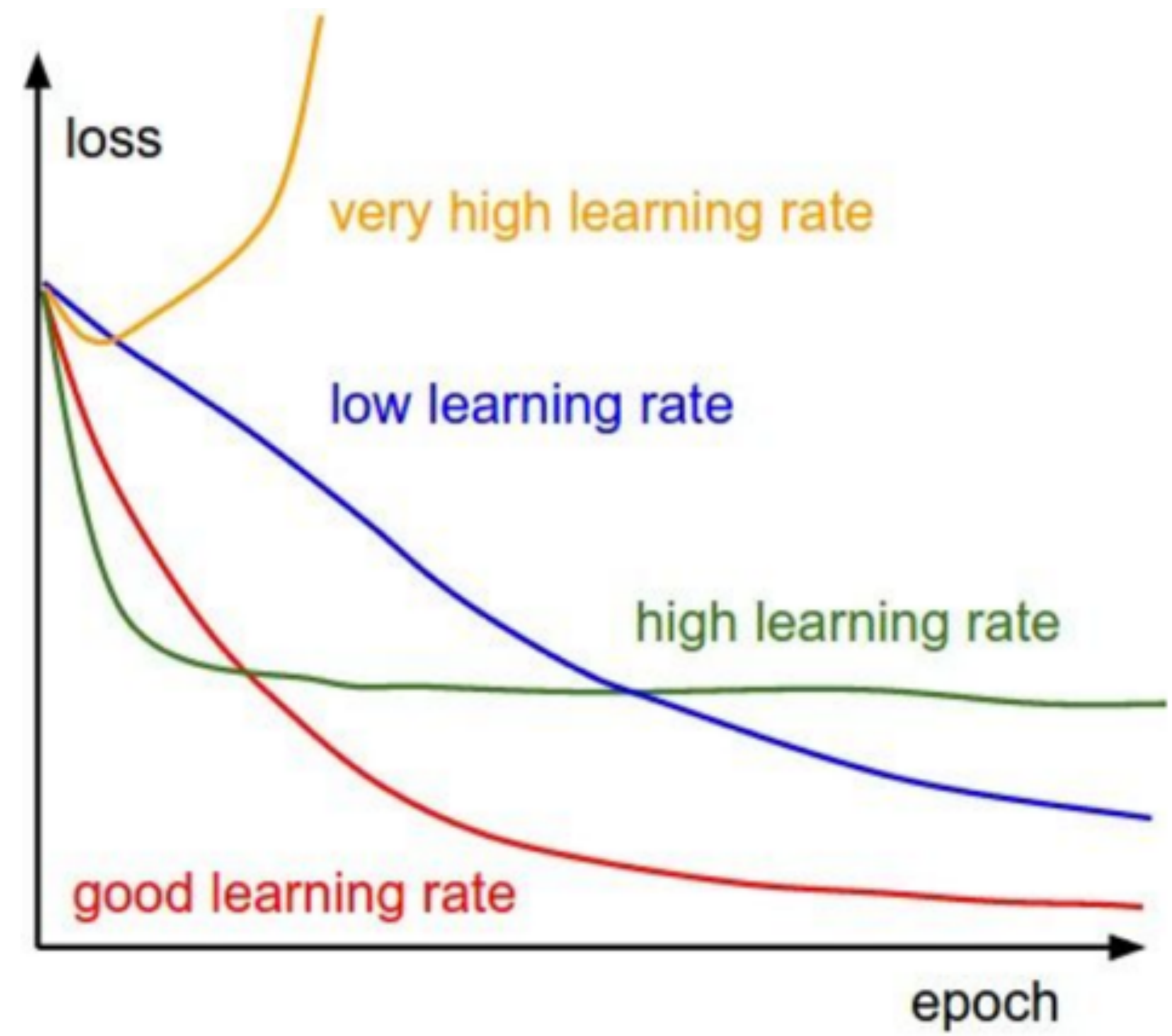
$$\frac{\partial L}{\partial x} \leftarrow \boxed{\text{Function}} \leftarrow \frac{\partial L}{\partial h} \leftarrow \cdots \leftarrow \boxed{\text{Function}} \leftarrow \frac{\partial L}{\partial s} \leftarrow L$$

(3) Weight update:

$$\theta \leftarrow \theta - \lambda \frac{\partial L}{\partial \theta}$$

(Recall) Babysitting the training process





Slide from Karpathy 2016

(Recall) Babysitting the training process

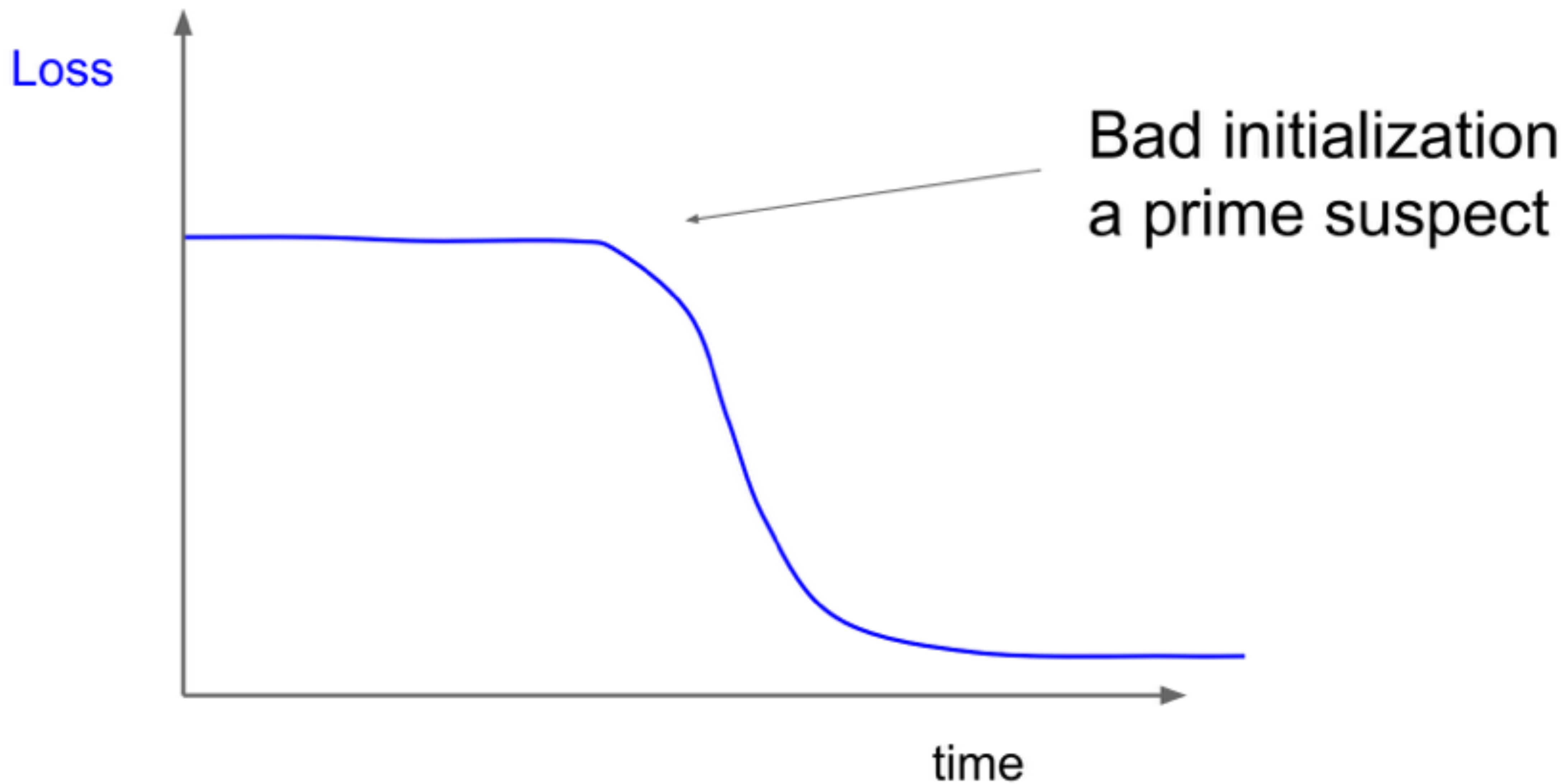


Figure: Andrej Karpathy

Weight Initialization

For deep nets, initialization is subtle and important:

Initialize weights to be smaller if there are more input connections:

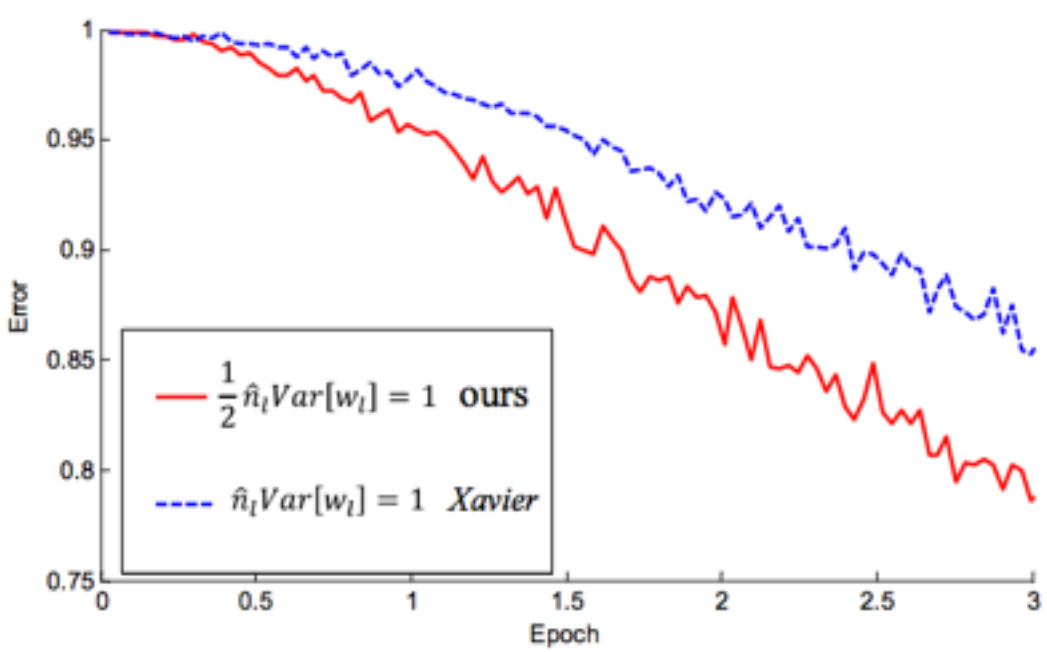
```
W = np.random.randn(n) * sqrt(2.0 / n)
```

For neural nets with ReLU, this will ensure all activations have the same variance

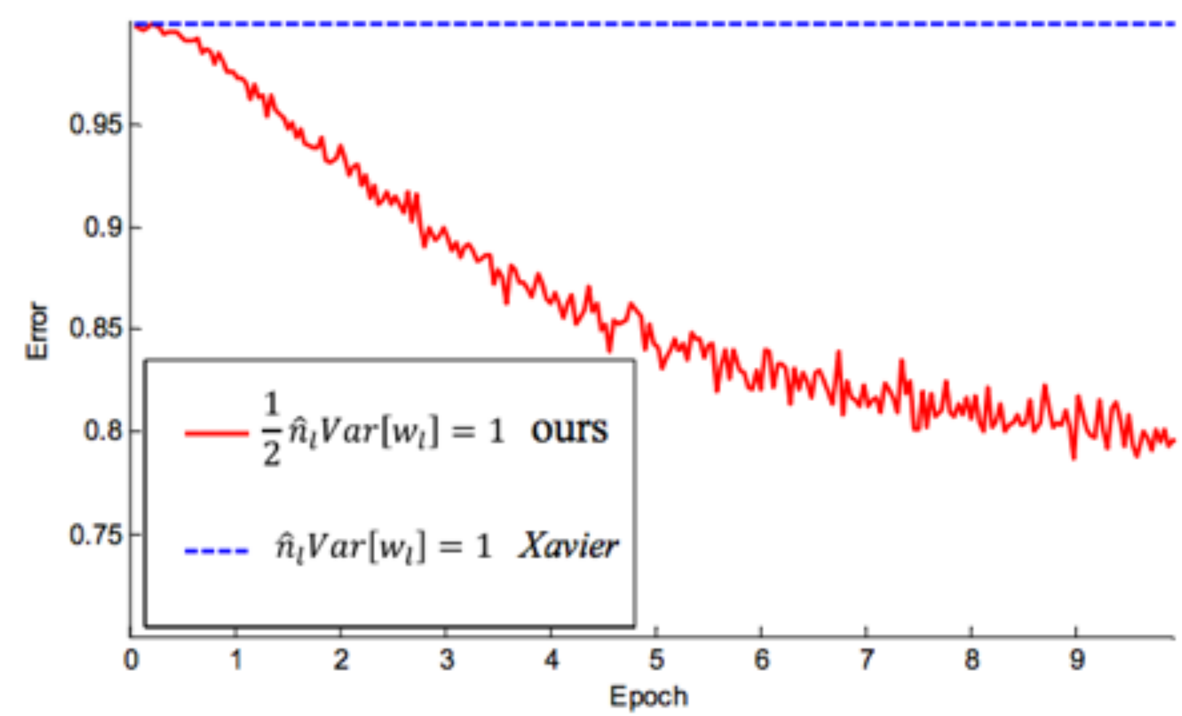
[He et al, “Delving Deep into Rectifiers:
Surpassing Human-Level Performance on ImageNet Classification”, arXiv 2015]

Initialization matters

Training can take much longer if not carefully initialized:



22 layer model



30 layer model

[He et al, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", arXiv 2015]

Proper initialization is an active area of research

Understanding the difficulty of training deep feedforward neural networks
by Glorot and Bengio, 2010

Exact solutions to the nonlinear dynamics of learning in deep linear neural networks by Saxe et al, 2013

Random walk initialization for training very deep feedforward networks by Sussillo and Abbott, 2014

Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification by He et al., 2015

Data-dependent Initializations of Convolutional Neural Networks by Krähenbühl et al., 2015

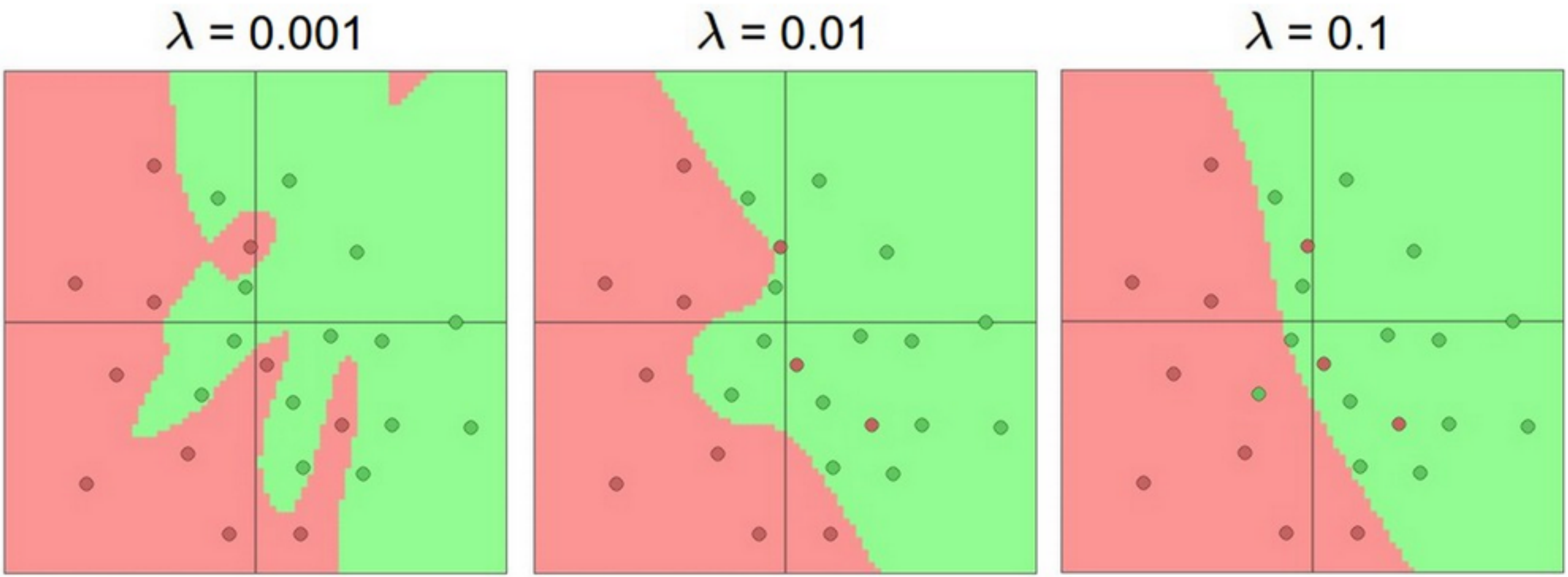
All you need is a good init, Mishkin and Matas, 2015
…

Slide: Andrej Karpathy

(Recall) Regularization reduces overfitting

$$L = L_{\text{data}} + L_{\text{reg}}$$

$$L_{\text{reg}} = \lambda \frac{1}{2} \|W\|_2^2$$



[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]

Example Regularizers

L2 regularization

$$L_{\text{reg}} = \lambda \frac{1}{2} \left\| W \right\|_2^2$$

(L2 regularization encourages small weights)

L1 regularization

$$L_{\text{reg}} = \lambda \left\| W \right\|_1 = \lambda \sum_{ij} \left| W_{ij} \right|$$

(L1 regularization encourages sparse weights: weights are encouraged to reduce to exactly zero)

"Elastic net"

$$L_{\text{reg}} = \lambda_1 \left\| W \right\|_1 + \lambda_2 \left\| W \right\|_2^2$$

(combine L1 and L2 regularization)

Max norm

Clamp weights to some max norm

$$\left\| W \right\|_2^2 \leq c$$

"Weight decay"

Regularization is also called "weight decay" because the weights "decay" each iteration:

$$L_{\mathrm{reg}} = \lambda \frac{1}{2} \|W\|_2^2 \longrightarrow \frac{\partial L}{\partial W} = \lambda W$$

Gradient descent step:

$$W \leftarrow W - \alpha\lambda W - \frac{\partial L_{\mathrm{data}}}{\partial W}$$

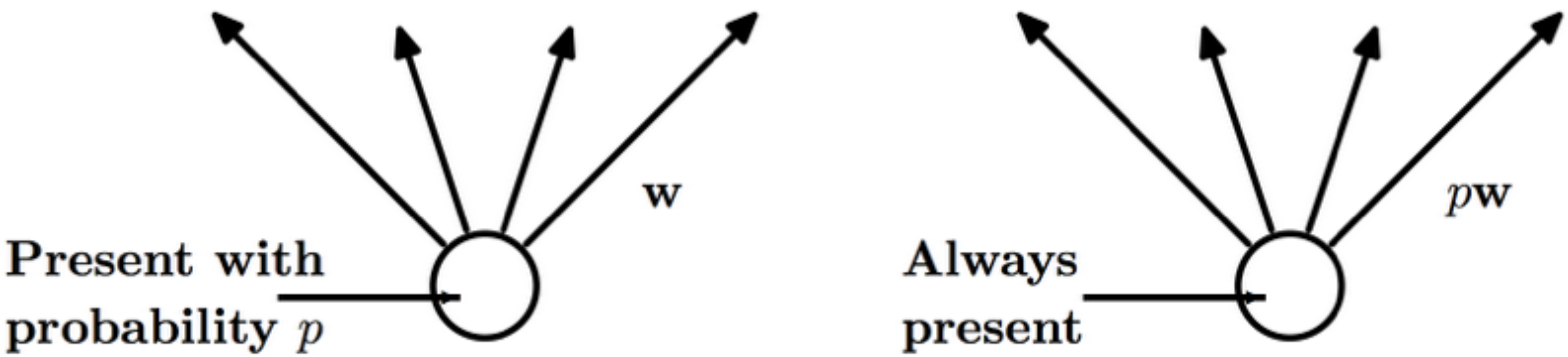
Weight decay: $\alpha\lambda$ (weights always decay by this amount)

Note: biases are sometimes excluded from regularization

[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]

Dropout

Simple but powerful technique to reduce overfitting:



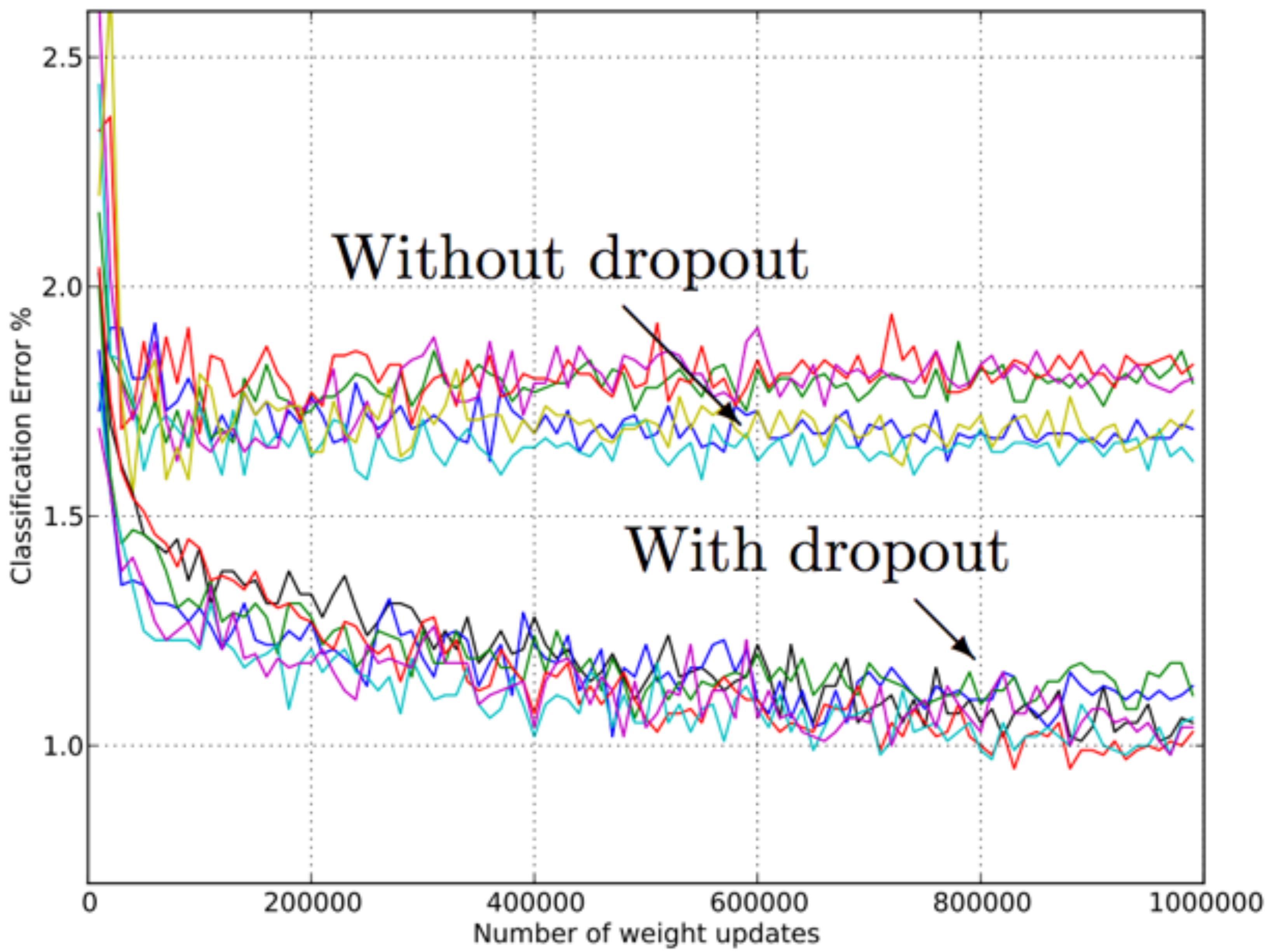
(a) At training time

(b) At test time

[Srivasta et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014]

Dropout

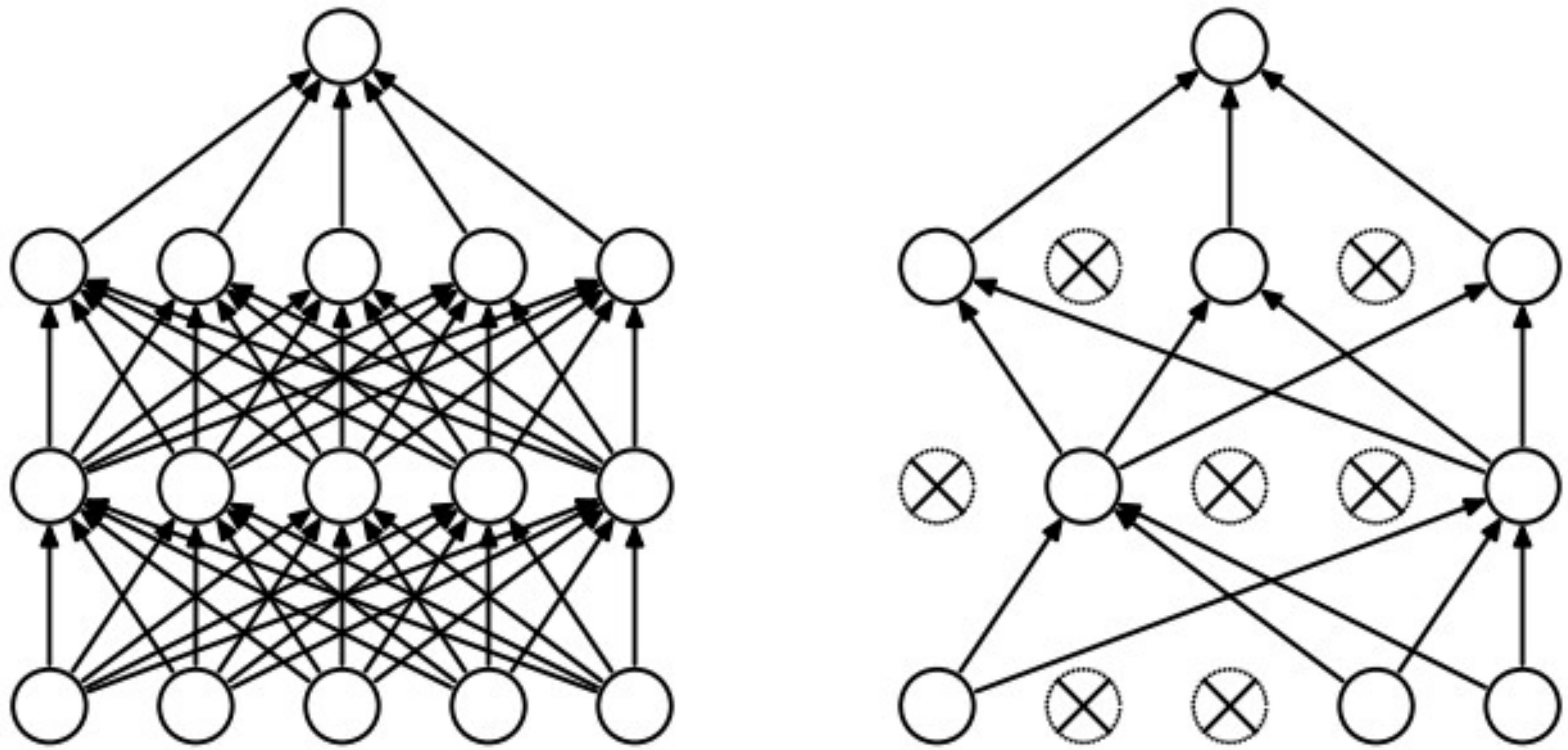
Simple but powerful technique to reduce overfitting:



[Srivasta et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014]

Dropout

Simple but powerful technique to reduce overfitting:



(a) Standard Neural Net

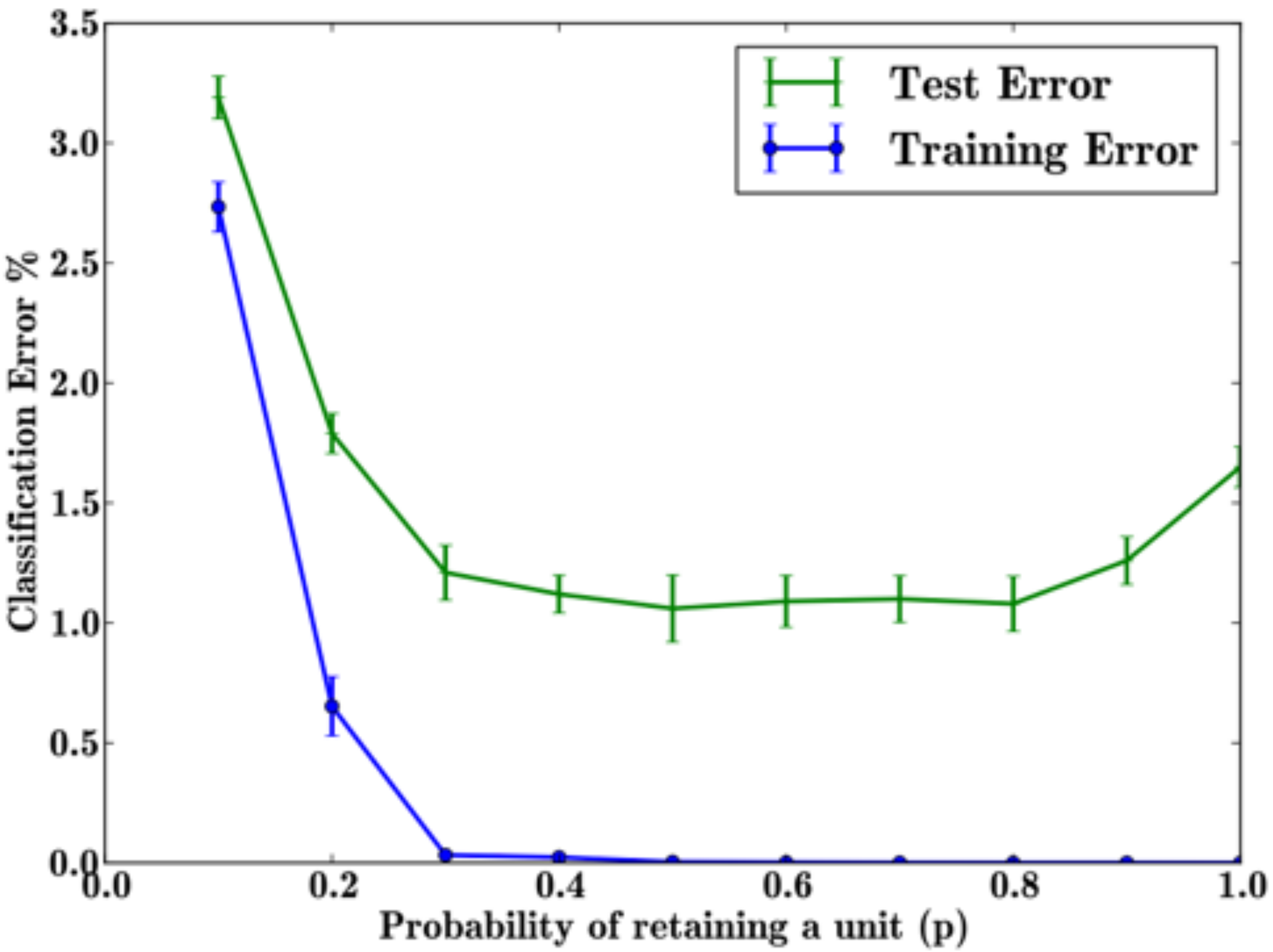
(b) After applying dropout.

Note: Dropout can be interpreted as an approximation to taking the geometric mean of an ensemble of exponentially many models

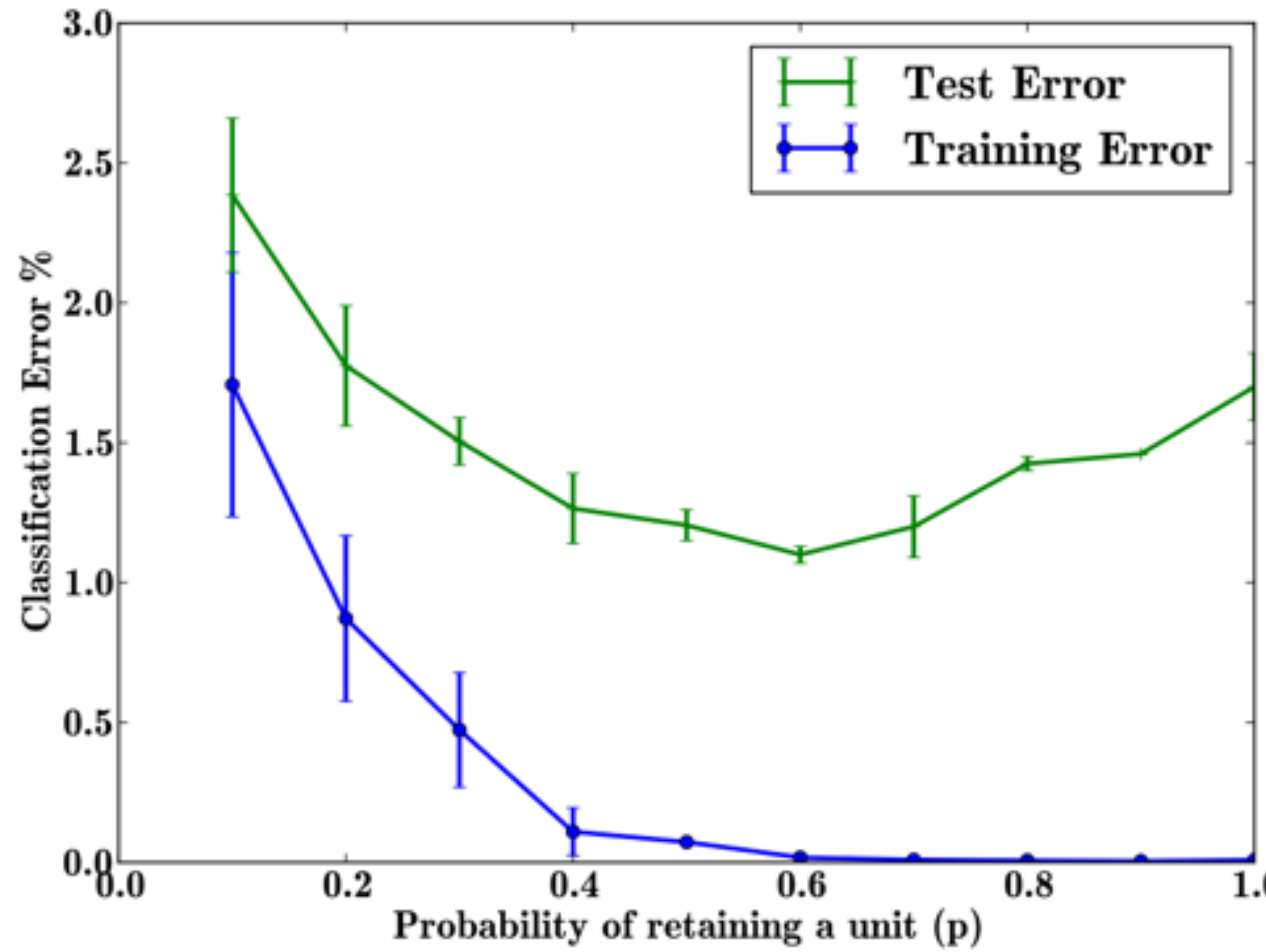
[Srivasta et al, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, JMLR 2014]

Dropout

How much dropout? Around p = 0.5



(a) Keeping n fixed.



(b) Keeping pn fixed.

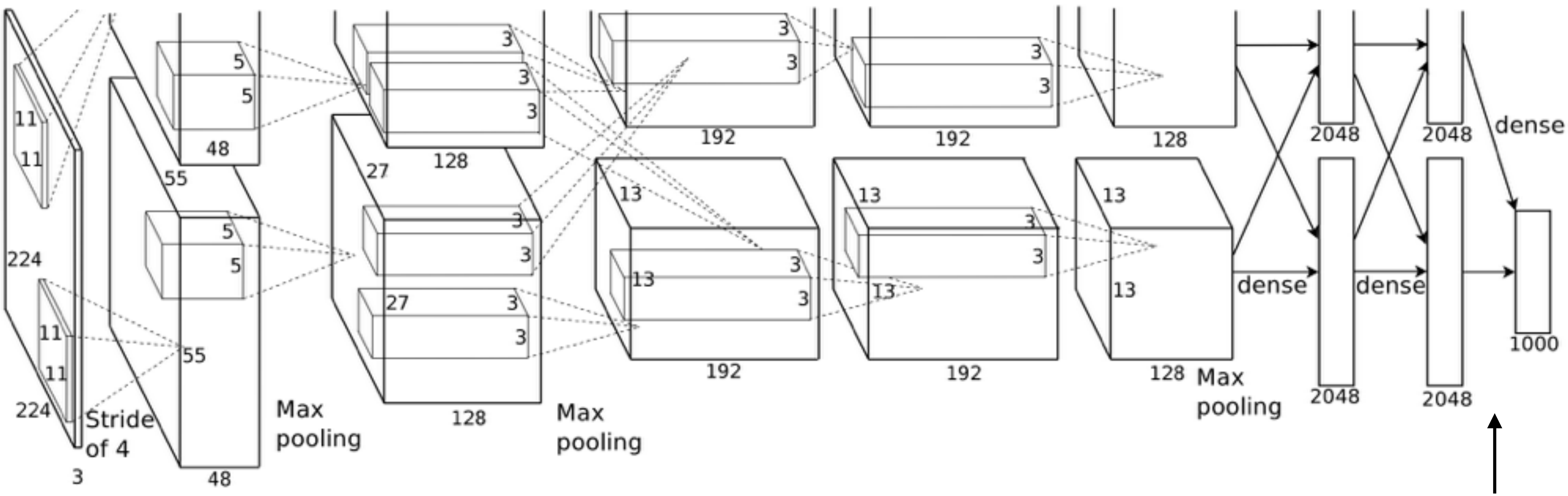
[Srivasta et al, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, JMLR 2014]

Dropout

Case study: [Krizhevsky 2012]

"Without dropout, our network exhibits substantial overfitting."

Dropout here



But not here — why?

[Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012]

Dropout

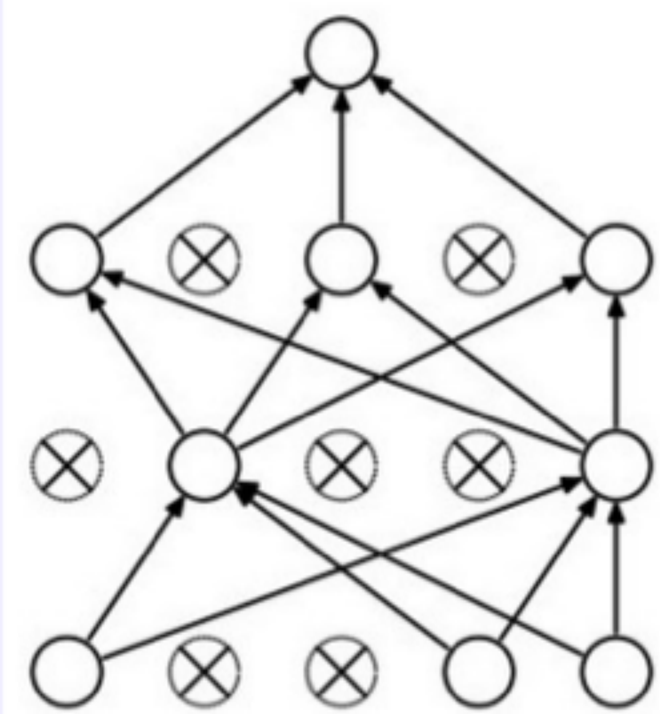
```
p = 0.5 # probability of keeping a unit active. higher = less dropout

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



(note, here X is a single input)

Figure: Andrej Karpathy

Dropout

Test time: scale the activations

Expected value of a neuron *h* with dropout:

$$E[h] = ph + (1-p)0 = ph$$

```
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
```

We want to keep the same expected value

Figure: Andrej Karpathy

Batch Normalization

[Ioffe and Szegedy, 2015]

"you want unit gaussian activations? just make them so."

consider a batch of activations at some layer.
To make each dimension unit gaussian, apply:

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

this is a vanilla differentiable function...

And then allow the network to squash the range if it wants to:

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

Slide: Andrej Karpathy

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

Slide: Andrej Karpathy

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

Note: at test time BatchNorm layer functions differently:

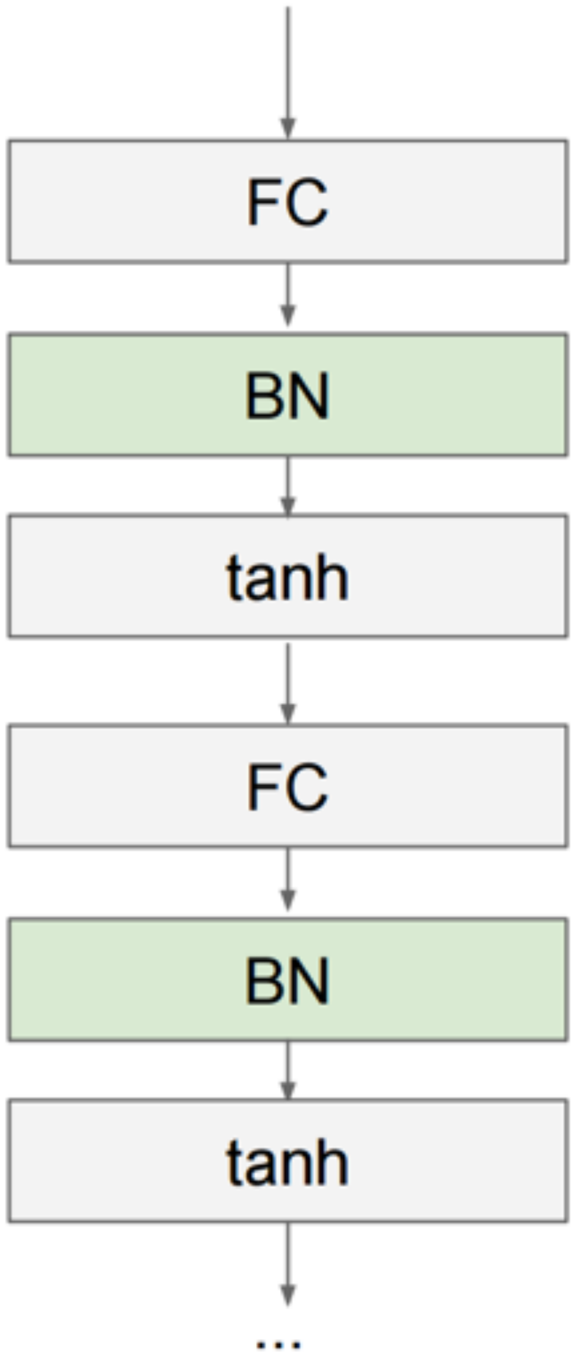
The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

(e.g. can be estimated during training with running averages)

Slide: Andrej Karpathy

Batch Normalization

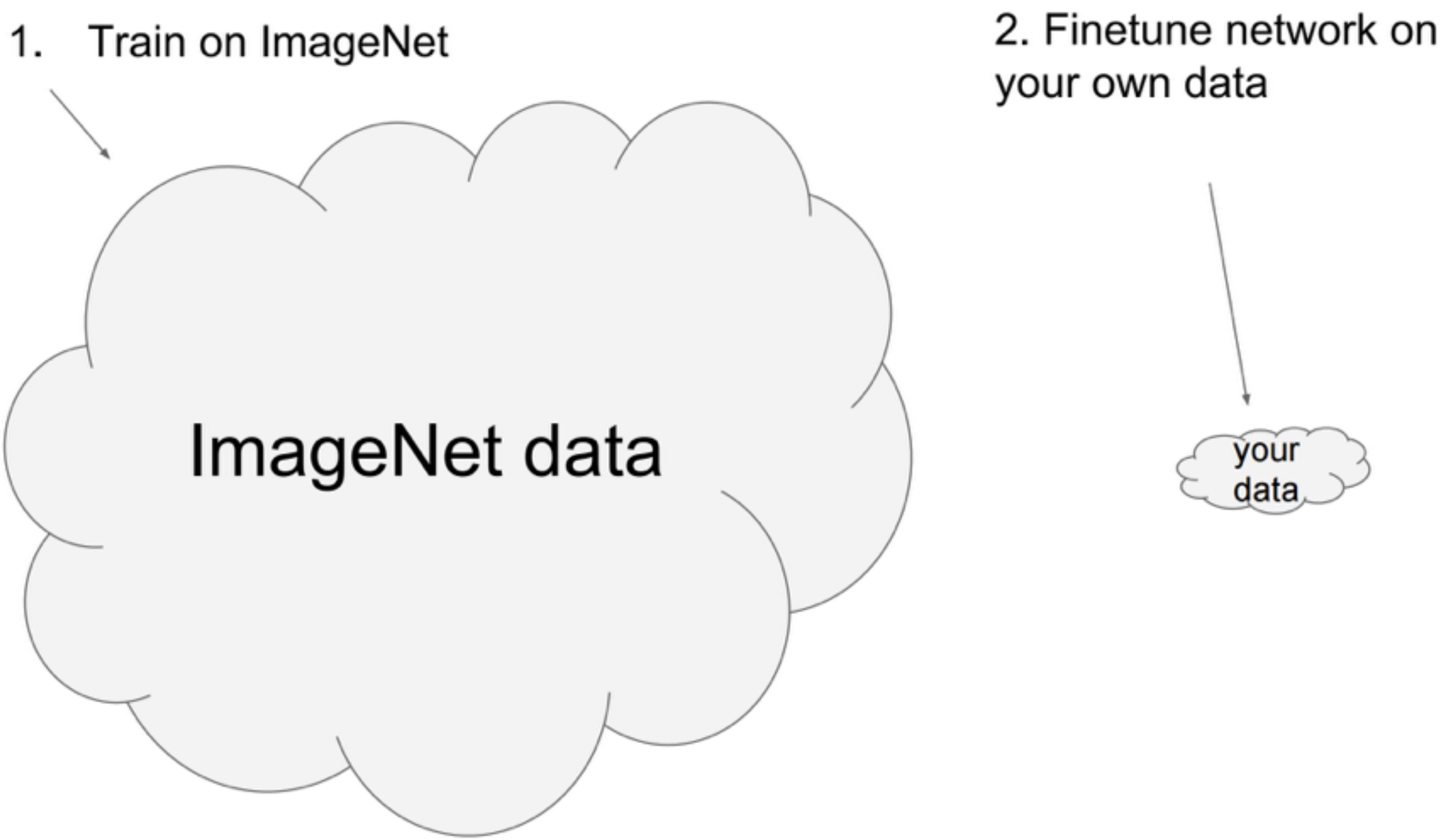
[Ioffe and Szegedy, 2015]



Place *after* a FC or Convolutional layer, and *before* nonlinearity

Slide: Andrej Karpathy

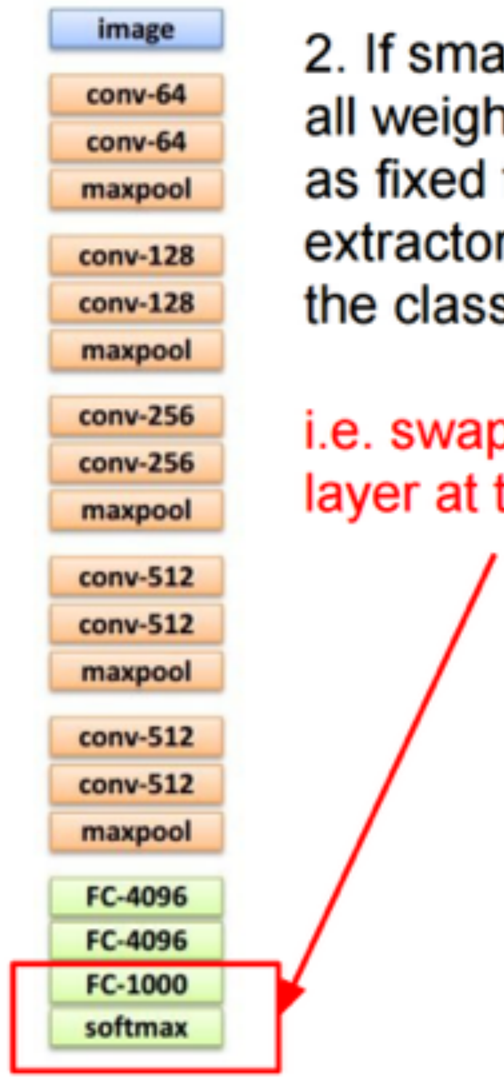
Transfer Learning ("fine-tuning")



Slide: Andrej Karpathy

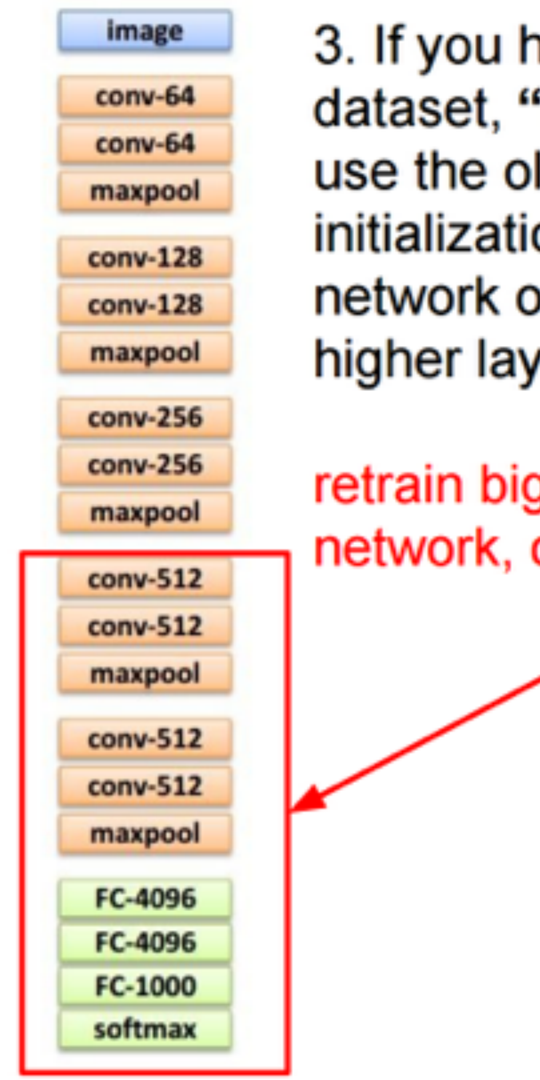
Transfer Learning (“fine-tuning”)

1. Train on ImageNet



2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end



3. If you have medium sized dataset, **“finetune”** instead: use the old weights as initialization, train the full network or only some of the higher layers

retrain bigger portion of the network, or even all of it.

This is not just a special trick;
this is “the” method used by most papers

Slide: Andrej Karpathy

Transfer Learning ("fine-tuning")

E.g. Caffe Model Zoo: Lots of pretrained ConvNets

https://github.com/BVLC/caffe/wiki/Model-Zoo

...

Slide: Andrej Karpathy

Summary

- Preprocess the data (subtract mean, sub-crops)
- Initialize weights carefully
- Use Dropout and/or Batch Normalization
- Use SGD + Momentum
- Fine-tune from ImageNet
- Babysit the network as it trains

You are now ready.

Slide: Andrej Karpathy

You are now ready.

Slide: Andrej Karpathy

You are now ready.

Slide: Andrej Karpathy