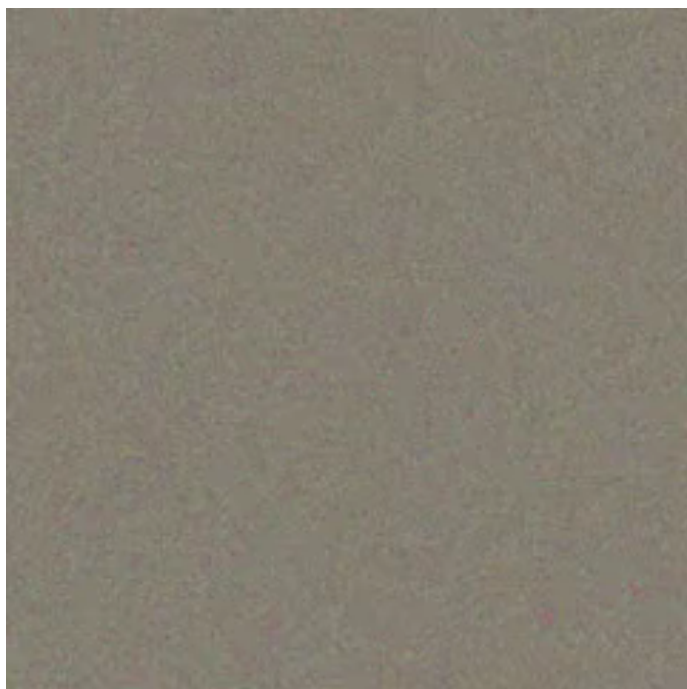


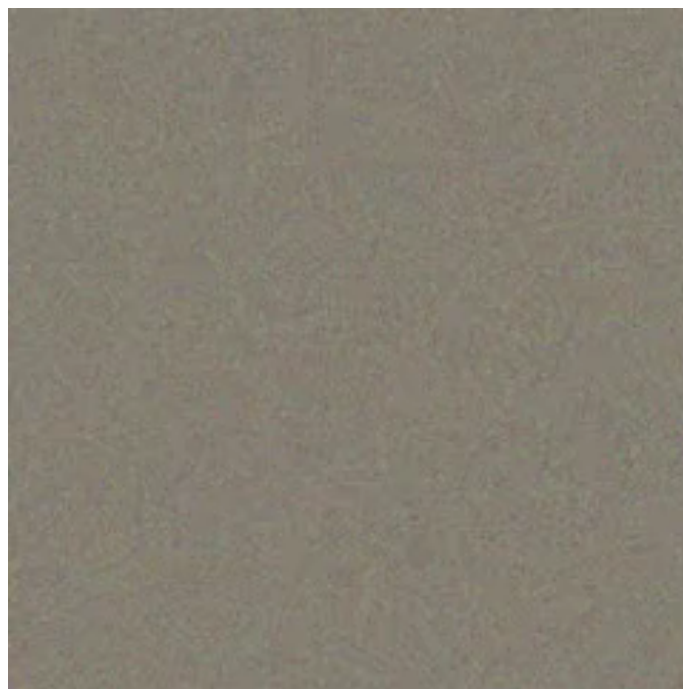
Lecture 39: Training Neural Networks (Cont'd)

CS 4670/5670

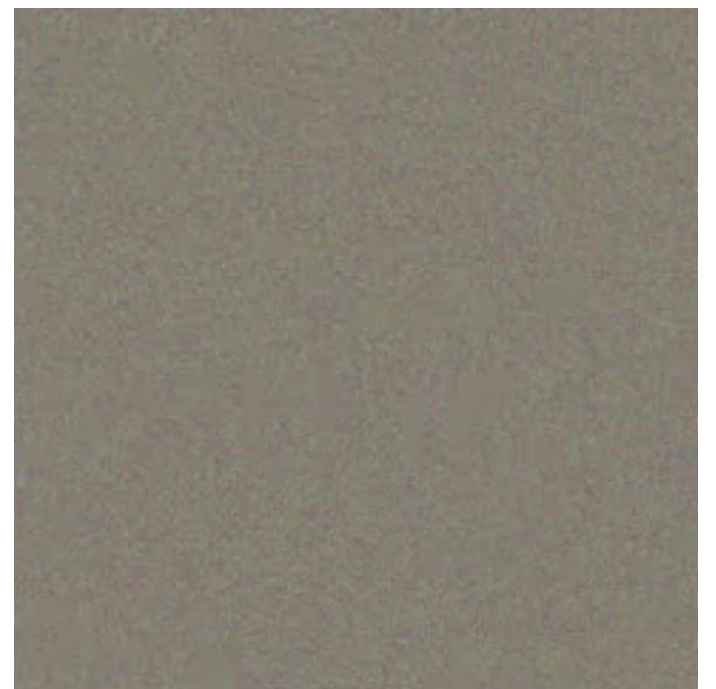
Sean Bell



Strawberry

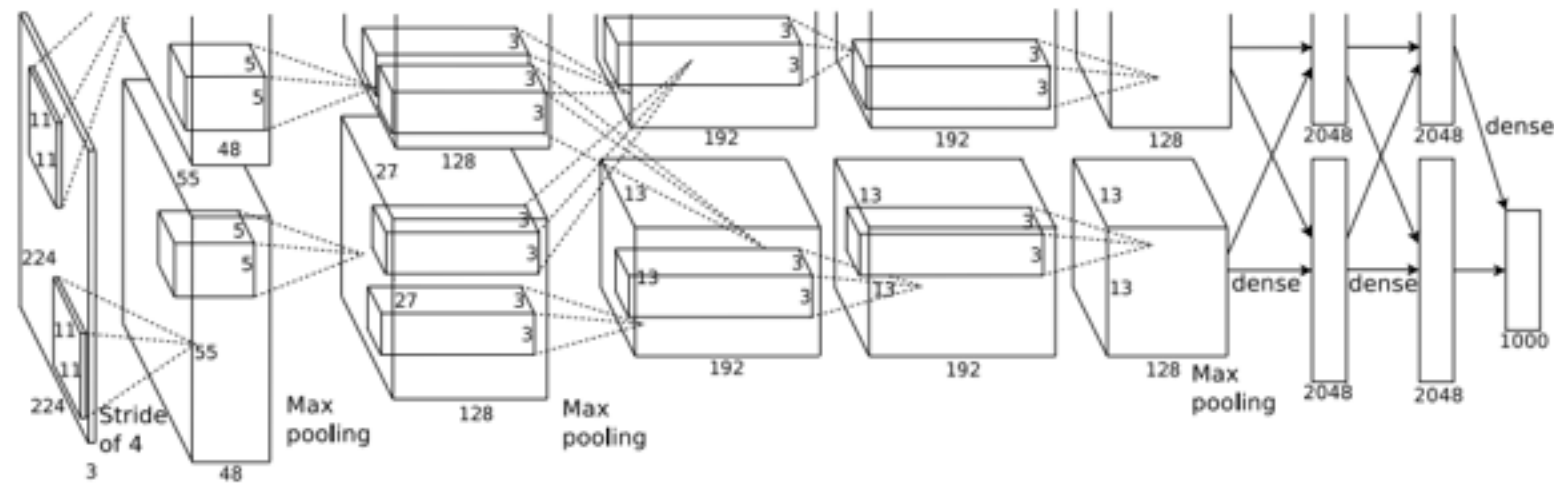
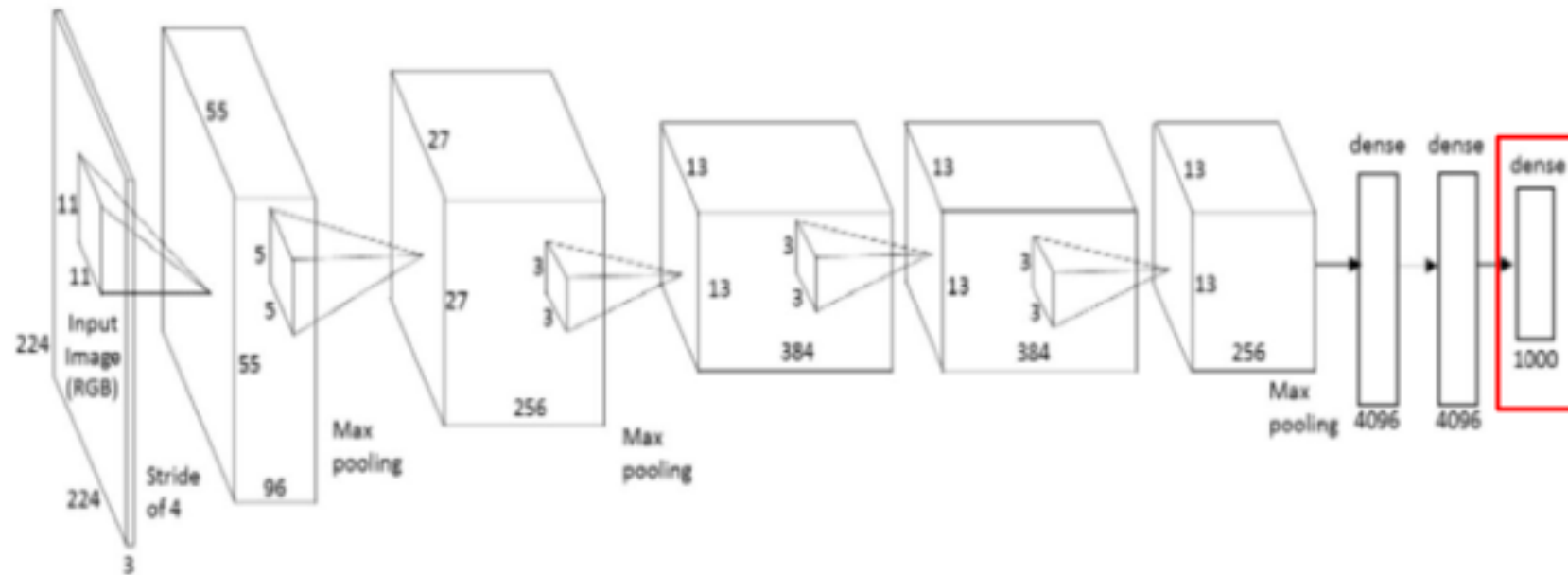


Goblet



Throne

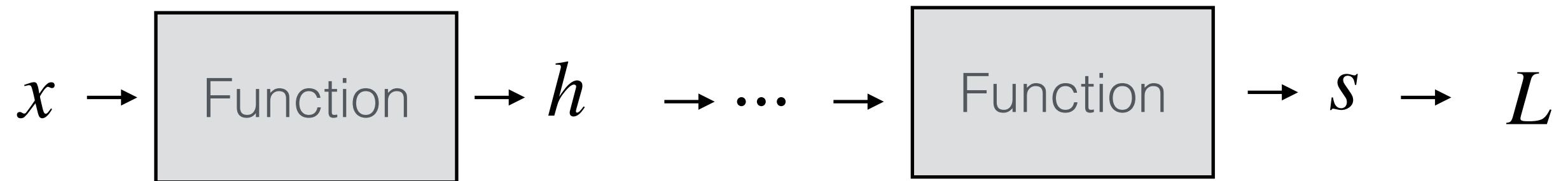
(Side Note for PA5) AlexNet: 1 vs 2 parts



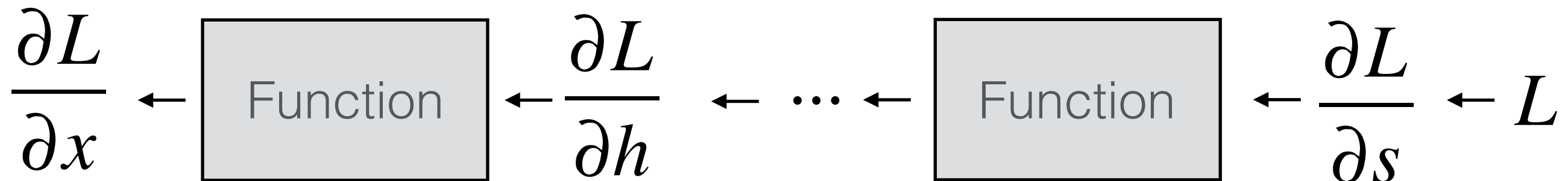
Caffe represents caffe like the above image, but computes as if it were the bottom image using 2 “groups”

(Recall) Each iteration of training

(1) Forward Propagation:

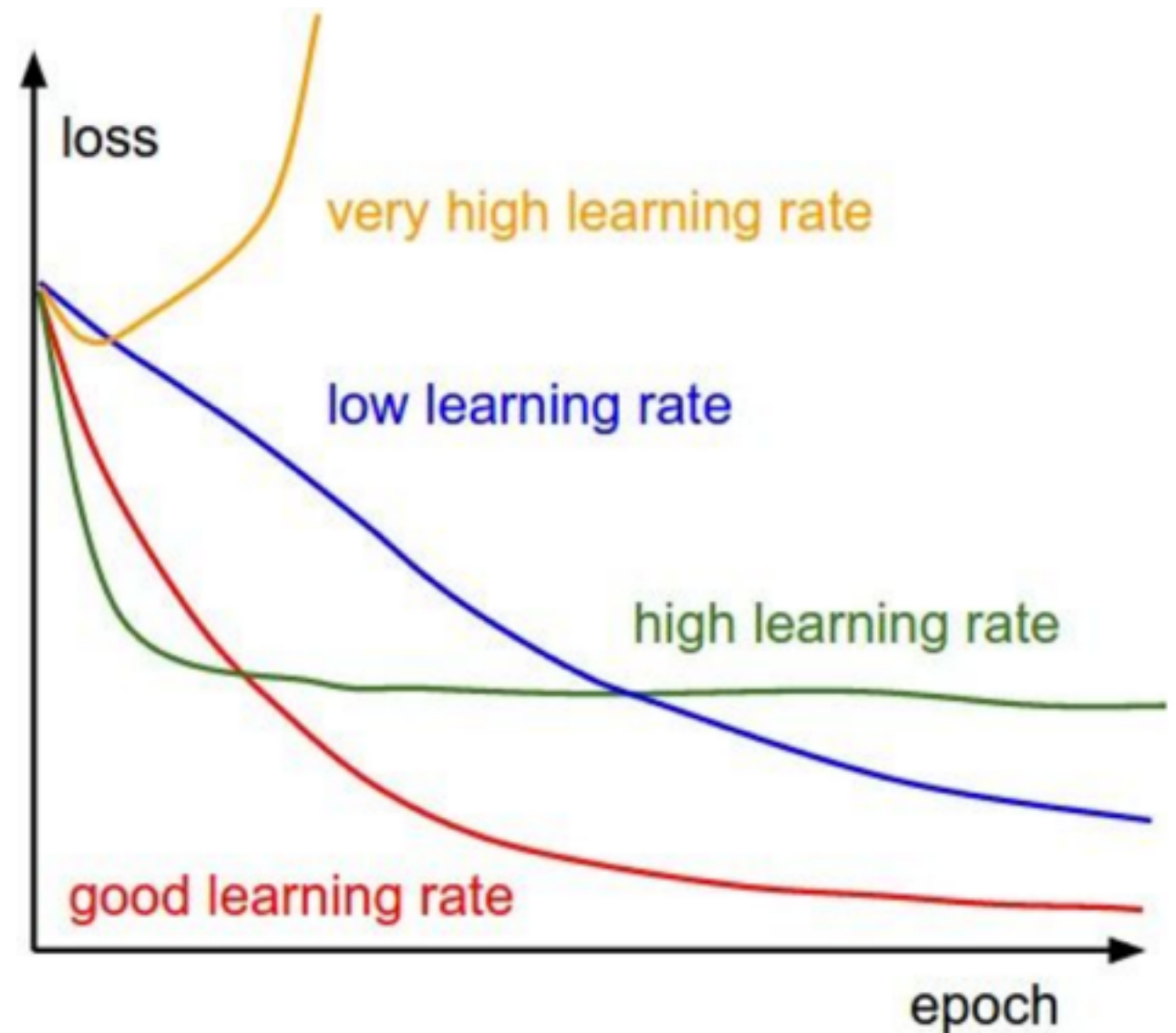
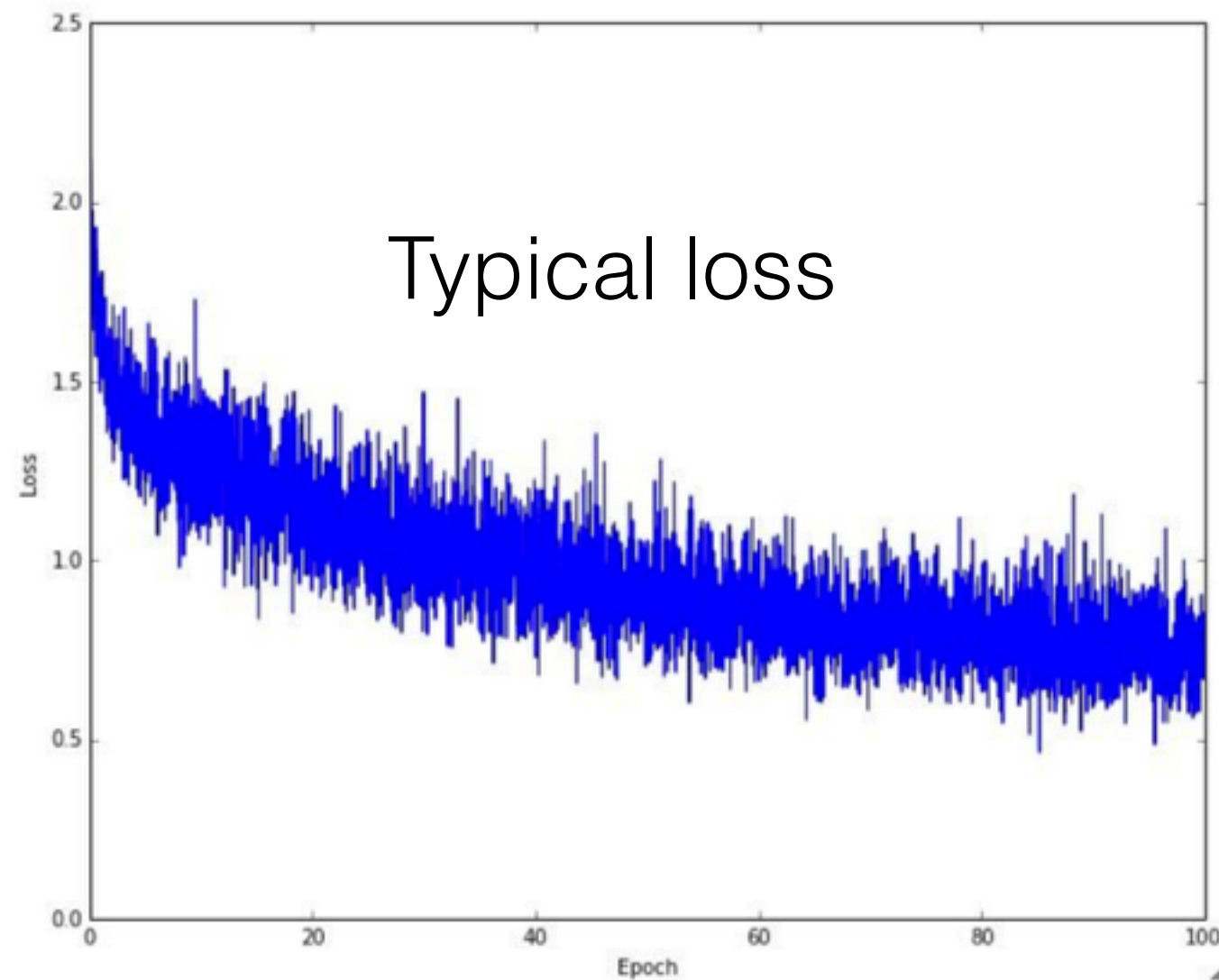


(2) Backward Propagation:



(3) Weight update: $\theta \leftarrow \theta - \lambda \frac{\partial L}{\partial \theta}$

(Recall) Babysitting the training process



(Recall) Babysitting the training process

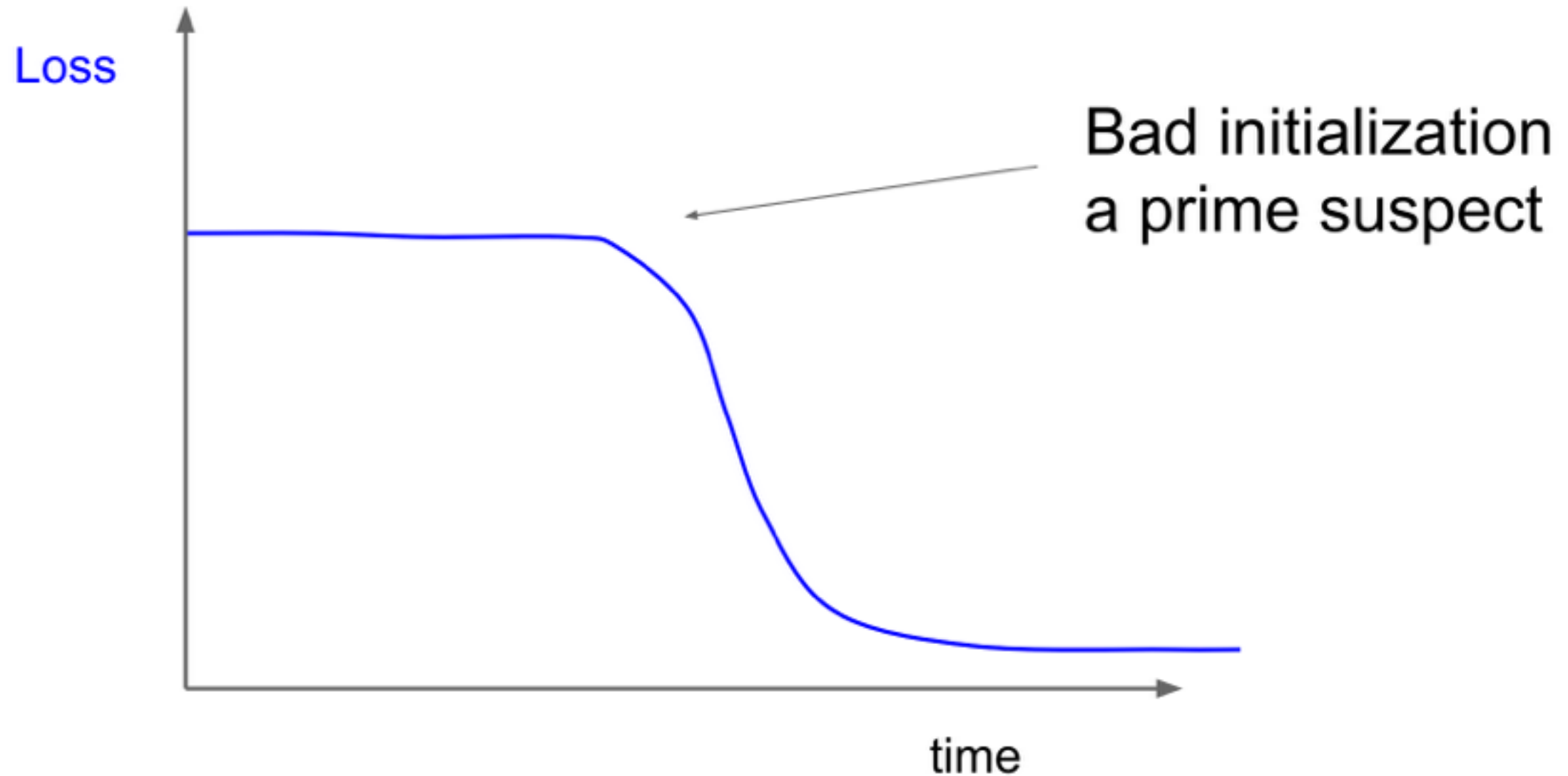
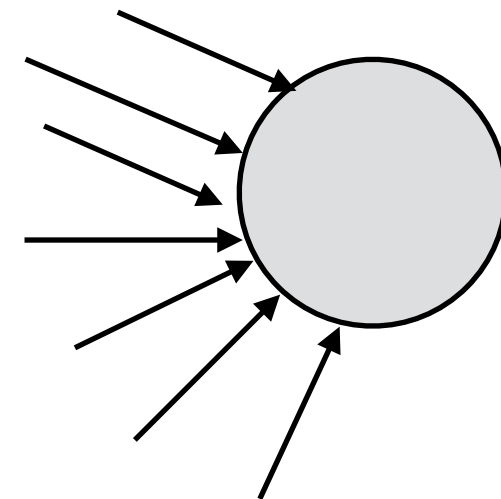
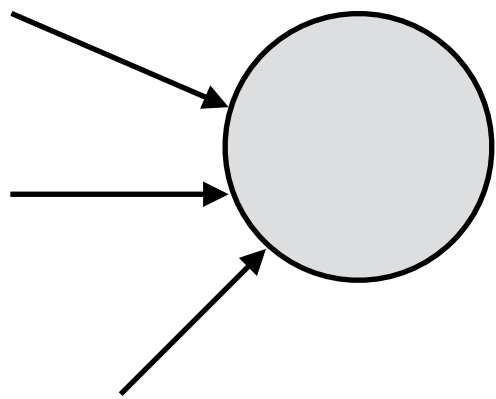


Figure: Andrej Karpathy

Weight Initialization

For deep nets, initialization is subtle and important:



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

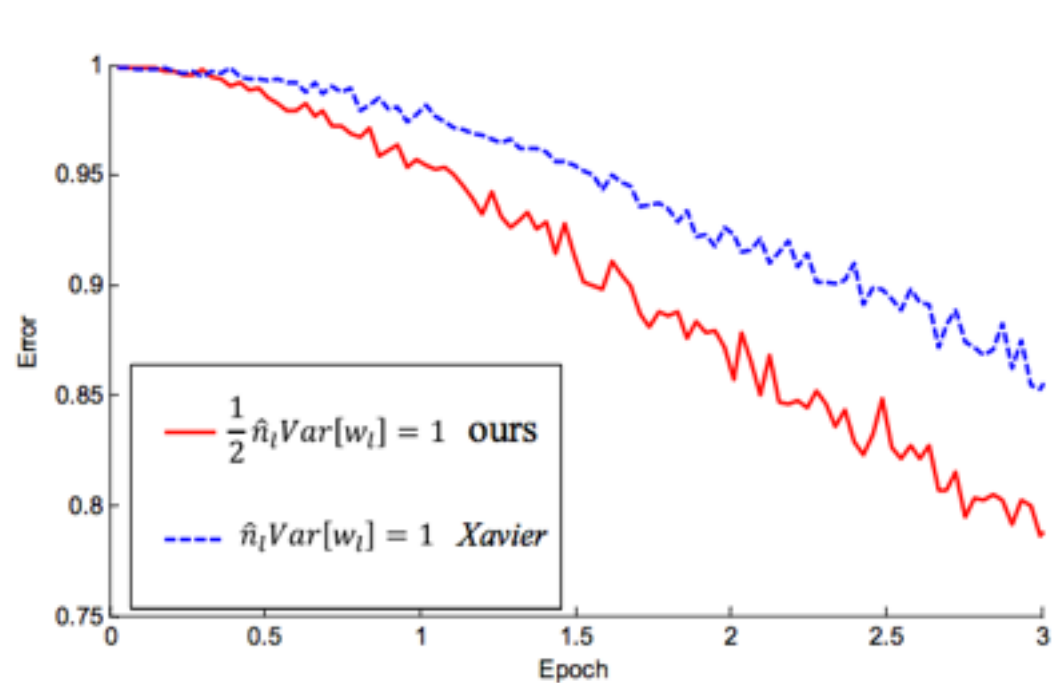
```
W = np.random.randn(n) * sqrt(2.0 / n)
```

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

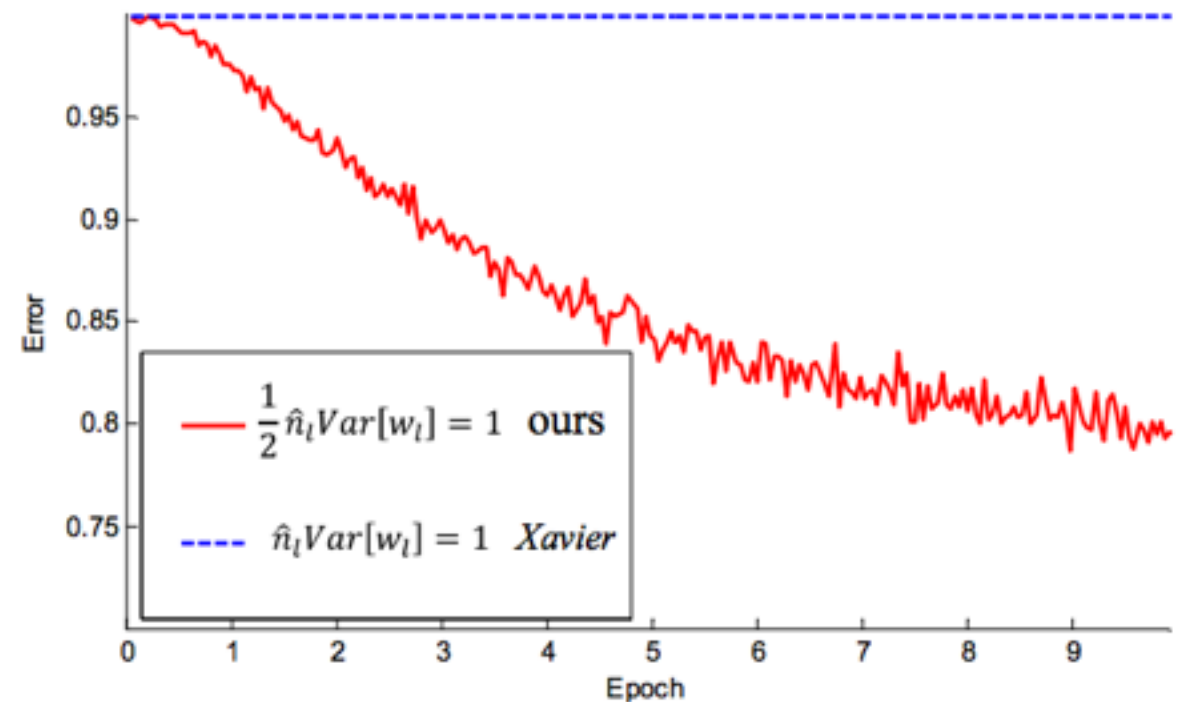
[He et al, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”, arXiv 2015]

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

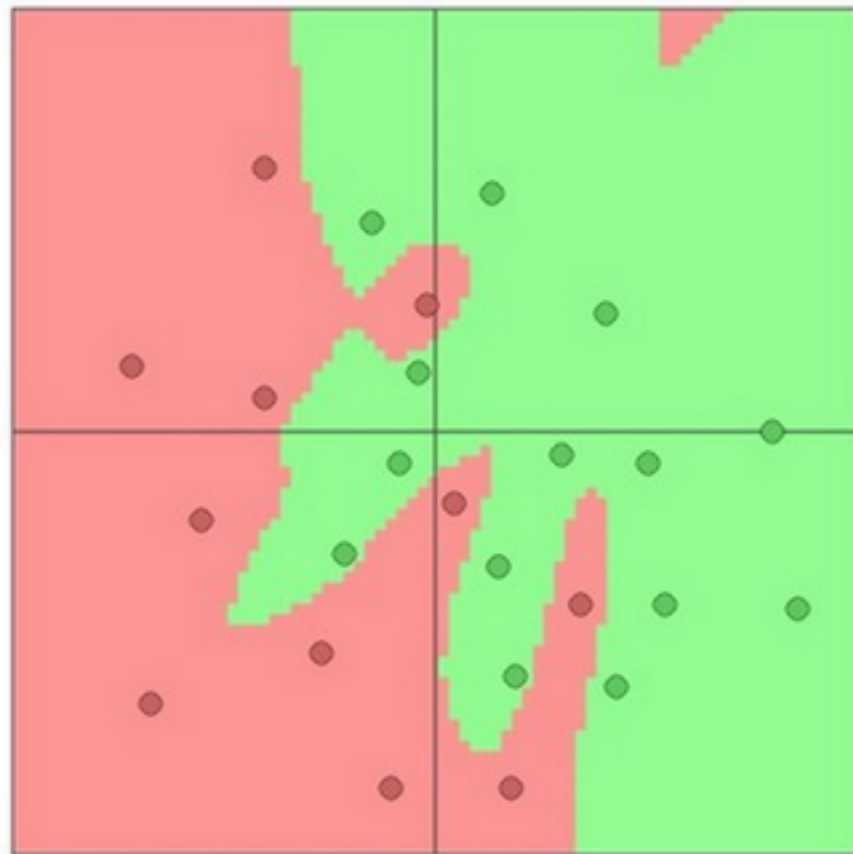
All you need is a good init, Mishkin and Matas, 2015

...

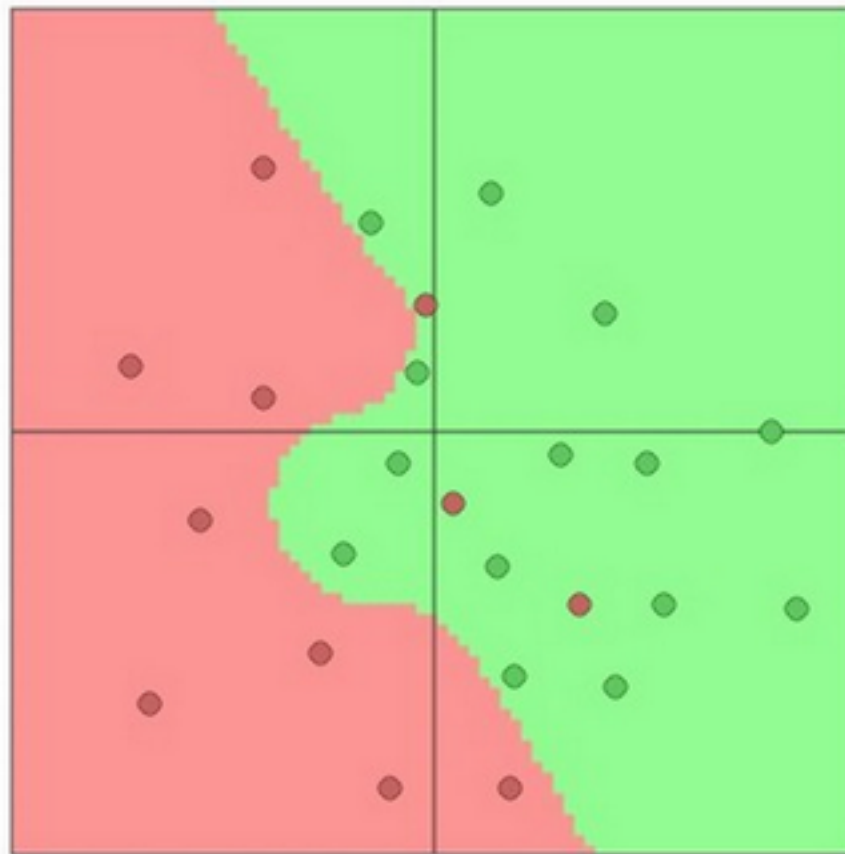
(Recall) Regularization reduces overfitting

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

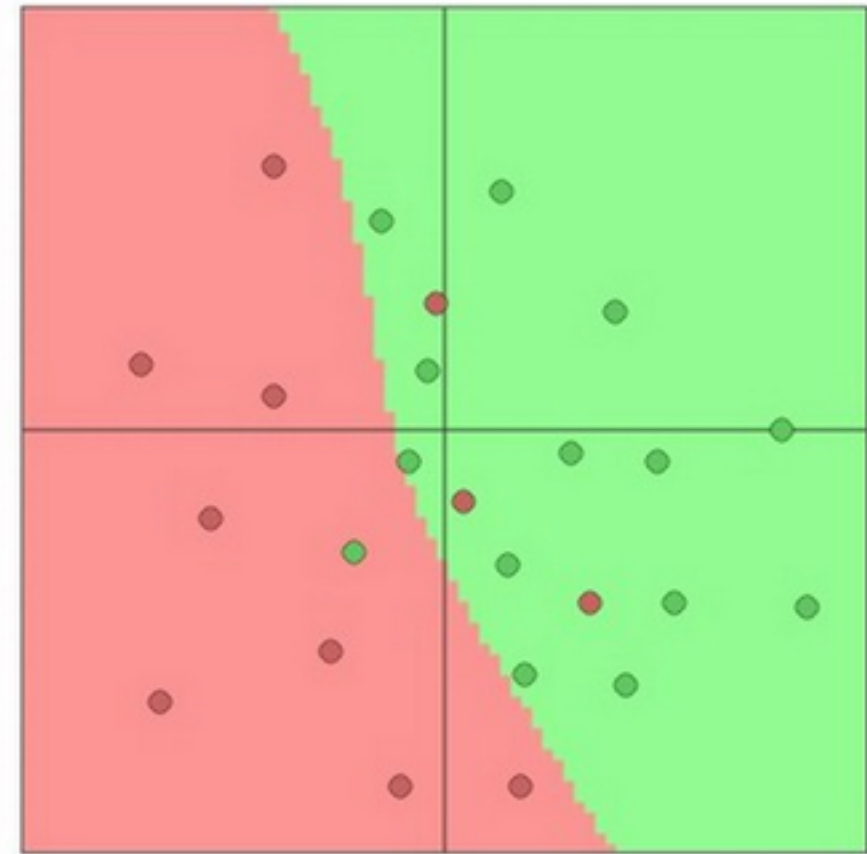
$\lambda = 0.001$



$\lambda = 0.01$



$\lambda = 0.1$



Example Regularizers

L2 regularization

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

(L2 regularization encourages small weights)

L1 regularization

$$L_{\text{reg}} = \lambda \|W\|_1 = \lambda \sum_{ij} |w_{ij}|$$

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

“Elastic net”

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

(combine L1 and L2 regularization)

Max norm

Clamp weights to some max norm

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

“Weight decay”

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

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

Gradient descent step:

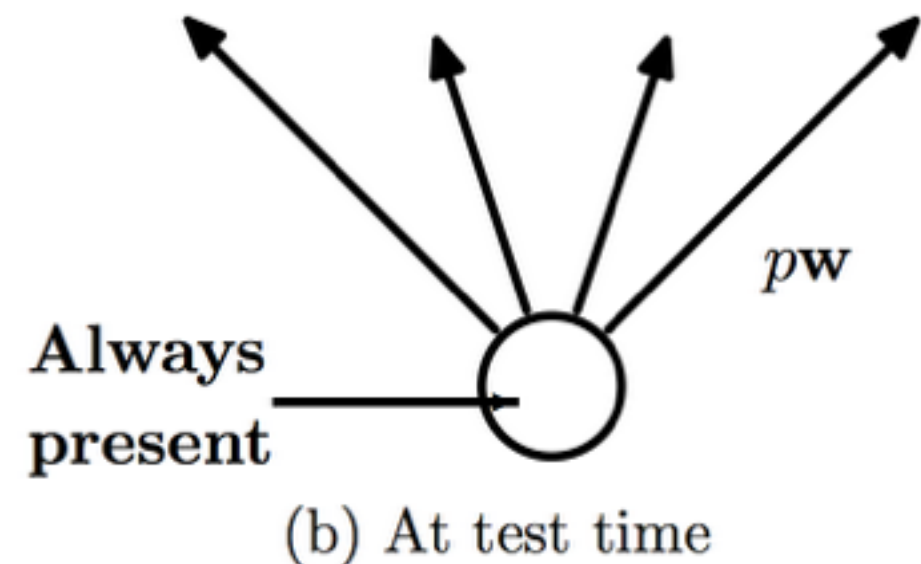
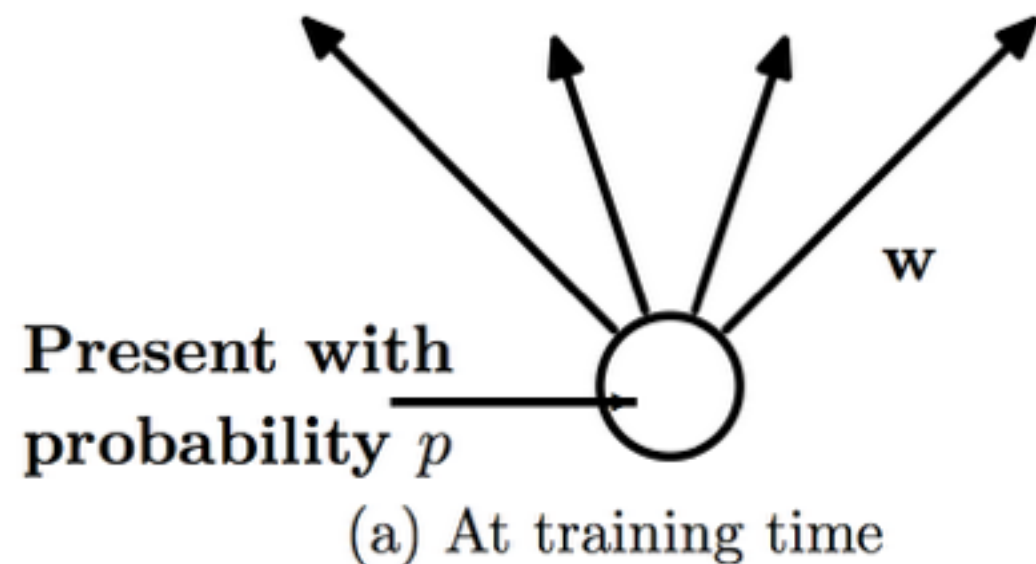
$$W \leftarrow W - \alpha \lambda W - \frac{\partial L_{\text{data}}}{\partial W}$$

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

Note: biases are sometimes excluded from regularization

Dropout

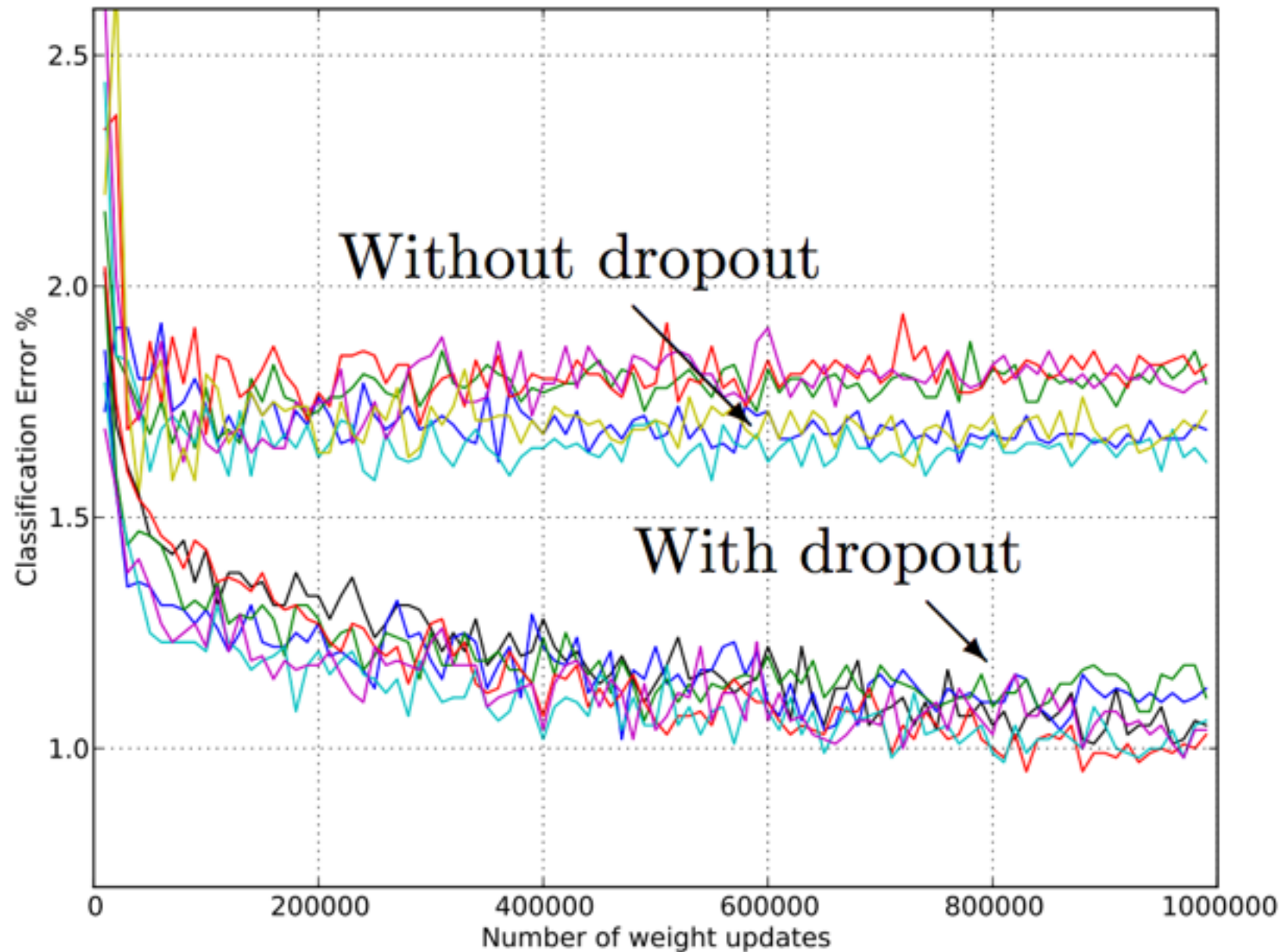
Simple but powerful technique to reduce overfitting:



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

Dropout

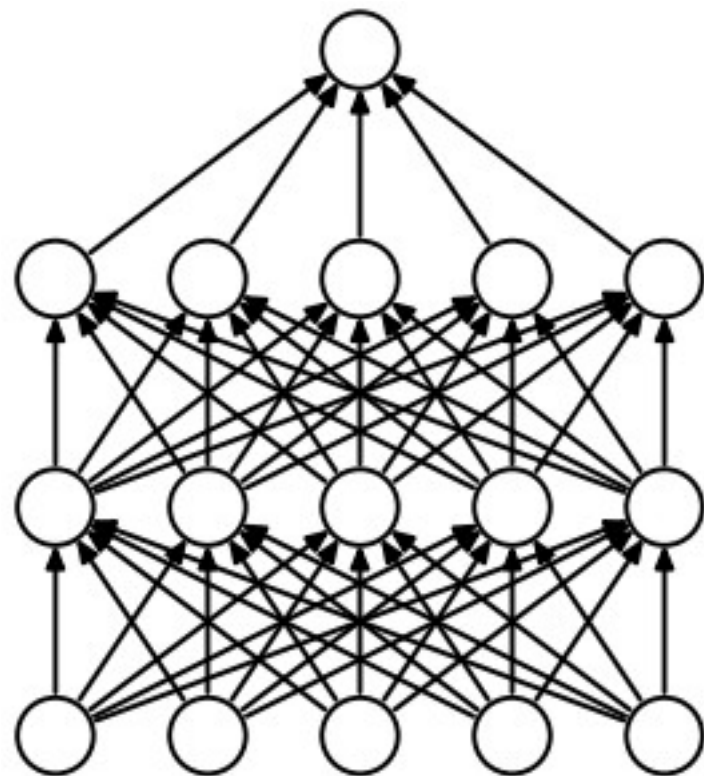
Simple but powerful technique to reduce overfitting:



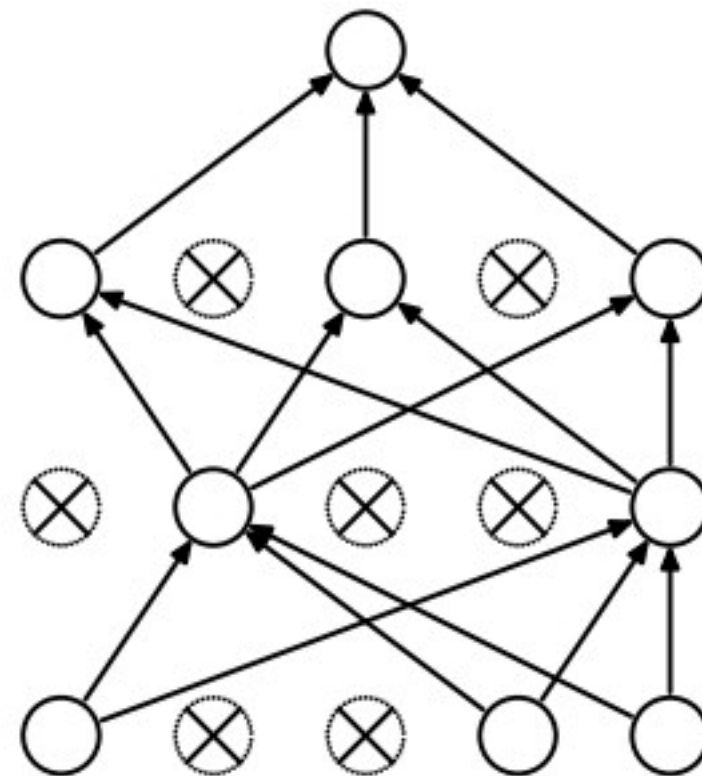
[Srivasta et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014]

Dropout

Simple but powerful technique to reduce overfitting:



(a) Standard Neural Net



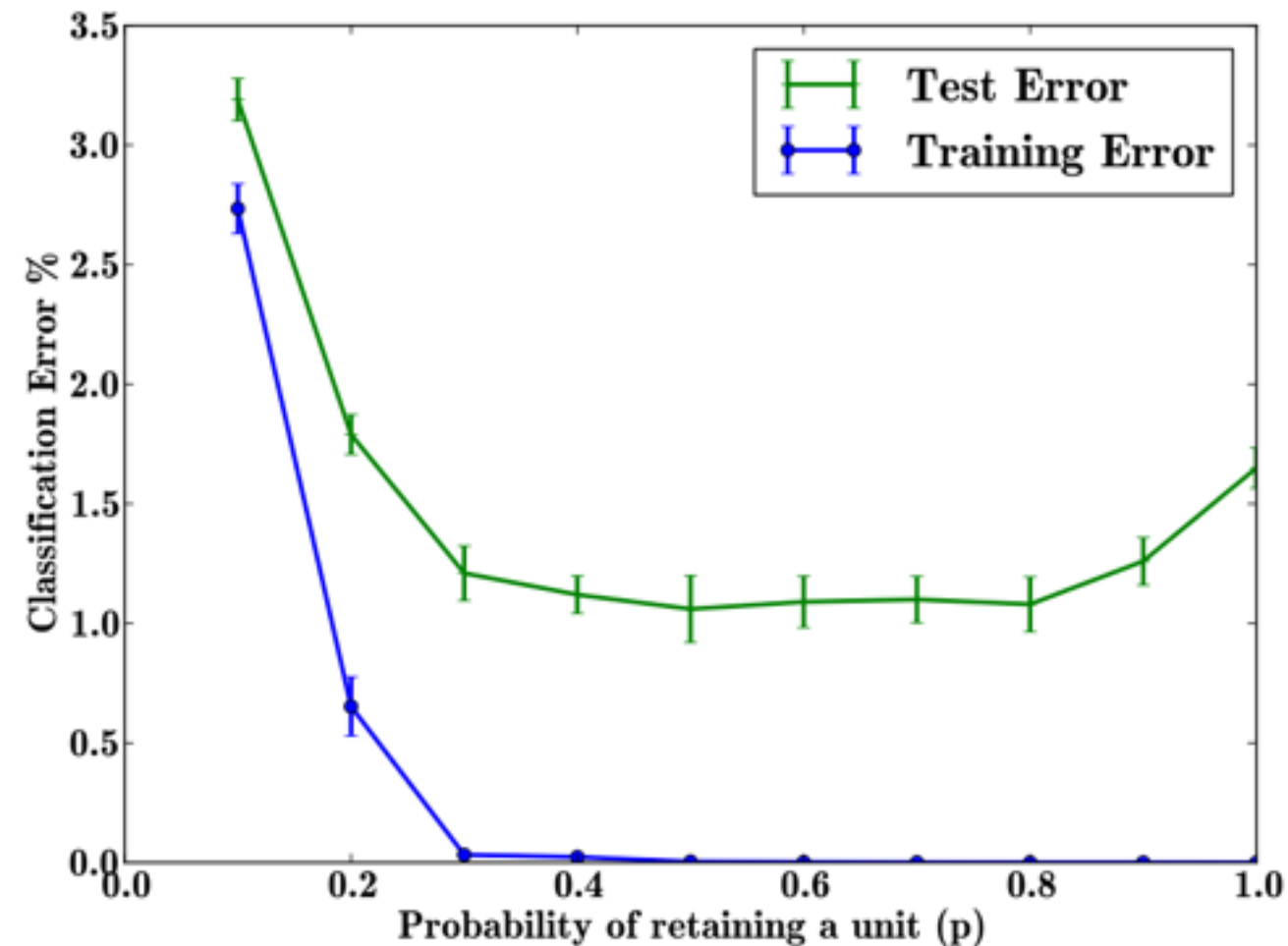
(b) After applying dropout.

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

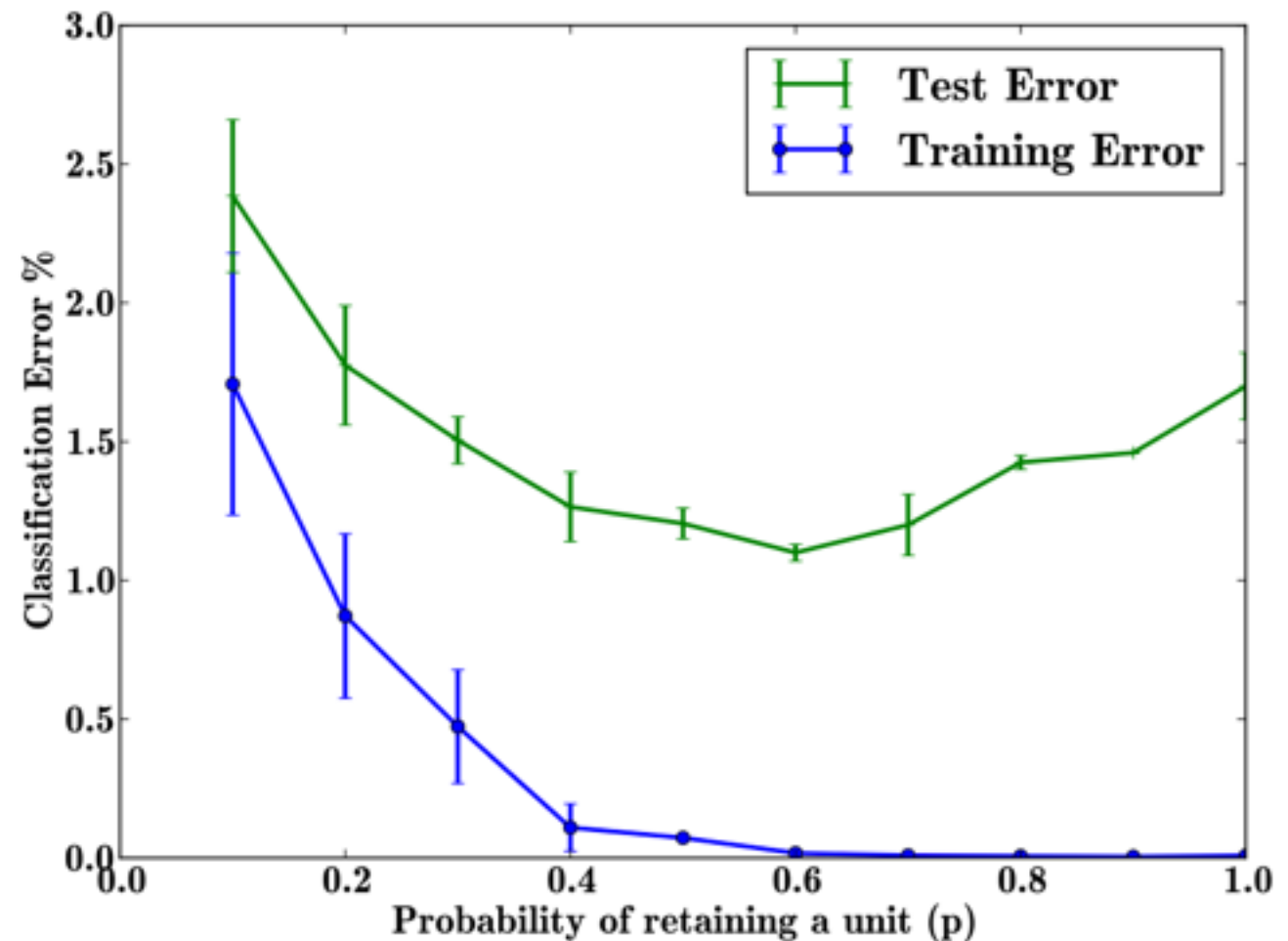
[Srivasta et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014]

Dropout

How much dropout? Around $p = 0.5$



(a) Keeping n fixed.



(b) Keeping pn fixed.

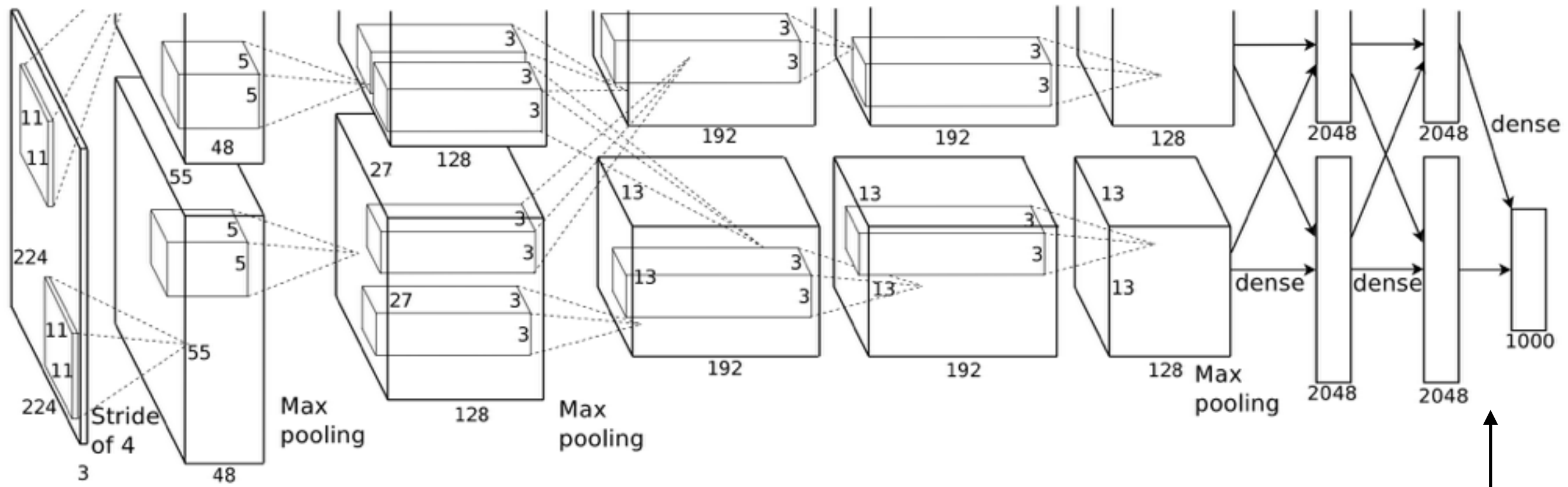
[Srivasta et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014]

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

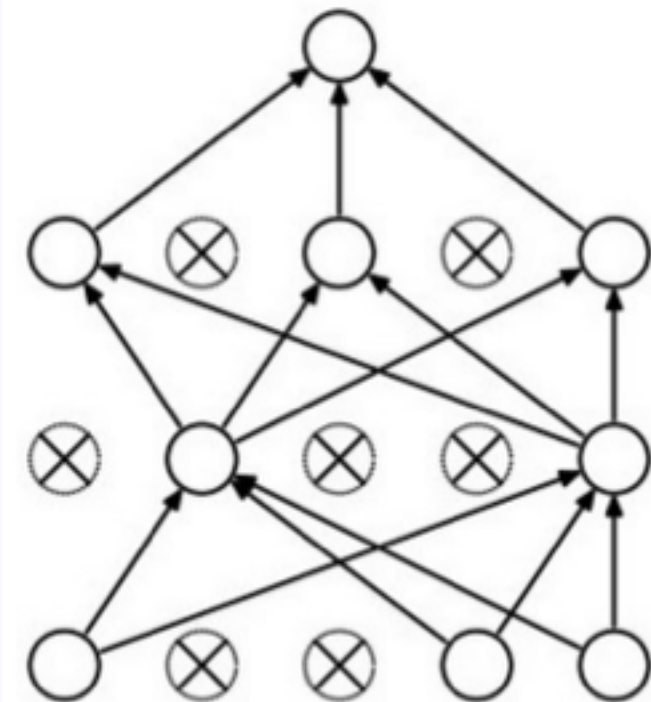
```
p = 0.5 # probability of keeping a unit active. higher = less dropout

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)

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

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:

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

this is a vanilla
differentiable function...

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

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

Batch Normalization

[Ioffe and Szegedy, 2015]

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

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

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

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

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

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

- 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

Batch Normalization

[Ioffe and Szegedy, 2015]

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

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

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

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

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

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

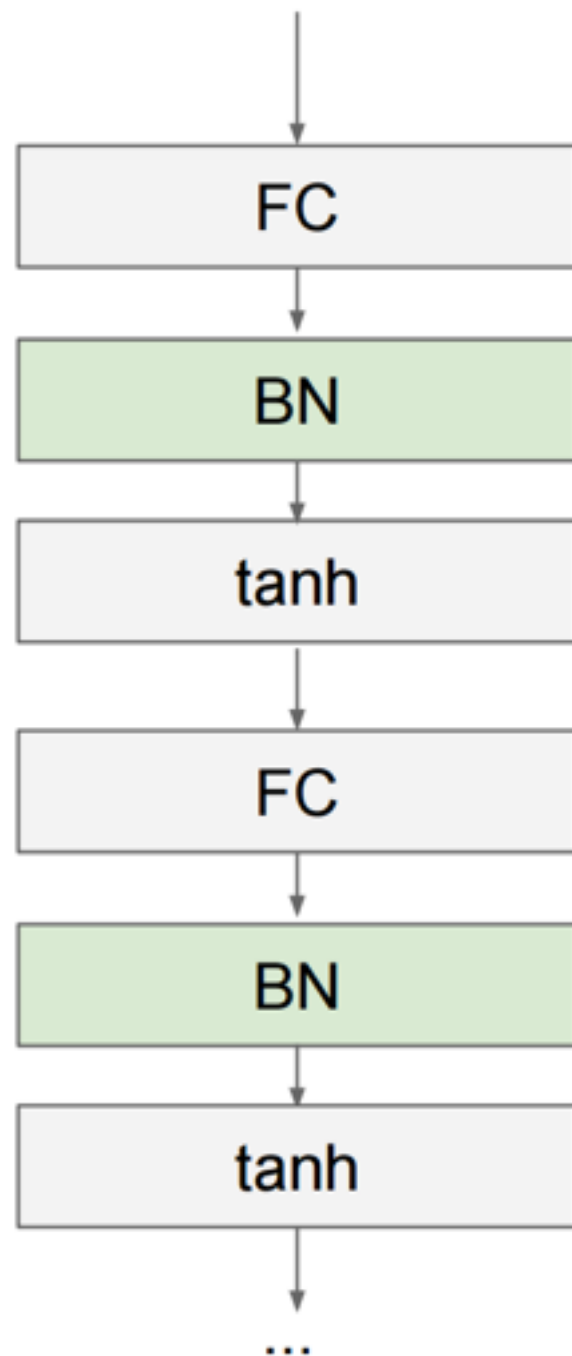
Note: at test time BatchNorm layer functions differently:

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)

Batch Normalization

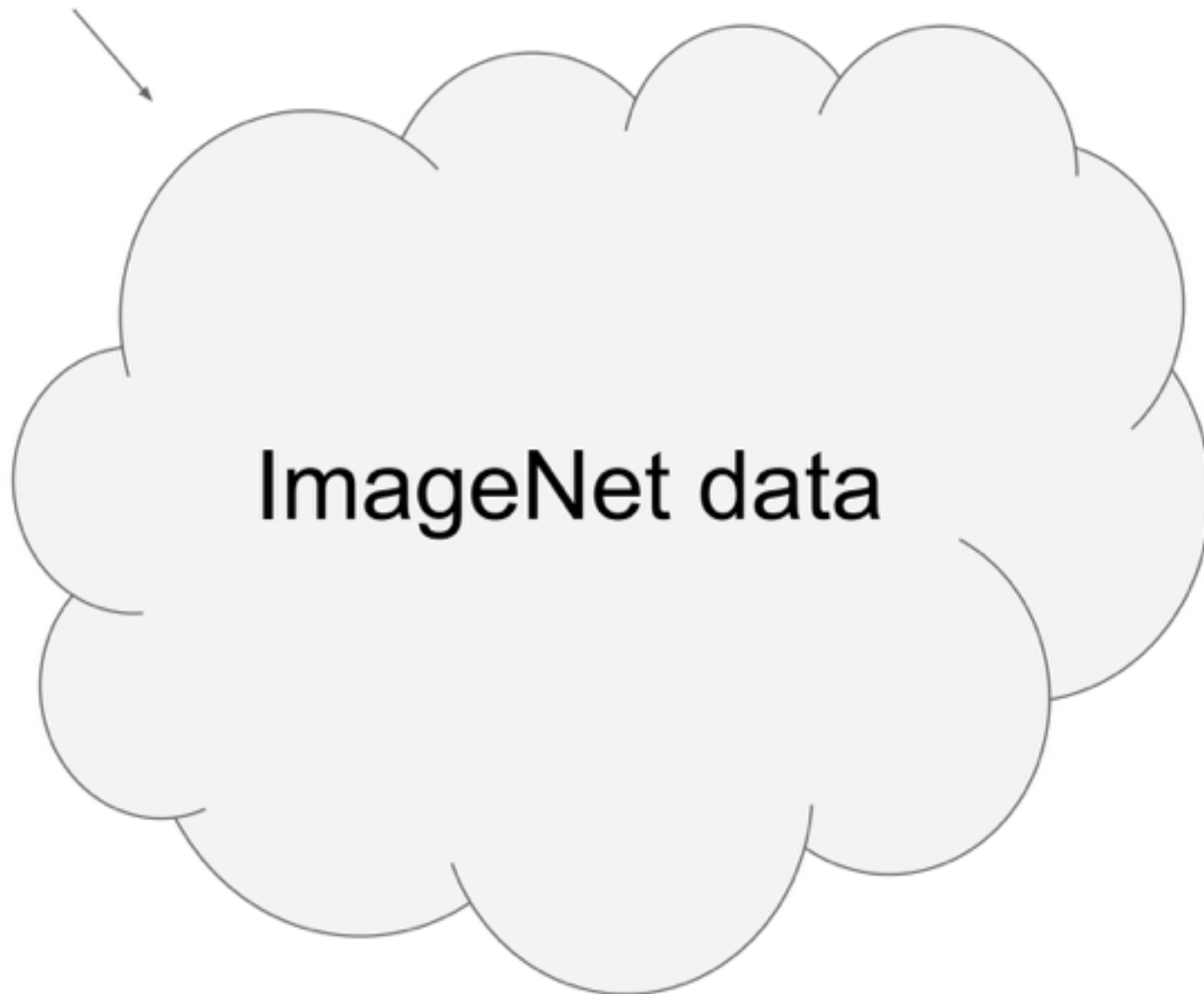
[Ioffe and Szegedy, 2015]



Place after a FC or Convolutional layer, and before nonlinearity

Transfer Learning (“fine-tuning”)

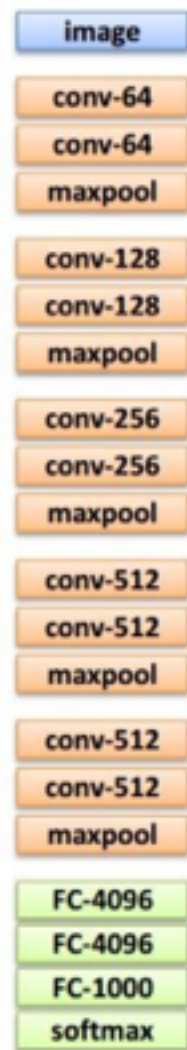
1. Train on ImageNet



2. Finetune network on your own data



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

Transfer Learning (“fine-tuning”)

E.g. Caffe Model Zoo: Lots of pretrained ConvNets
<https://github.com/BVLC/caffe/wiki/Model-Zoo>

Model Zoo

2014 edited 10 days ago · 10 revisions

Check out the [model zoo documentation](#) for details.

To acquire a model:

- download the model .prototxt by `././scripts/download_model_from_github.py caffe_net_1000_places101` to load the model metadata, architecture, solver configuration, and so on. (REDIRECT is optional and defaults to caffe/models)
- download the model weights by `././scripts/download_model_weights.py 1000_places101` where 1000_places101 is the job directory from the first step.

or visit the [model zoo documentation](#) for complete instructions.

Berkeley-trained models

- Pretraining on Flickr Style: same as provided in 1000_places101, but listed here as a DNN for an example.
- BVLC GoogLeNet: 1000_places101_googlenet.

Network in Network model

The Network in Network model is described in the following ICML-2014 paper:

Network in Network
K. Sim, S. Chu, S. Yan
International Conference on Learning Representations, 2014 (arXiv:1409.0552)

Please cite the paper if you use the models.

Models:

- INI-magnum: a small(2MB) model for imagenet, yet performs slightly better than AlexNet, and fast to train. (Note: a more caffe-compatible version with correct convolutional weights shape: <https://raw.githubusercontent.com/BVLC/caffe/master/models/imagenet/magnum.prototxt>)
- INI-CIFAR10: INI model on CIFAR10, originally published in the paper Network in Network. The error rate of this model is 10.4% on CIFAR10.

Models from the BMVC-2014 paper "Return of the Devil in the Details: Delving Deep into Convolutional Nets"

The models are trained on the ILSVRC-2012 dataset. The details can be found on the [project page](#) or in the following BMVC-2014 paper:

Return of the Devil in the Details: Delving Deep into Convolutional Nets
K. Sim, S. Chu, S. Yan, S. Kwong, A. Liew, S. Zou
British Machine Vision Conference, 2014 (arXiv:1404.3292)

Please cite the paper if you use the models.

Models:

- VGG_CNN_3: 15.1% top-5 error on ILSVRC-2012 val
- VGG_CNN_36: 13.7% top-5 error on ILSVRC-2012 val
- VGG_CNN_36_2048: 13.5% top-5 error on ILSVRC-2012 val
- VGG_CNN_36_1024: 13.7% top-5 error on ILSVRC-2012 val
- VGG_CNN_36_136: 13.8% top-5 error on ILSVRC-2012 val
- VGG_CNN_36: 14.7% top-5 error on ILSVRC-2012 val

Models used by the VGG team in ILSVRC-2014

Places-CNN model from MIT

Places-CNN is described in the following CVPR-2015 paper:

Places: A 101 Category Image Net
L. Zhang, L. Liu, Y. Li, S. Saito, and S. Belongie
Learning Deep Features for Scene Recognition using Places Database
Arxiv preprint arXiv:1505.04544, 2015.

The project page is here.

Models:

- Places-CNN: CNN trained on 101 scene categories of Places Database used in ILSVRC-2012 with ~12 million images. The architecture is the same as Caffe reference network.
- Places-CNN_VGG: VGG architecture CNN trained on 101 scene categories from Places Database and 400k depth maps from the same data. It is a VGG-like architecture with ~14 million images. The architecture is the same as Caffe reference network.
- Places-CNN_DeepSVM: DeepSVM CNN trained on 101 scene categories of Places Database. It is used to bridge in the [DeepSVM model](#).

GoogLeNet GPU Implementation from Princeton

An implementation of GoogLeNet using a single GPU. Our main contribution is an effective way to reduce the memory and to make it compatible to the GPU hardware constraints by accumulating gradients over the training iterations.

- Places-CNN-GoogLeNet: GoogLeNet architecture trained on 101 scene categories. The model is trained on ImageNet and Places, and the testing code can be found in `././scripts/evaluate_places.py`. It has a top-5 error rate of 10.4% on the Places Database. The error rate on ILSVRC-2012 is 10.4%.

Fully Convolutional Semantic Segmentation Models (FC8-X)

These models are described in the paper:

Using Hierarchical Features for Semantic Segmentation
Jonathan Long, Evan Shelton, Trevor Darrell
CVPR 2015

They are available under the same license as the Caffe model zoo, i.e., for unrestricted use with the [Creative Commons Attribution-NonCommercial-ShareAlike license](#).

These are pre-release models. They do not run in any current version of BVLC/Caffe, so they require unmodified FFIs. They should run in the private branch provided at <https://github.com/longjiaohong/caffe-fc8-x>. The FC8-X model is a deep neural network consisting of convolutional, pooling, concatenation, and fully connected layers and softmax.

Models trained on Places101 using data sets from [Kohler et al.](#) and extracted from the ILSVRC2012 VGG-16 model are:

- FC8-X Places101: single stream, 10 layer predictor single stream
- FC8-X Places101: two stream, 10 layer predictor single stream
- FC8-X Places101: three stream, 10 layer predictor single stream
- FC8-X Places101: four stream, 10 layer predictor single stream

To reproduce the validation results, use the `test.prototxt` file defined in the paper in `././scripts/evaluate_places.py` and `././scripts/evaluate_places.py`. We also include in the repository the `test.prototxt` file for validation purposes.

Models trained on SIFT Flow also trained from [Kohler et al.](#):

- FC8-X SIFT Flow: two stream, 10 layer predictor single stream

Models trained on Imagenet using data extracted from [Kohler et al.](#) and using data extracted from [Kohler et al.](#) are:

- FC8-X Imagenet: single stream, 10 layer predictor single stream
- FC8-X Imagenet: two stream, 10 layer predictor single stream
- FC8-X Imagenet: three stream, 10 layer predictor single stream

CaffeNet fine-tuned for Oxford flowers dataset

https://raw.githubusercontent.com/BVLC/caffe/master/models/flowers/caffe_net_1000_flowers.prototxt

This is the reference CaffeNet model trained on the ILSVRC-2012 dataset. The number of nodes in the test prediction layer has been set to 100 to reflect the number of flower categories. Incompatibility between model files is producing confusion for some users on their own. The project page is a linked while the training script is the best to understand in context relative to the other files.

After 1000 iterations, the top-1 error is 7.7% on the test set of 1,000 images.

CNN Models for Salient Object Subitizing

CNN models described in the following CVPR-13 paper: "Salient Object Subitizing"

Salient Object Subitizing
Z. Zhang, S. Li, M. S. Baerentzen, S. Sclaroff, R. Bales, Z. Liu, S. Shen, S. Prasad and S. Gidycz, 2013.

Models:

- ASubit: CNN model trained on the Salient Object Subitizing dataset (~6000 images). The architecture is the same as the Caffe reference network.
- VGG19: CNN model trained on the Salient Object Subitizing dataset (~6000 images). The architecture is the same as the VGG19 network. This model gives better performance than the AlexNet model, but is slower for training and testing.

Deep Learning of Binary Hash Codes for Fast Image Retrieval

We present an effective deep learning framework to create the hash-like binary codes for fast image retrieval. The details can be found in the following CVPR-13 paper:

Deep Learning of Binary Hash Codes for Fast Image Retrieval
S. Liu, W.-P. Tang, J.-H. Zhou, S.-H. Ding, D.-M. Song, 2013, Springer-Verlag

Please cite the paper if you use the model.

- CFH-CNN11: See our code release on GitHub, which allows you to train your own deep learning model and create binary hash codes.
- CFH10-ABC: Proposed 40-000 CNN model trained on CFH10.

Places_CNN50_models on Scene Recognition

- Places_CNN50: A "ResNet-like" deep Convolutional Neural Networks model trained on MIT Places Database with Deep Supervision.

The details of training this model are described in the following [report](#). Please cite this work if the model is useful for you.

Training Script: Convolutional Semantics with Deep Supervision
L. Wang, C. Liu, Z. Tu, S. Liao, 2015, arXiv:1505.04544, 2015

Models for Age and Gender Classification

- AgeGender: out-of-the-box models for age and gender classification trained on the [Adience-300](#) dataset. See the [Project page](#).

The models are described in the following paper:

Age and Gender Classification using Convolutional Neural Networks
Sui Lian, and Tao Xiang
2008 Workshop on Analysis and Modeling of Faces and Gestures (AMFG), at the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Boston, June 20

If you find our models useful, please add suitable reference to our paper in your work.

GoogLeNet_cars on car model classification

DeepCNN_cars is the GoogLeNet model pre-trained on Imagenet classification task and fine-tuned on 421 car models in [Cars196](#) dataset. It is described in the [technical report](#). Please cite the following work if the model is useful for you:

A Large Scale Car Dataset for Fine-Grained Car Detection and Verification
S. Tang, F. Liu, S. S. Tang, S. S. Tang, arXiv:1506.02595, 2015

Holistically-Nested Edge Detection

The model and code provided are described in the ICCV-2015 paper:

Holistically-Nested Edge Detection
Saining Xie and Zhendong Tu
ICCV 2015

For details about training/including HED, please take a look at <https://github.com/xieyongyong/hed>.

Model trained on BSD500 test dataset (extracted from the VGG16):

- HED-BSD500-200

Translating Videos to Natural Language

These models are described in the [FAccS-2014 paper](#):

Translating Videos to Natural Language using Deep Recurrent Neural Networks
S. Wang, K. Yu, J. Susskind, R. Rotherbach, R. Manley, A. Senior
FAccS-2014-2014

More details can be found on the [project page](#).

Model:

VGG16-Face: This model is an improved version of the model posted model described in the [FAccS-2014 paper](#). It uses video frame features from the VGG-19 layer model. This is trained only on the Youtube video dataset.

Compatibility: These are pre-release models. They do not run in any current version of BVLC/Caffe, so they require unmodified FFIs. The models are currently supported by the [FCOUTSIDE](#) branch of the Caffe fork provided at <https://github.com/longjiaohong/caffe-fc-outside> and <https://github.com/longjiaohong/caffe-fc-outside>.

VGG Face CNN descriptor

These models are described in the [BMVC-2014 paper](#):

Deep Face Recognition
Alex N. Lucey, Andrei H. Neuman, Andrew Zisserman
BMVC 2014

More details can be found on the [project page](#).

Model: VGG-Face: This is the only deep architecture based model trained from scratch using 2.6 million images of identities collected from the web. The model has been modified to work with Caffe from the original model trained using MATLAB/Netlib.

If you find our models useful, please add suitable reference to our paper in your work.

Yearbook Photo Dating

Model from the [ICCV-2013 Extreme Image Classification Workshop paper](#):

A Century of Portraits: Exploring the Visual Evolution Record of American High School Yearbook Photos
Alex Hertzog, Brian Han, Sarah Smith, Aljunta Ethier
ICCV Workshop 2013

Model and prenet files: [Yearbook](#).

CCNN: Constrained Convolutional Neural Networks for Weakly Supervised Segmentation

These models are described in the [ICCV-2011 paper](#):

CONVOLUTIONAL NEURAL NETWORKS FOR WEAKLY SUPERVISED SEGMENTATION
Sheng-Wei Chen, Philipp Krähenbühl, Trevor Darrell
ICCV 2011

These are pre-release models. They do not run in any current version of BVLC/Caffe, so they require unmodified FFIs. Full details, source code, models, protocols are available from [CCNN](#).

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.



You are now ready.



You are now ready.

