6.812/6.825

Hardware Architectures for Deep Learning

Machine Learning Basics

February 9, 2024

Vivienne Sze and Joel Emer

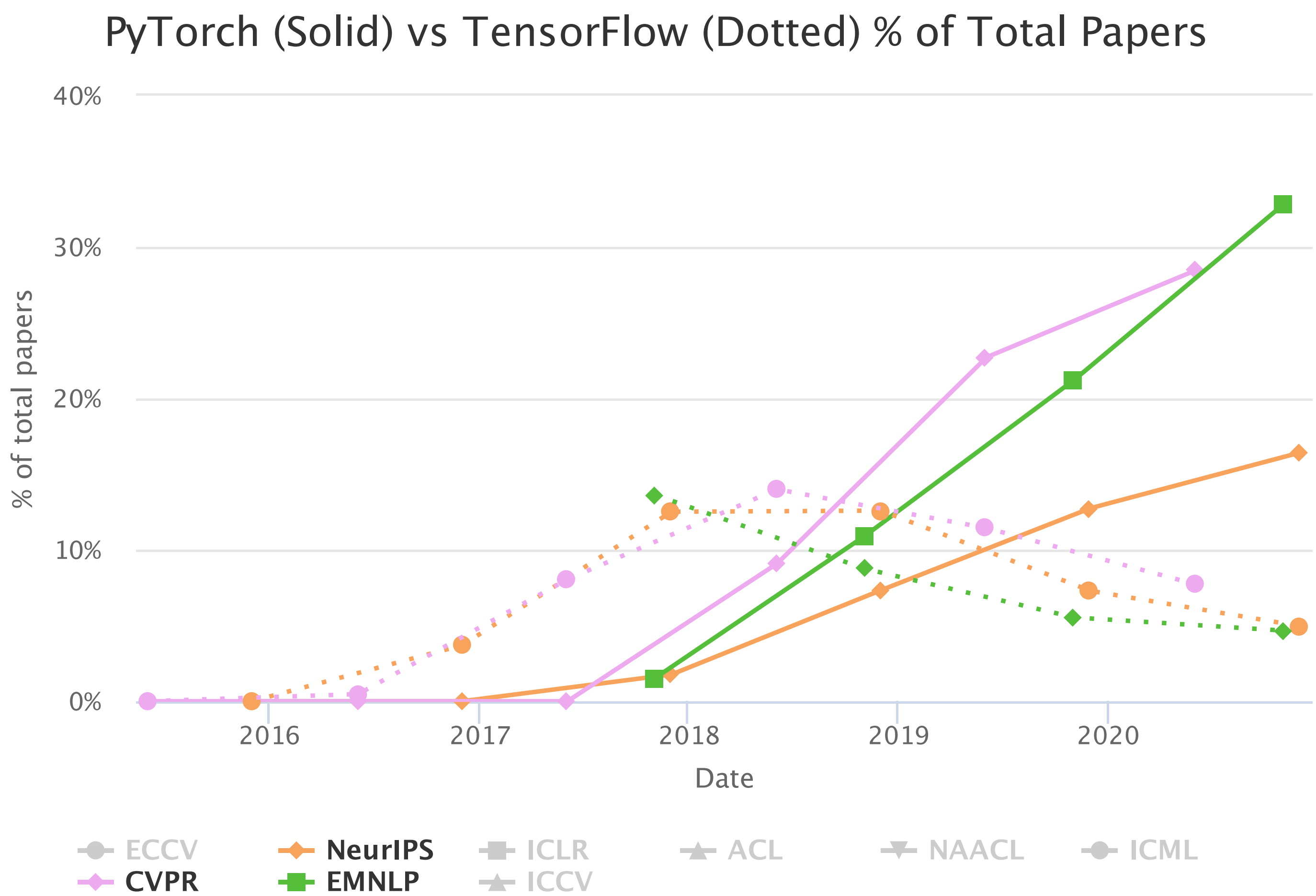
Massachusetts Institute of Technology
Electrical Engineering & Computer Science

Goals of Today's Recitation

- Brief overview of the key concepts in Machine Learning
- Use Image Classification as the driving example
  - Image representation
  - Training process
  - Hyperparameters & regularization
  - Feature extraction
- Know some basic ideas about PyTorch

PyTorch is Convenient for Research

- Easier to debug compared to TensorFlow



http://horace.io/pytorch-vs-tensorflow/

Image Classification Task

One Image

[Source: Stanford cs231n]

One Label

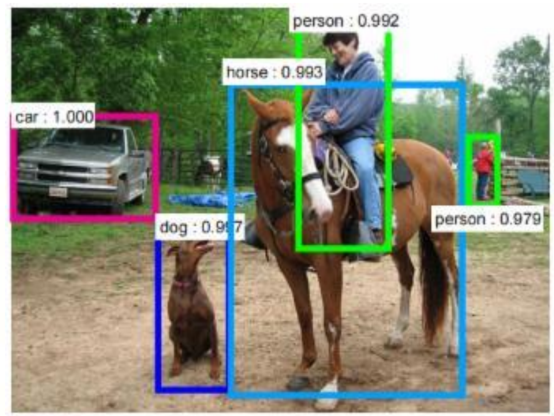
→ Classifier → **Cat**

Image Classification Task

[Source: Stanford cs231n]

Core Problem in Computer Vision.
Also referred to as *Image Recognition*

Can be extended to other vision tasks



Object Detection

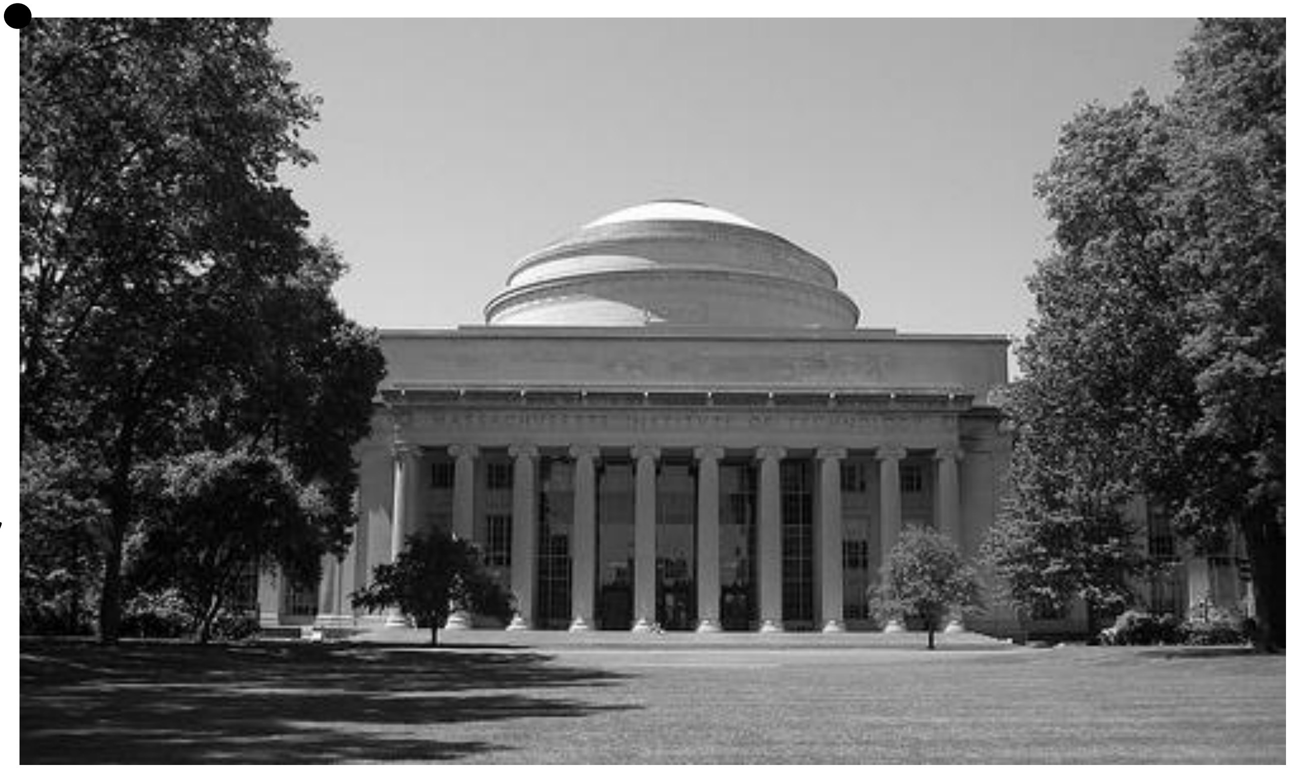
Image Segmentation

What is an Image?

- Images are **2-D functions**: *f(x, y)*
  - *x, y* are spatial coordinates
  - *f(x,y)* is the intensity/amplitude at *(x,y)*

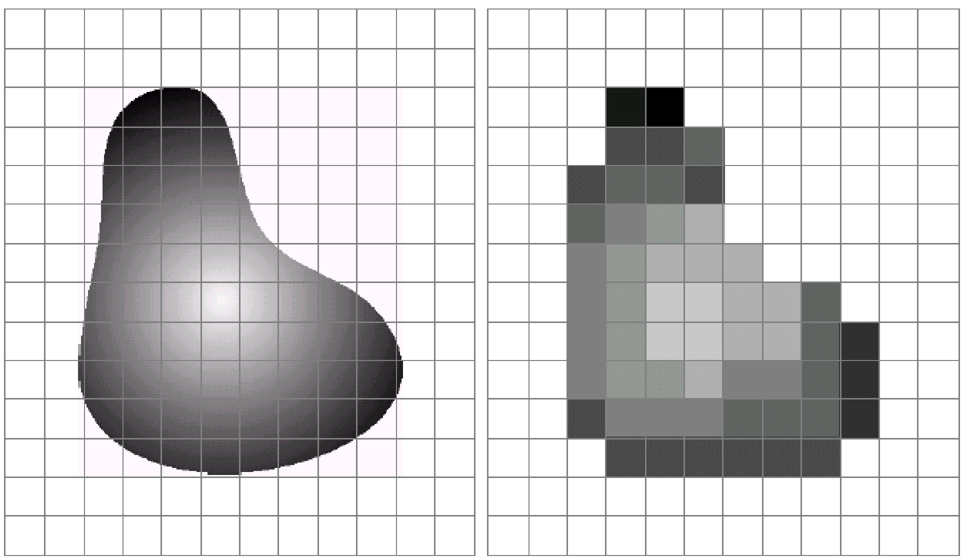
(0,0) → ***x***

↓ ***y***



What is a Digital Image?

- Sampling [Spatial] → Resolution
  - Size in terms of pixels (integer values)
- Quantization [Amplitude]→ Bits per pixel
  - e.g. 8-bits per pixel (amplitude has values between 0 to 255)



a b

FIGURE 2.17 (a) Continuos image projected onto a sensor array. (b) Result of image sampling and quantization.

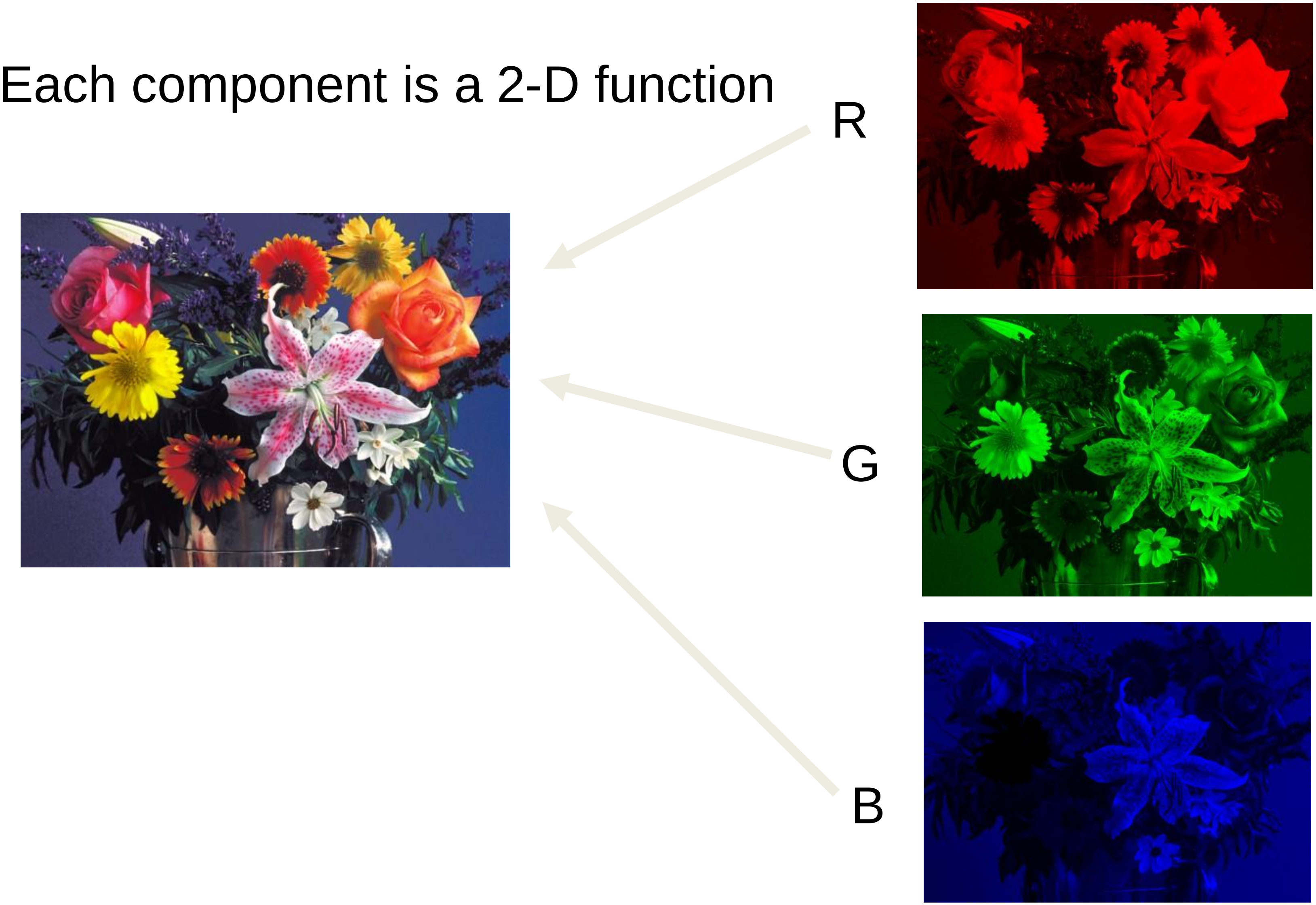
f (x, y)

y \ *x*	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	22	0	0	0	0	0	0	0	0
3	0	0	0	71	78	95	0	0	0	0	0	0
4	0	0	78	98	102	71	0	0	0	0	0	0
5	0	0	95	126	150	175	0	0	0	0	0	0
6	0	0	126	150	175	175	175	0	0	0	0	0
7	0	0	126	150	199	199	175	175	95	0	0	0
8	0	0	126	150	199	199	175	715	95	47	0	0
9	0	0	126	150	150	175	126	126	85	47	0	0
10	0	0	71	126	126	126	95	95	95	47	0	0
11	0	0	0	71	71	71	71	71	71	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0

[Image Source: R. C. Gonzalez & R. E. Woods]

Color Images

Each component is a 2-D function



Generate Other Colors from RGB

Red, green and blue each have values between 0 to 255

256*256*256 = **16,777,216** possible colors

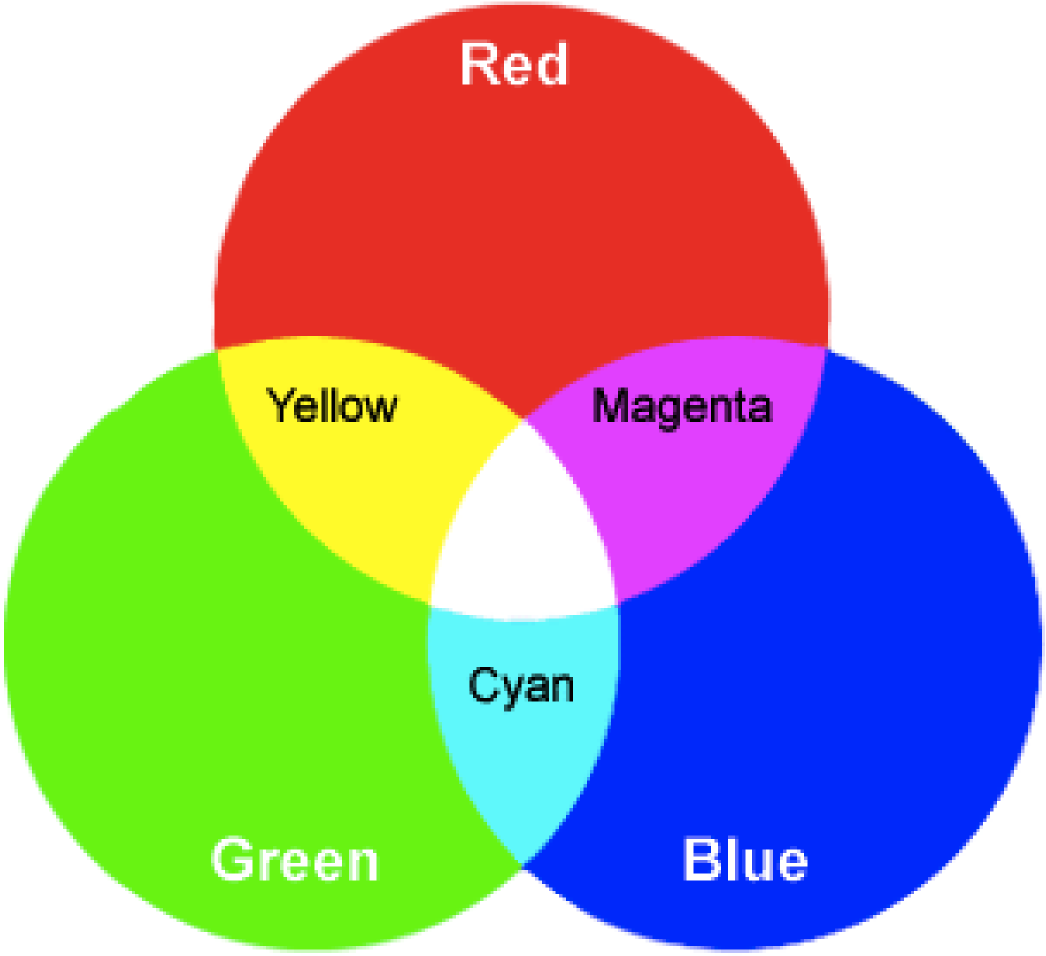
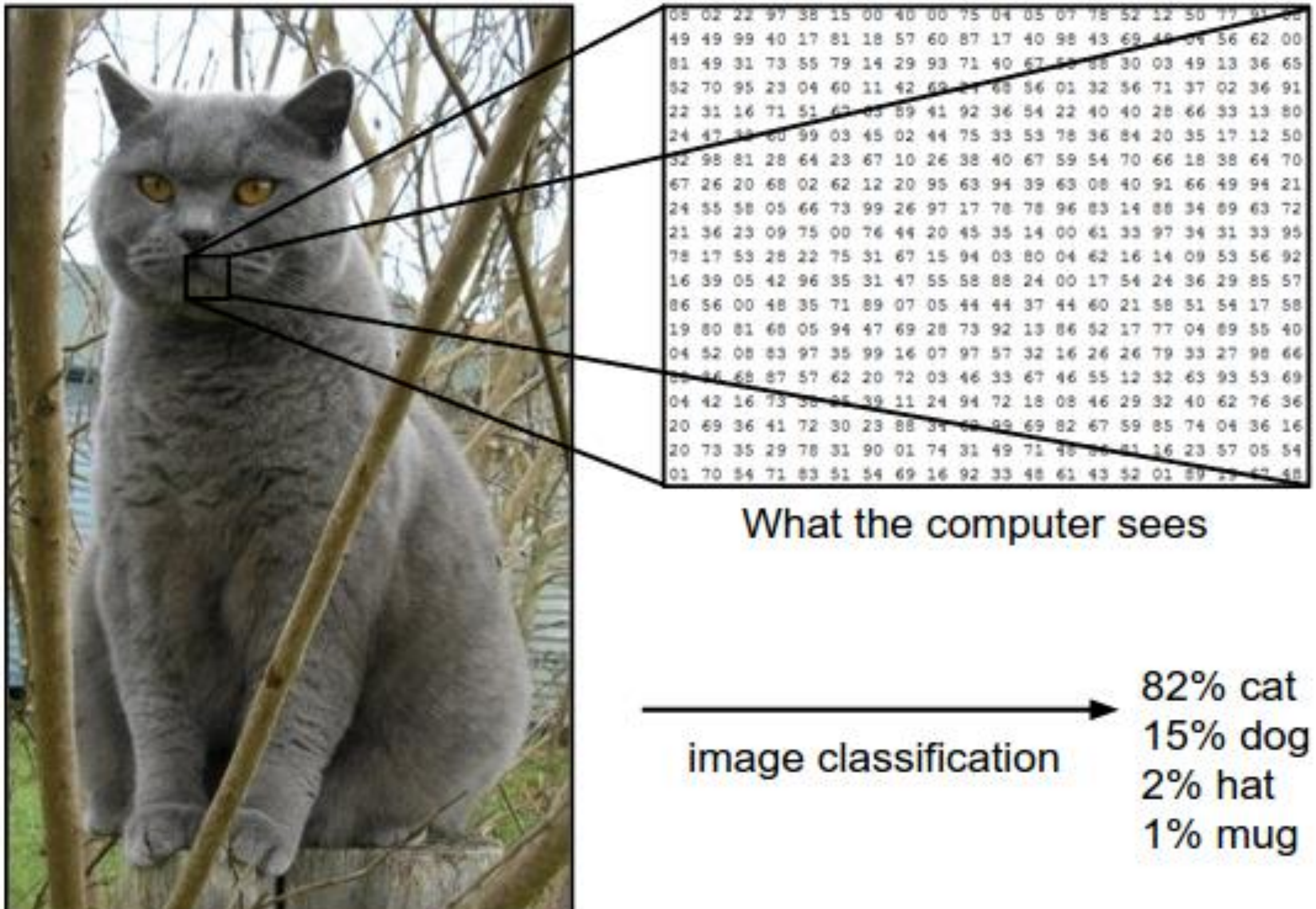


Image Classification Task



[Source: Stanford cs231n]

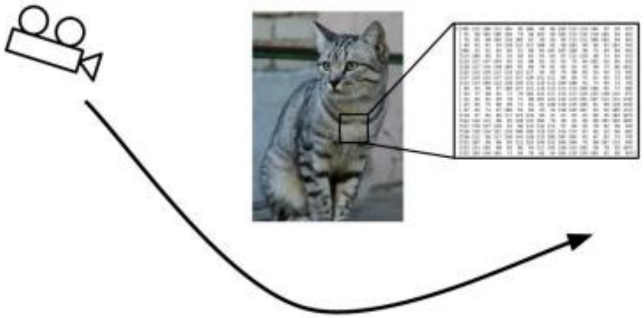
One hot encoding

$$\begin{bmatrix} \mathbf{1} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

class membership

Image Classification Challenge

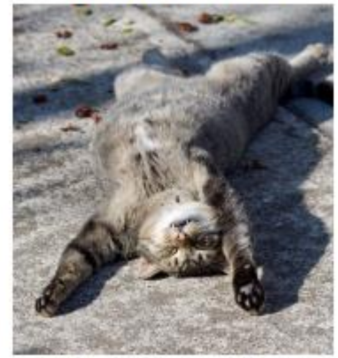
Viewpoint



Illumination

This image is CC0 1.0 public domain

Deformation



This image by Umberto Salvagnin is licensed under CC-BY 2.0

Occlusion

This image by jonsson is licensed under CC-BY 2.0

Clutter

This image is CC0 1.0 public domain

Intraclass Variation

This image is CC0 1.0 public domain

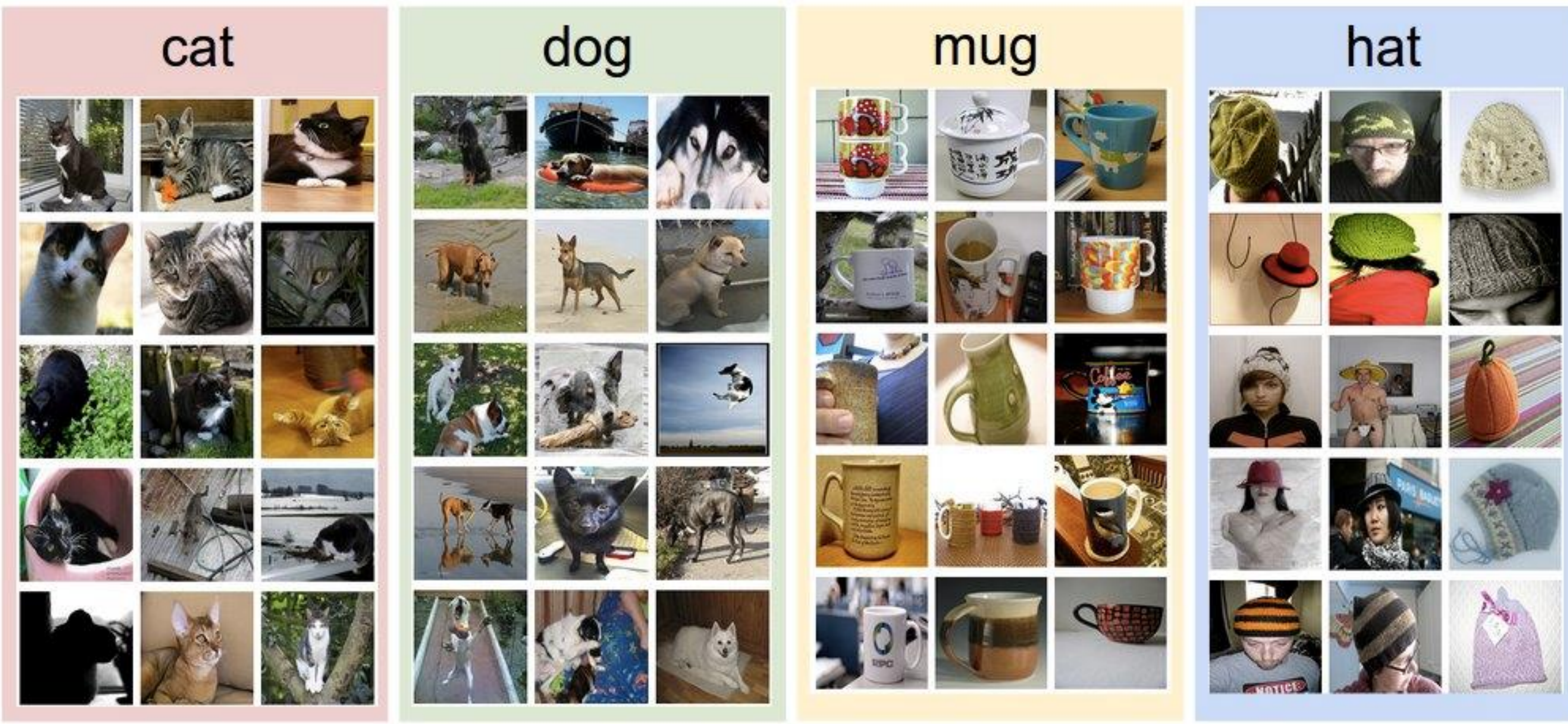
[Source: Stanford cs231n]

Need a classifier that is invariant to these variations, but still sensitive to inter class variations

Use Data Driven Approach

Give the computer example images to "learn" from

Collect dataset of labeled images



[Image Source: Stanford cs231n]

Example Dataset: CIFAR-10

32x32 pixels (color)
10 classes
50,000 Training
10,000 Testing

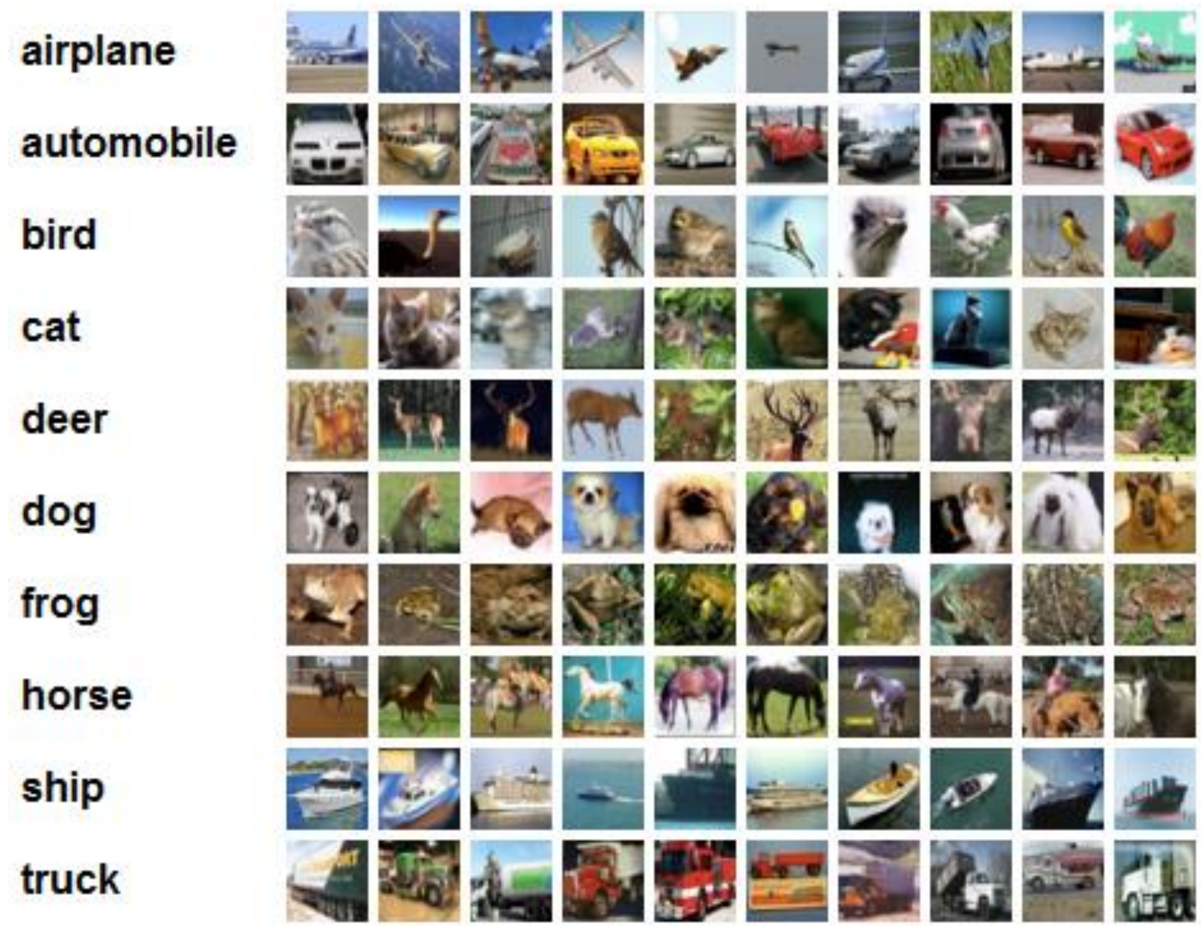


Image Source: http://karpathy.github.io/

Download from:
https://www.cs.toronto.edu/~kriz/cifar.html

Subset of 80M [Tiny Images Dataset](#) (Torralba)

In PyTorch: `torchvision.datasets`

General Steps

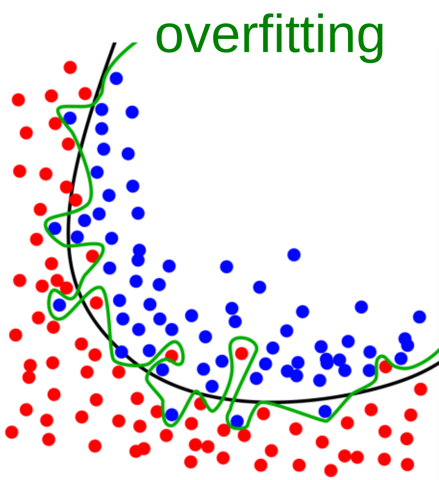
1. **Collect Labeled Dataset**
   – Use subsets of the data for **training** and **testing**
2. **Train the Model**
   – Use the **training set** to **learn** task
3. **Test the Model**
   – Use the model to **predict** labels for the **test set** **that it has never seen before**, and compare to true labels (ground truth)
4. **Use the Model (Inference)**
   – Apply model to unlabeled inputs

Steps for Training an Image Classifier

1. **Collect Labeled Dataset**
   - A set of N images, each labeled with one of K different classes
2. **Train the Model**
   - Use the **training set** to train classifier to **learn** what each of the classes looks like
3. **Test the Model**
   - Use the classifier to **predict** labels for the **test set** of images **that it has never seen before**, and compare to true labels
4. **Use the Model (Inference)**
   - Apply model to unlabeled images

Generalization

- After achieving adequate accuracy on the training set, the ultimate quality of the model is determined by how accurate it performs on unseen data
  - **The test set is a surrogate for unseen data**
- **Generalization** refers to how well the model maintains the accuracy between training and unseen data
  - Generalization means not **overfitting**
  - **Overfitting:** fit noise rather than signal
- What are techniques that can help the model generalize?



[Image source: Wikipedia]

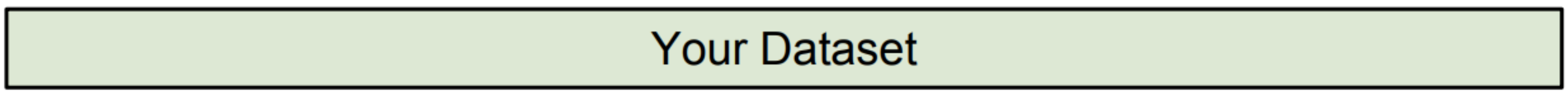
Hyper-Parameters

- Hyper-parameters are **design choices** about the algorithm that we **set rather than learn**
- Example for DNNs:
  - What is the **number of layers**?
  - What is the **shape of filter**?
- Need to try out several times

Evaluating Hyper-Parameters

<u>Example</u>: selecting number of layers

Idea #1: Choose hyperparameters
that work best on the data

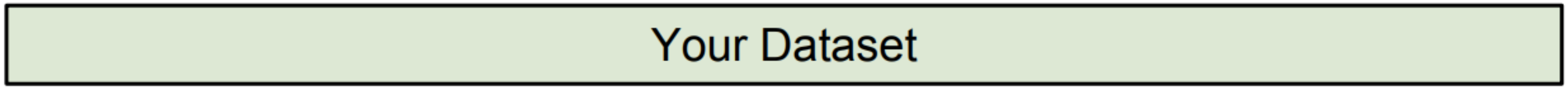


If we use the entire dataset to select the hyper-parameters, we cannot evaluate how the model generalizes.

Evaluating Hyper-Parameters

Example: selecting number of layers

Idea #1: Choose hyperparameters that work best on the data



Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

If we use the test data to select the hyper-parameters, we will need to access the test data often.

Each access to the test data “leaks information” and makes it less of a surrogate for unseen data.

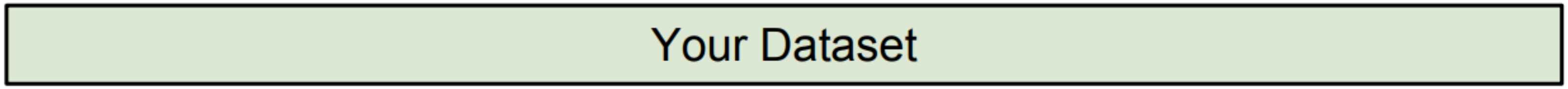
[Image Source: Stanford cs231n]

Use Validation Set

- Use **validation set** to help **choose hyper-parameters**
  - **Minimize access to test set**

<u>Example</u>: selecting number of layers

Idea #1: Choose hyperparameters that work best on the data



Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!

For ImageNet Challenge, test set not released!

Summary

1. **Collect Labeled Dataset**
   - Partition into training, validation and test set
2. **Train Model**
   - Select hyper-parameters
   - Use the **training set** to **learn** task
3. **Evaluate Model**
   - Compare results of model with true answers (ground truth labels) on the **validation set**
   - If not happy, repeat step 2!
4. **Test Model**
   - Compare results of model with true answers (ground truth labels) on the **test set**
5. **Deploy Model (Inference)**
   - **$$$**

Linear Classifier

- A linear function that maps images to class scores
  - **Input**: Image pixels (or features – discussed later)
  - **Parameters**: Weights and bias (values to be trained)
  - **Scores:** Indicate how likely image belongs to a class
  - **Labels**: Indicate which class

Image

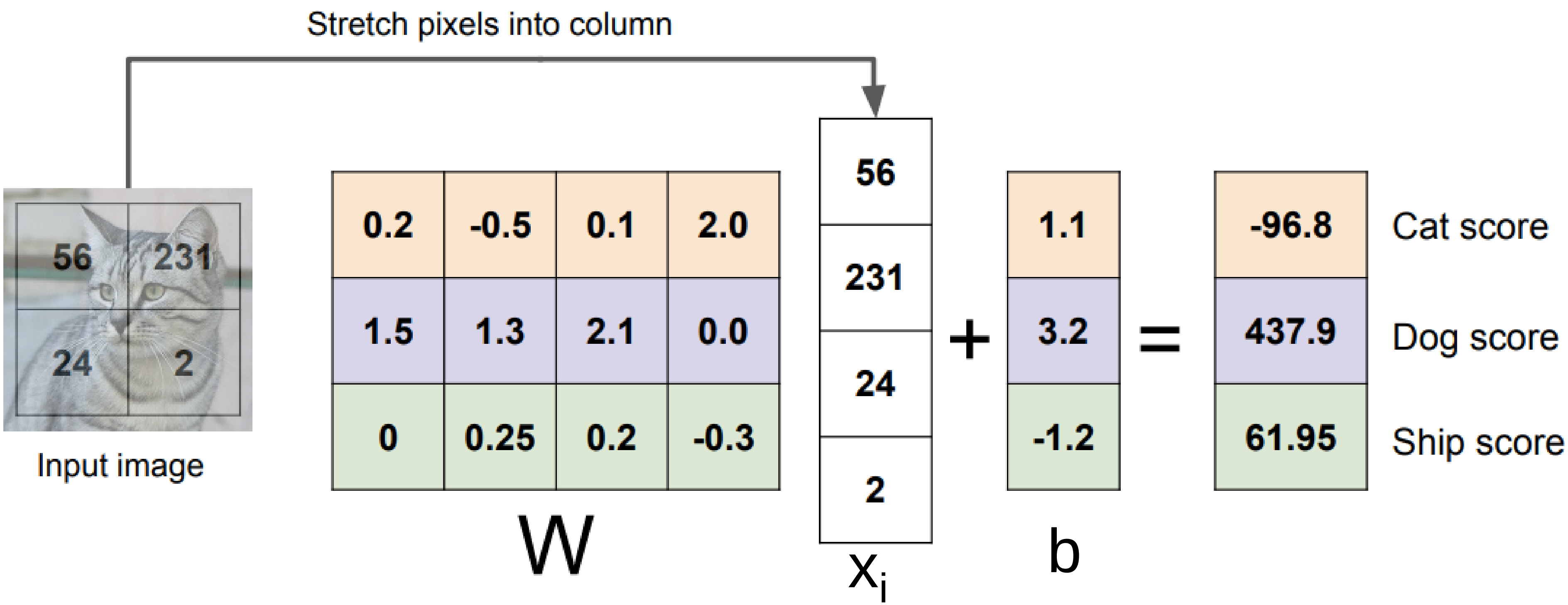
Array of **32x32x3** numbers
(3072 numbers total)

→ $f(x, W, b)$ → **10** numbers giving class scores → Labels

[Modified from Source: Stanford cs231n]

Linear Classifier

$$f(x_i, W, b) = Wx_i + b$$



For CIFAR-10 [10 x 3072] [3072 x 1] [10 x 1] [10 x 1]

In PyTorch: `torch.nn.Linear`

[Source: Stanford cs231n]

Linear Classifier

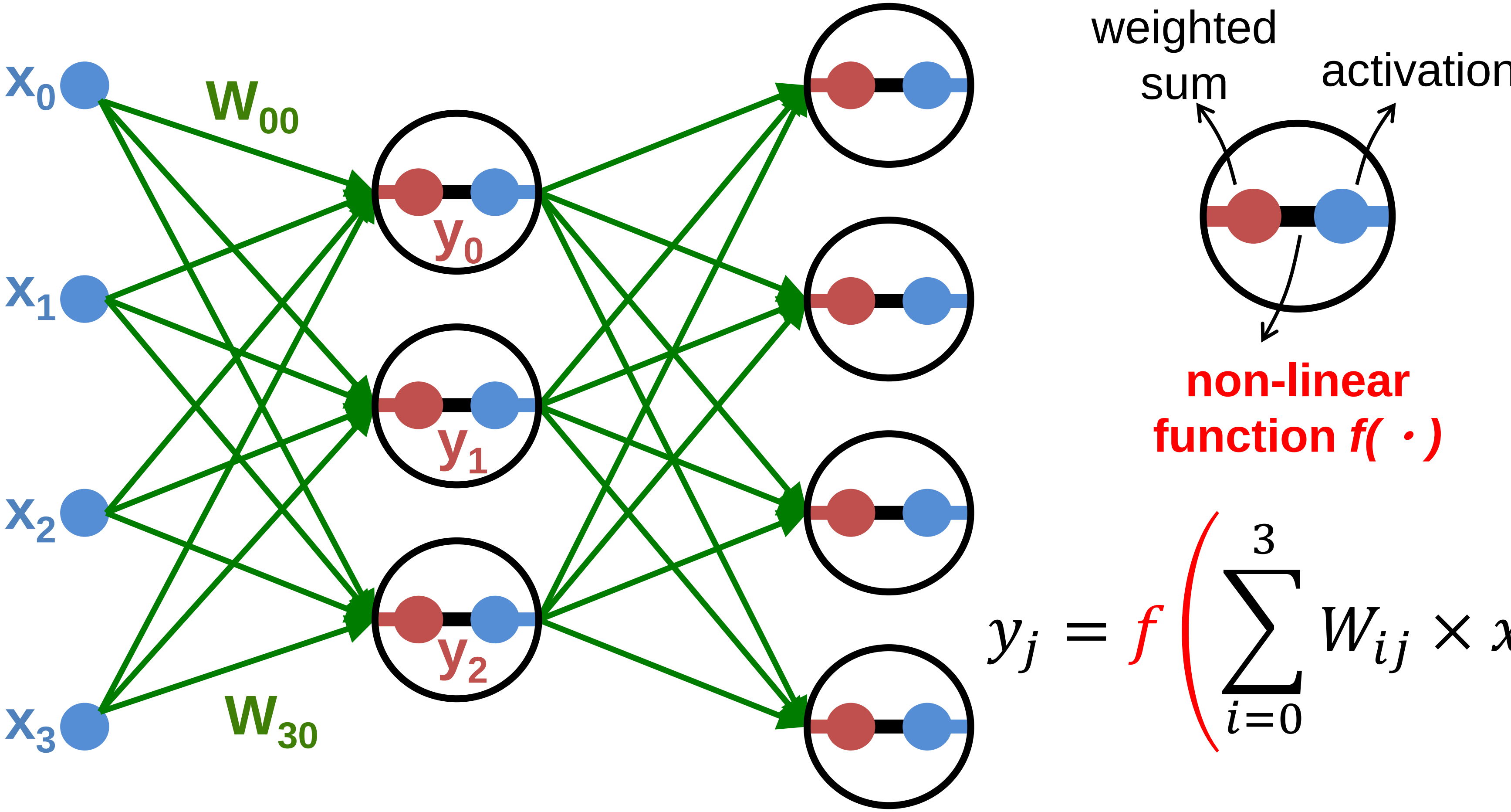
Combine bias and weights in to a single Weight matrix

$$\underbrace{\begin{bmatrix}0.2&-0.5&0.1&2.0\\1.5&1.3&2.1&0.0\\0&0.25&0.2&-0.3\end{bmatrix}}_{W}\underbrace{\begin{bmatrix}56\\231\\24\\2\end{bmatrix}}_{x_i}+\underbrace{\begin{bmatrix}1.1\\3.2\\-1.2\end{bmatrix}}_{b}\longleftrightarrow\underbrace{\boxed{\underbrace{\begin{bmatrix}0.2&-0.5&0.1&2.0\\1.5&1.3&2.1&0.0\\0&0.25&0.2&-0.3\end{bmatrix}}_{W}\underbrace{\begin{bmatrix}1.1\\3.2\\-1.2\end{bmatrix}}_{b}}}_{\text{new, single W}}\underbrace{\begin{bmatrix}56\\231\\24\\2\\1\end{bmatrix}}_{x_i}$$

- Each row of matrix W is a classifier for a given class
- **Single Matrix Multiplication** evaluates **multiples classes in parallel**

Linear Classifier

Linear Classifier can be thought of as a basic building block in the **neural network**



$$y_j = f\left(\sum_{i=0}^{3} W_{ij} \times x_i\right)$$

Intuition of Classifier

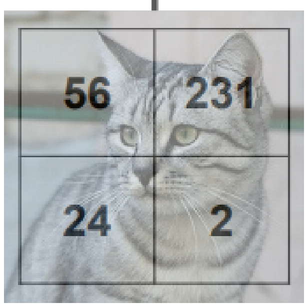
Visualizing weights of each classifier
Can be thought of as a template for the class

[Image Source: Stanford cs231n]

Linear Classifier

$$f(x_i, W, b) = Wx_i + b$$

Stretch pixels into column



Input image

$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{x_i} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b} = \begin{array}{ll} \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix} & \begin{array}{l} \text{Cat score} \\ \text{Dog score} \\ \text{Ship score} \end{array} \end{array}$$

For each image, the classifier generates scores for all classes.

How do we evaluate the quality the classifier?

Use Loss Function for Evaluation

- **Loss function** quantifies the agreement between the predicted scores and the ground truth labels
  - Scores are also referred to as **logits**
- Quantifying loss allows us to improve classifier (i.e. update weights) – *how good is the classifier?*

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Goal

- Want the class that **matches the ground truth** label to have **highest score**
- Want the classes that **don't match the ground truth** label to have **low scores**

[Image Source: Stanford cs231n]

Loss Function

Cross-Entropy Loss (Softmax)

Compute score for each class $s_j = f(x_i, W)_j$

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

Score of correct class

Score of each class

Loss function is derived from **minimizing cross-entropy** between estimated class probabilities and ground truth

In PyTorch: `torch.nn.CrossEntropyLoss`

Update weights such that the correct label has the highest probability

Use Loss Function for Evaluation

- **Loss function** quantifies the agreement between the predicted scores and the ground truth labels
  - Scores are also referred to as **logits**

Scores (logits)

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Ground Truth Labels

1	0	0
0	**1**	0
0	0	**1**

[Image Source: Stanford cs231n]

Loss Function

Compute Average Loss on Training Examples

i = index of example
N = number of examples

Average Loss

Loss Function

$$L = \underbrace{\frac{1}{N}\sum_i}_{\text{Average across number of training examples}} L_i(\overbrace{f(x_i, W)}^{\text{Prediction Score}}, y_i)$$

Input training image (given)

Weight Matrix (trained)

Correct class of training image (given)

Many possible functions for L_i

Use Loss Function for Evaluation

- Ratios of scores can be used to evaluate the quality of the classifier
- Use the **softmax function** to keep values between 0 and 1

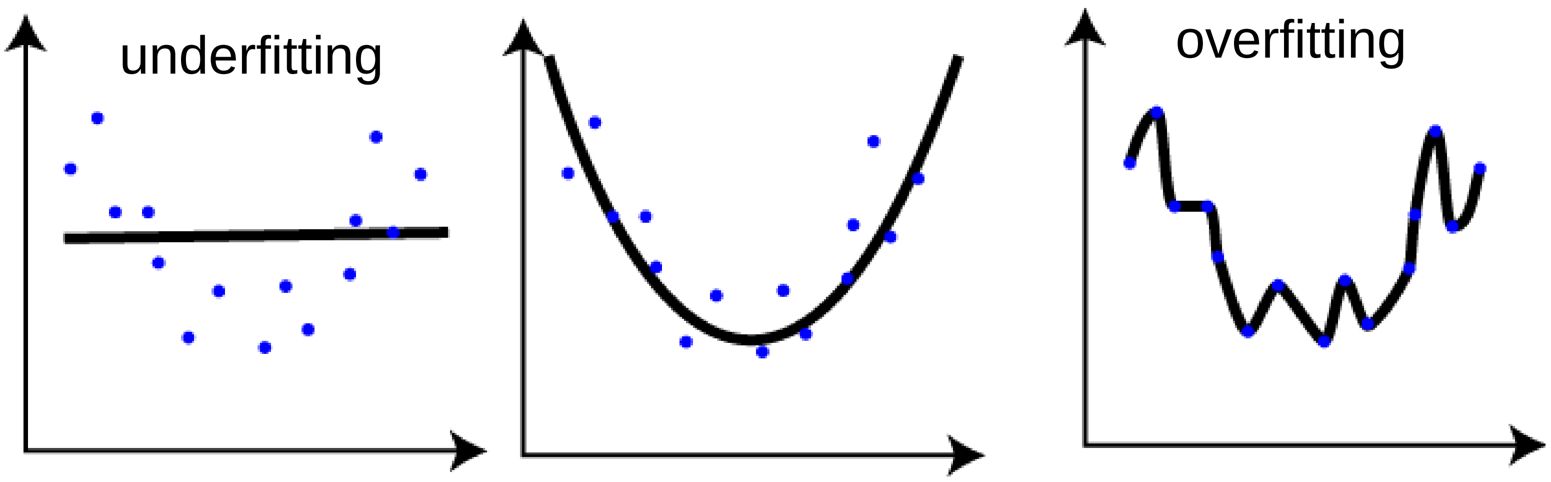
$$f = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}}$$ Softmax function

	Scores (logits)	$e^{s_{y_i}}$	$f = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}}$
			
cat	1.3	3.67	0.025
car	**4.9**	**134.3**	**0.924**
frog	2.0	7.39	0.051

Probabilities (takes on values close to ground truth labels)

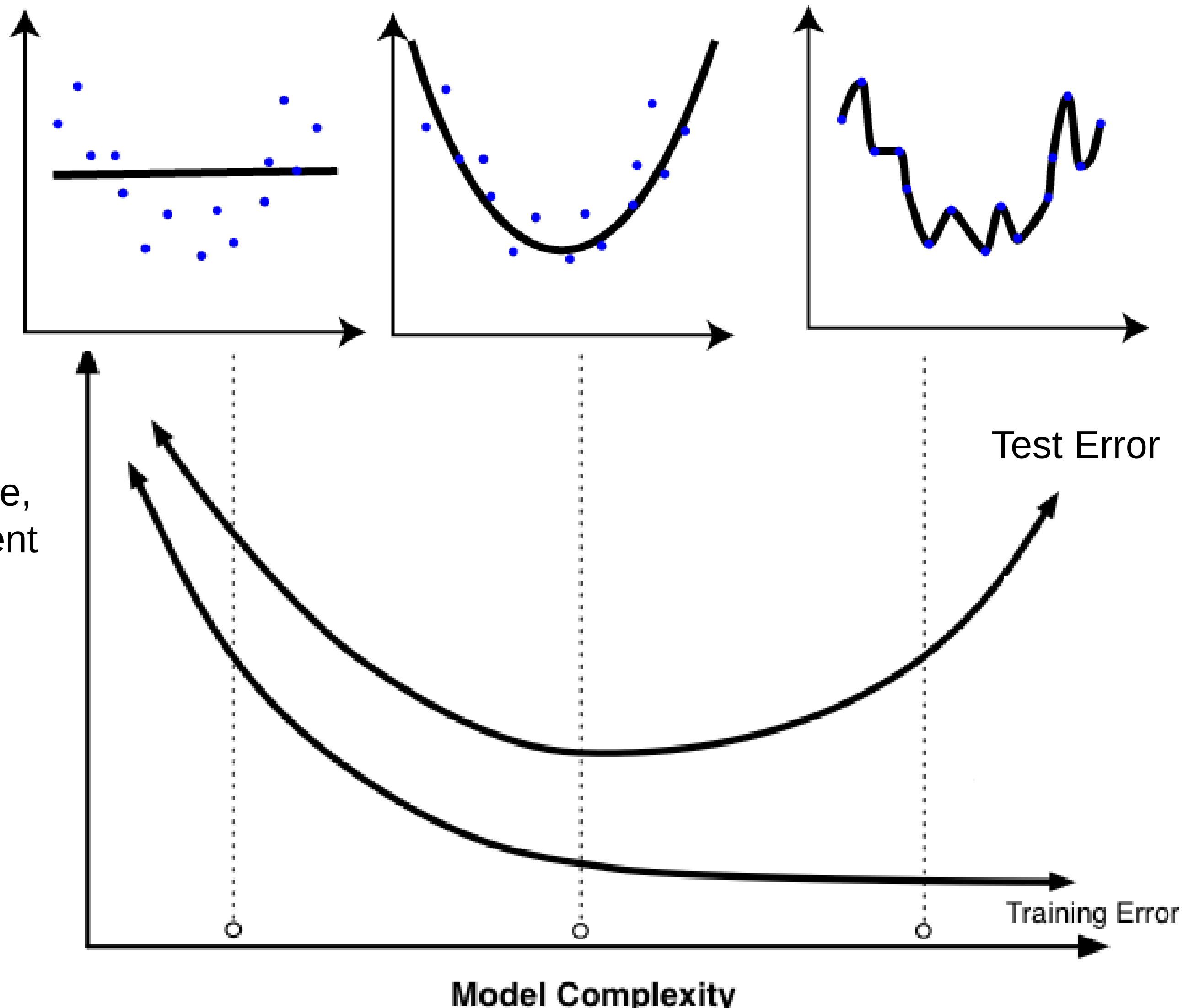
Regularization

- **Regularization** adds constraints to improve **generalizability of model**
  - Examples: smoothness, number of parameters, size of the parameters (weight decay), prior distribution or structure



[Image source: The Shape of Data]

Regularization: Training vs. Test Error

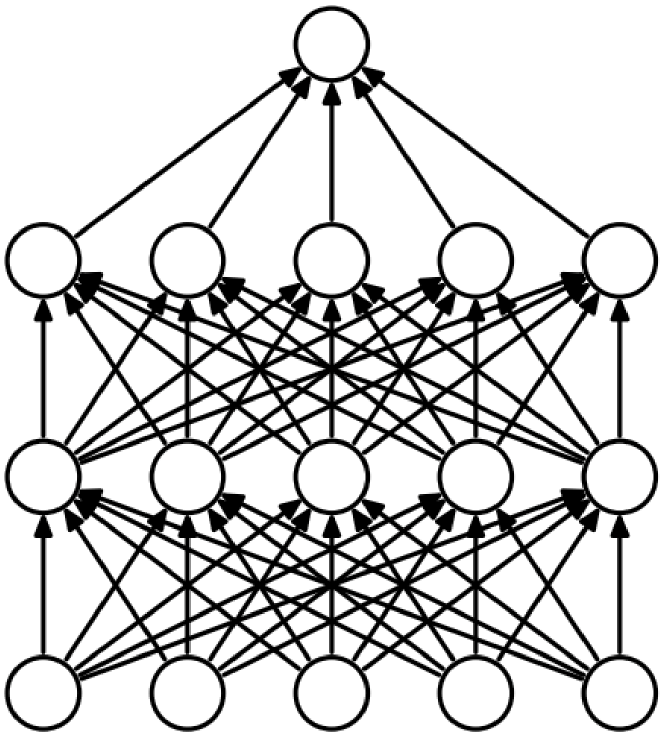


Note: In this case, **error** is equivalent to **loss**

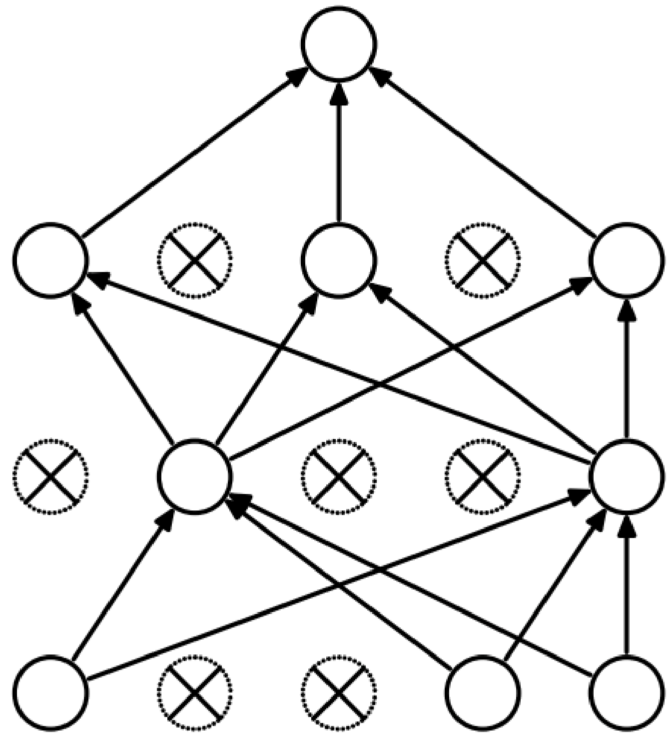
[Modified from Source: scott.fortmann-roe.com/]

Regularization for DNN: Dropout

During training, **randomly** set some activations and their weights to zero.
Reduces over-fitting by helping the activations (i.e. feature detectors) to be robust to changes in its neighbors.



(a) Standard Neural Net

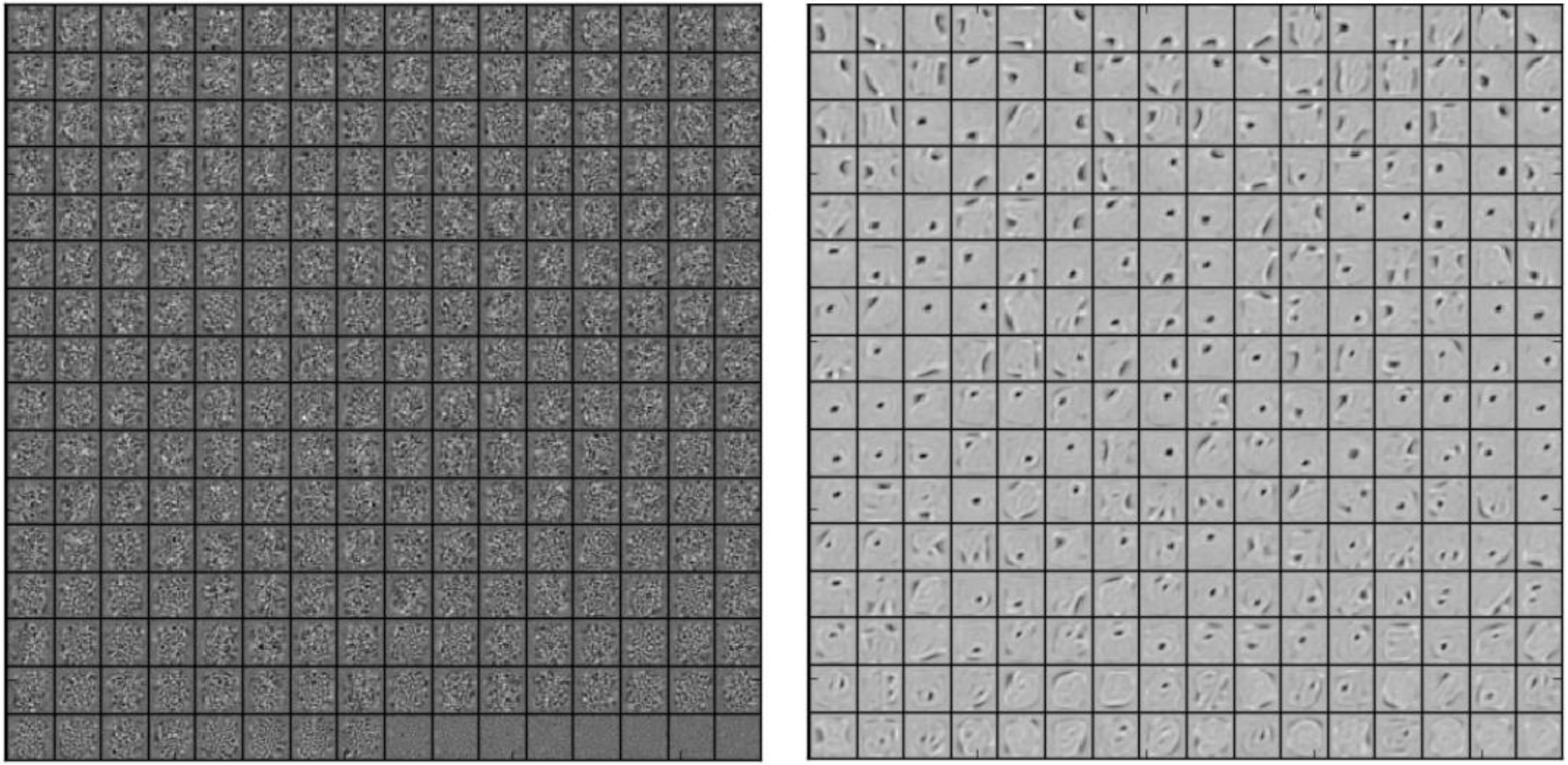


(b) After applying dropout.

[**Srivastava**, *JMLR* 2014]

Regularization for DNN: Dropout

Dropout results in more meaningful learned features (e.g. detect edges, strokes and spots in different parts of the image). Results on MNIST shown.



(a) Without dropout (b) Dropout with $p = 0.5$.

[**Srivastava**, *JMLR* 2014]

Total Loss

$$L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

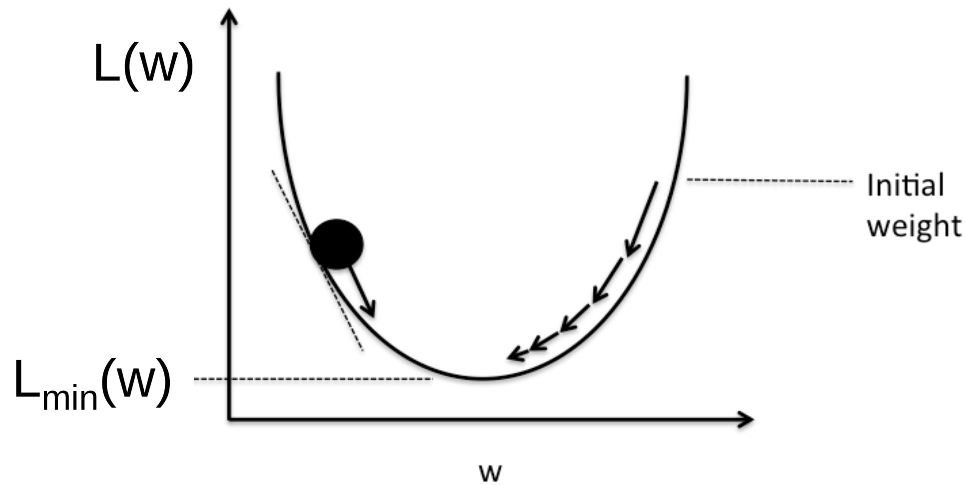
λ is a hyper-parameter set during training.
Larger **λ** improves generalizability, but may increase data loss.

Gradient Descent

- **Goal:** Determine set of weights to minimize loss
- Use **gradient descent** to incrementally update weights to reduce loss
  - Compute derivative of loss relative to weights to indicate how to change weights (linear approximation of loss function)

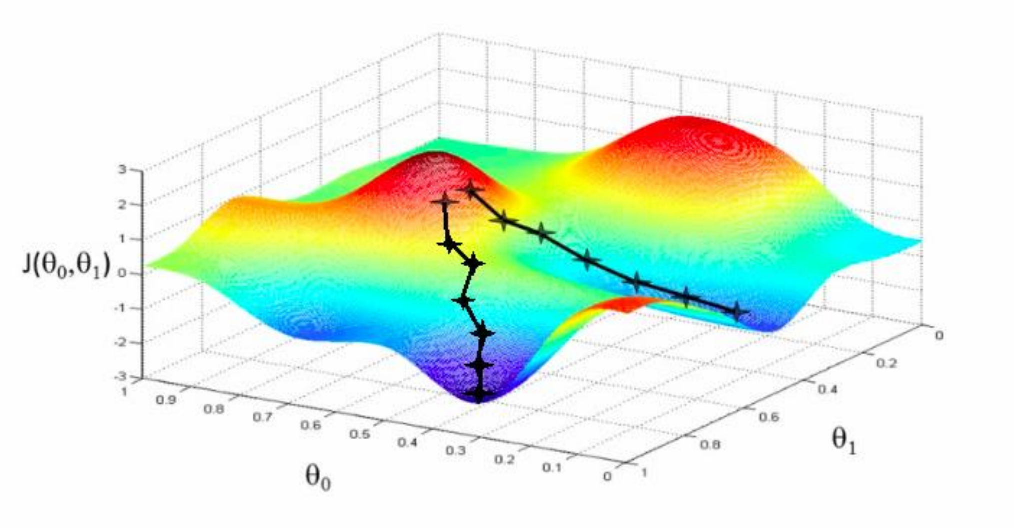
$$w_{ij}^{t+1} = w_{ij}^{t} - \alpha \frac{\partial L}{\partial w_{ij}}$$

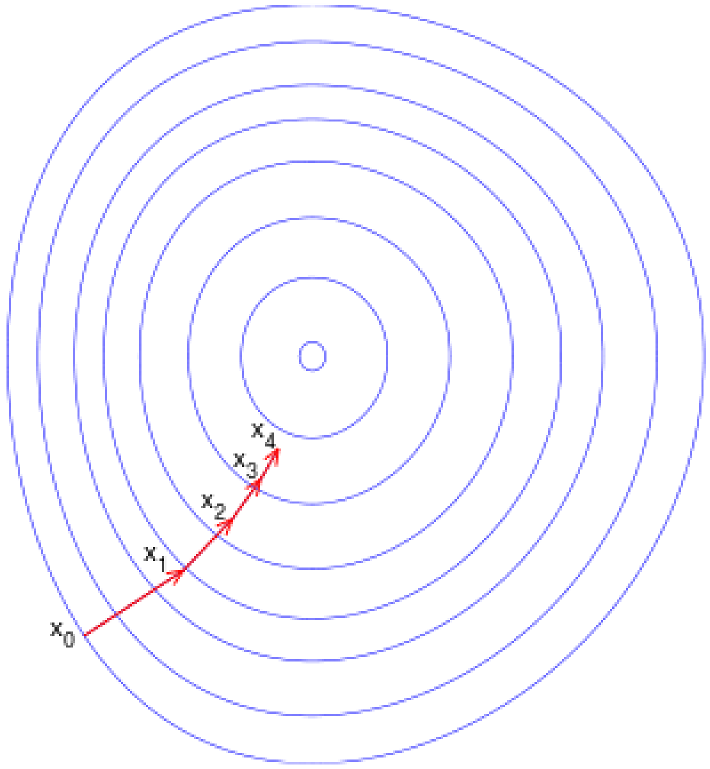
Learning rate



[Image Source: http://sebastianraschka.com/]

Visualization of Gradient Descent



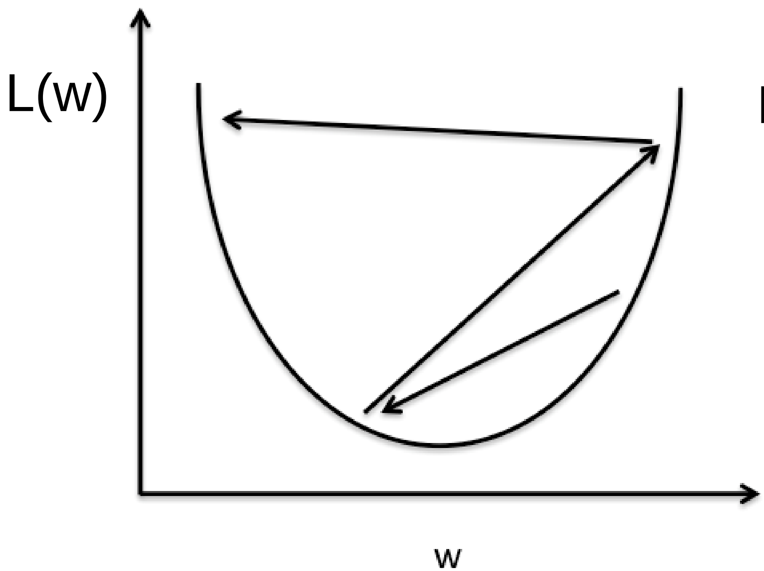


[Image Source: Wikipedia]

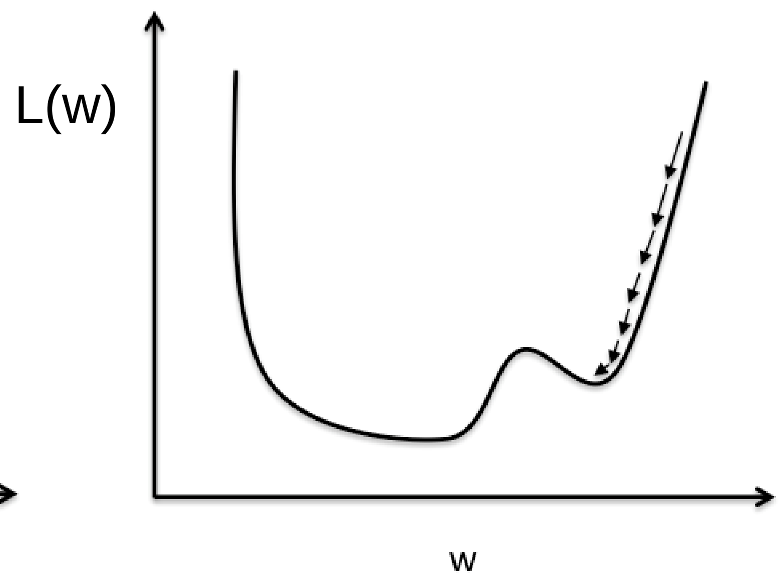
Learning Rate

- Many algorithms designed to set the learning rate
  - Momentum, RMSProp, **Adam**, etc.

In PyTorch: `torch.optim.Adam`



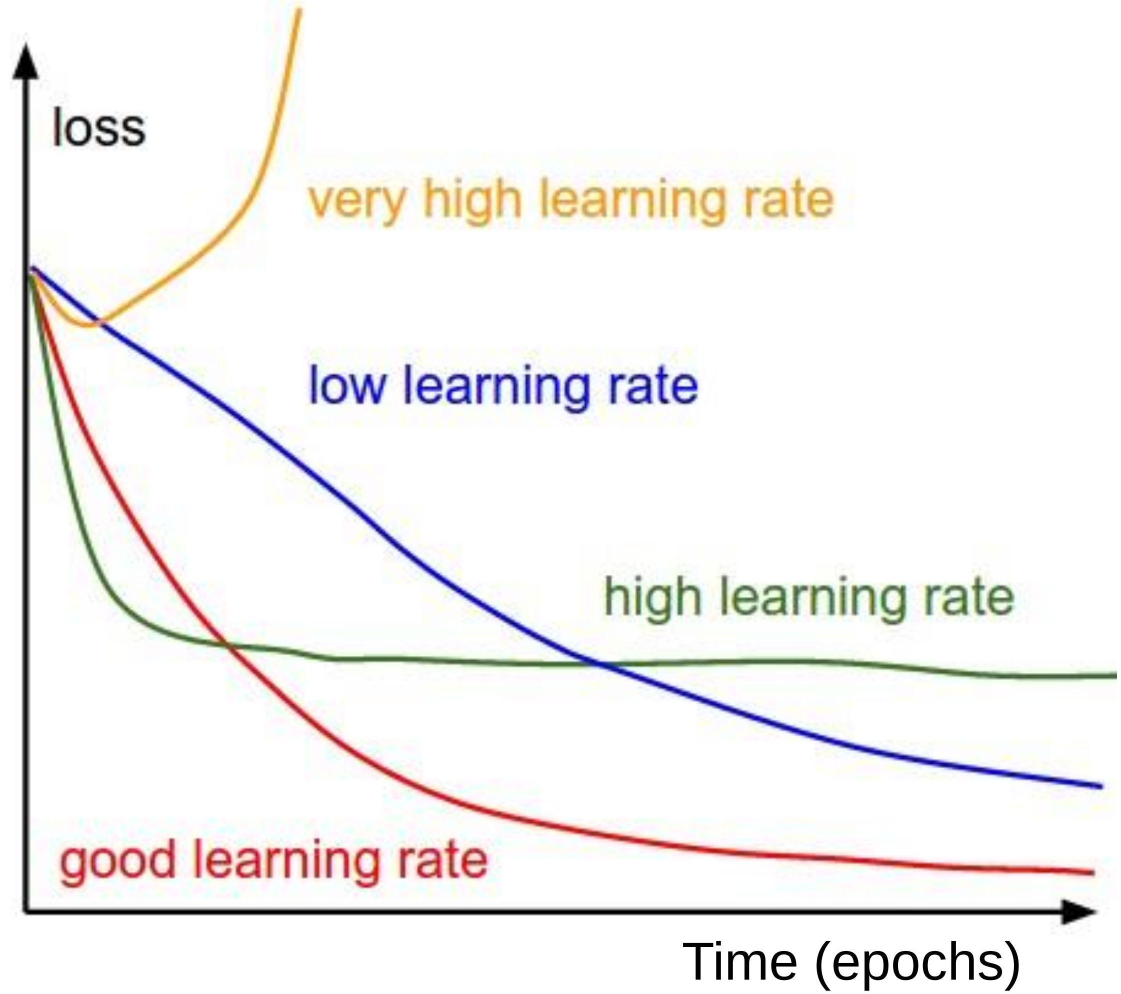
Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.

[Image Source: http://sebastianraschka.com/]

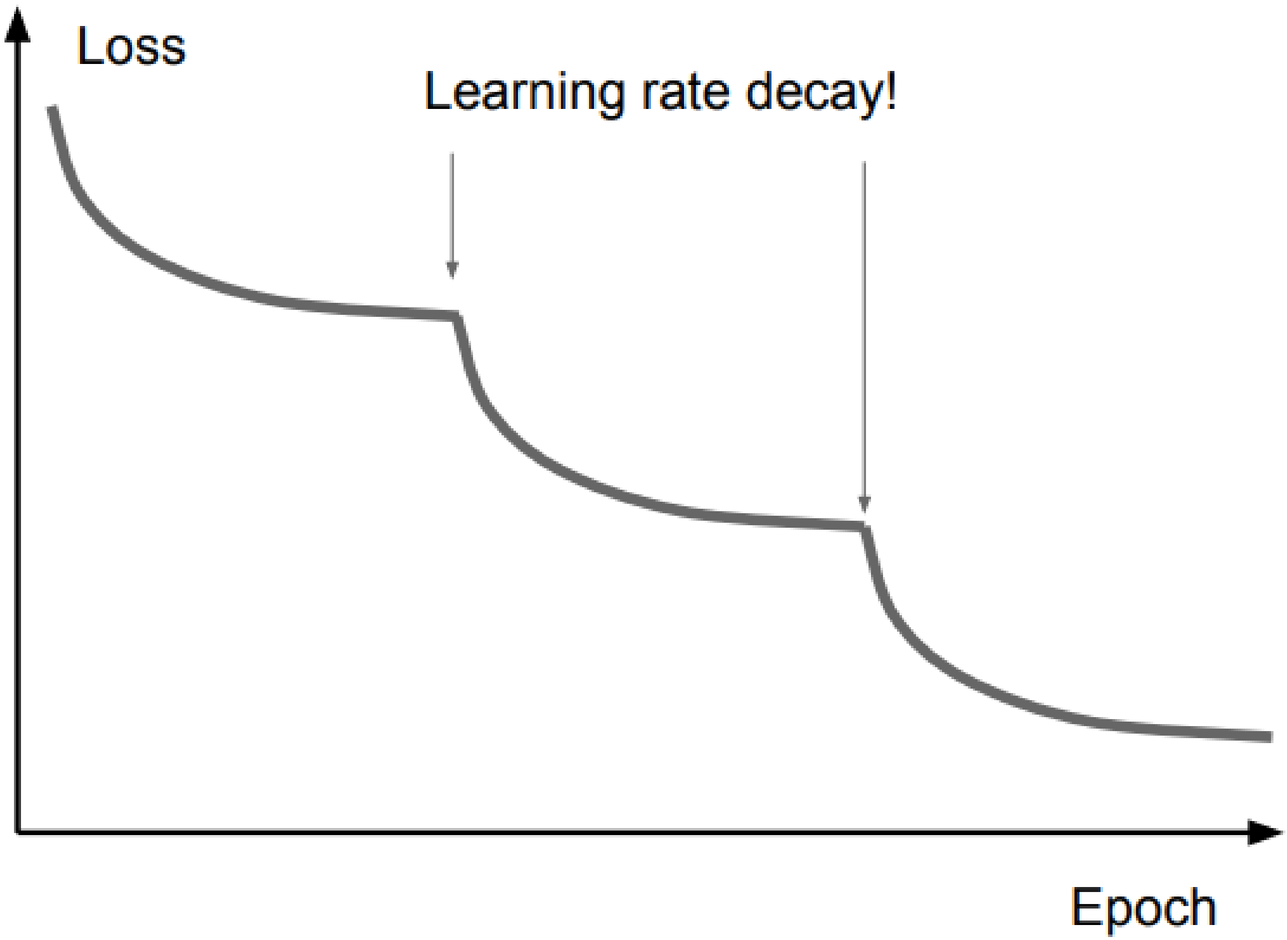
Impact of Learning Rate



Can also decay learning rate over time for faster convergence

[Image source: http://blogs.sas.com/]

Learning Rate Decay



[Image Source: Stanford cs231n]

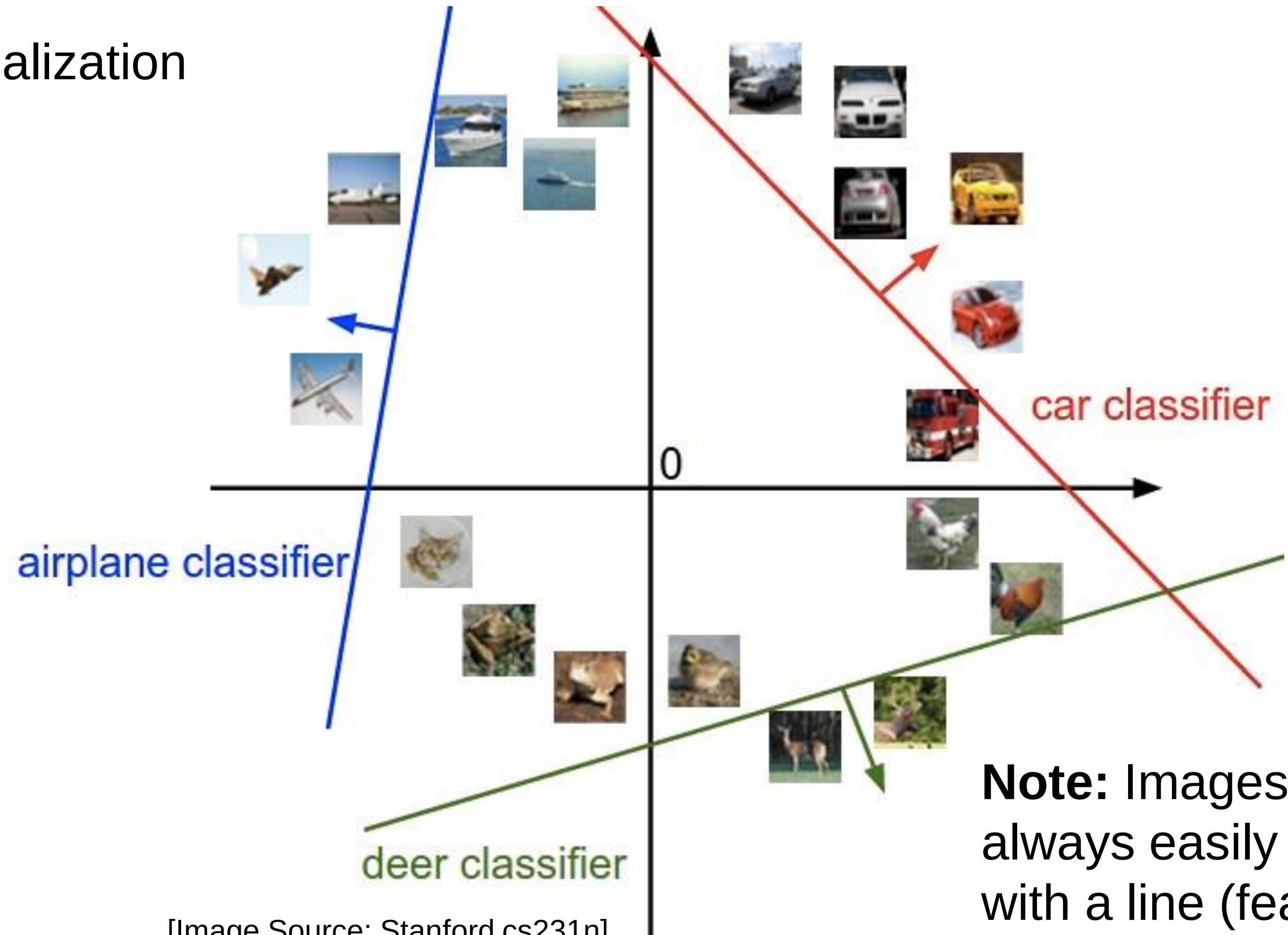
Frequency of Weight Updates

- **Batch Gradient Descent**
  - Update weights after computing loss on the **entire** training set
  - Computationally expensive to compute loss
- **Stochastic Gradient Descent**
  - Update weights after computing loss on a **single** training example; shuffle examples after going through entire training set
  - Fast, but might go in the wrong direction (noisy)
- **Mini-batch Gradient Descent**
  - Divide training set into smaller sets called **mini-batch**, and update weights based on loss of each mini-batch (a.k.a. 'batch')
- Each pass through the entire training set is referred to as an **epoch**

Intuition of Classifier

Images are points in high dimensional space
(e.g. CIFAR images in 3072-dimensional space)

2-D Visualization



[Image Source: Stanford cs231n]

Note: Images not always easily separated with a line (features!)

Feature Extraction

- Use **features** rather than pixels as input into the classifier
- **Feature extraction** can be thought of as transforming pixels into a space where the images can more easily separated by the classifier
  - The transformation can be non-linear

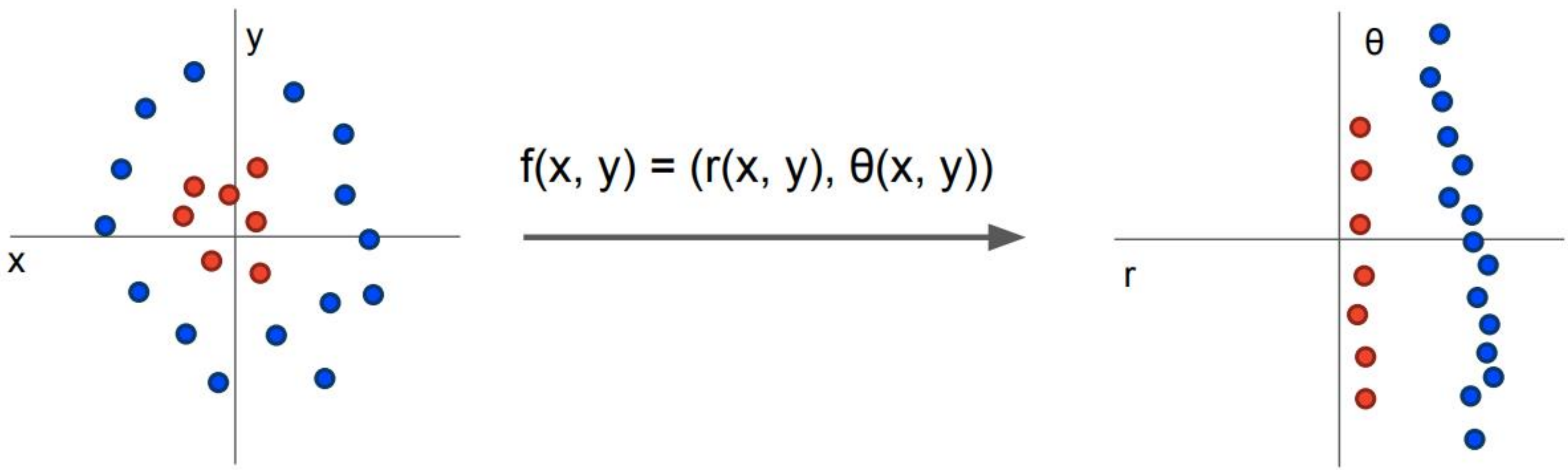
Perform feature extraction before classification

$$f_{class}(\mathbf{x},\mathbf{W},\mathbf{b}) \Rightarrow f_{class}(f_{extract}(\mathbf{x}),\mathbf{W},\mathbf{b})$$

Image

→ Feature Extraction → Classifier **W,b** → **Cat**

Feature Extraction



Cannot separate red and blue points with linear classifier

After applying feature transform, points can be separated by linear classifier

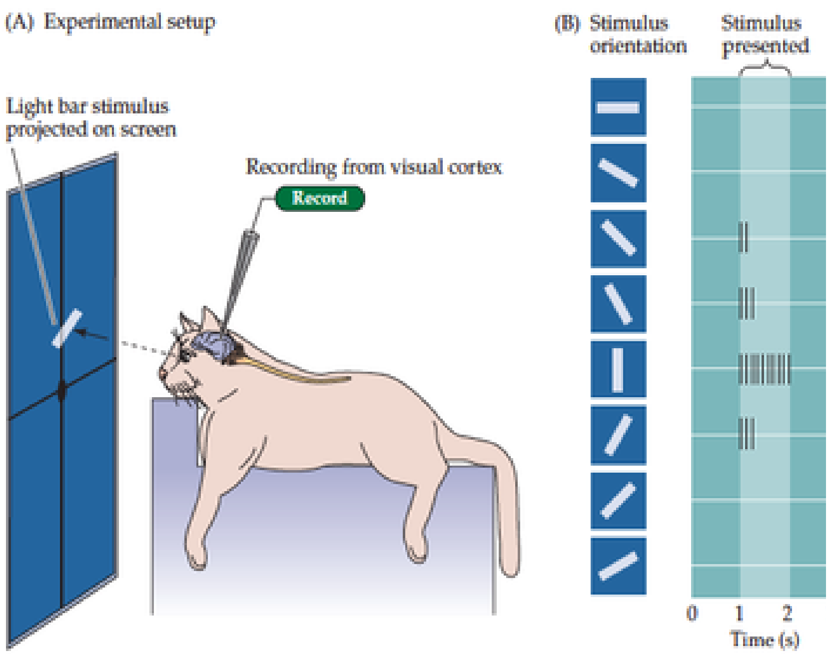
[Source: Stanford cs231n]

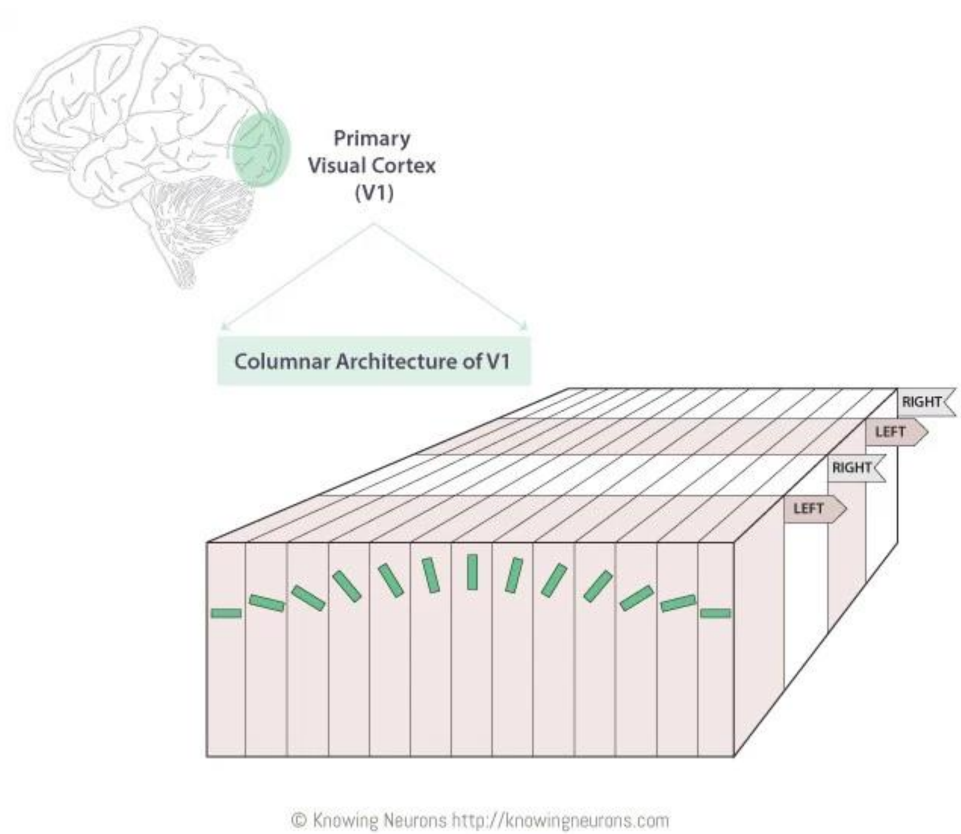
Example Hand-Crafted Features

Edges contain a lot of information

Example Hand-Crafted Features

V1 cells in the primary visual cortex are sensitive to edges

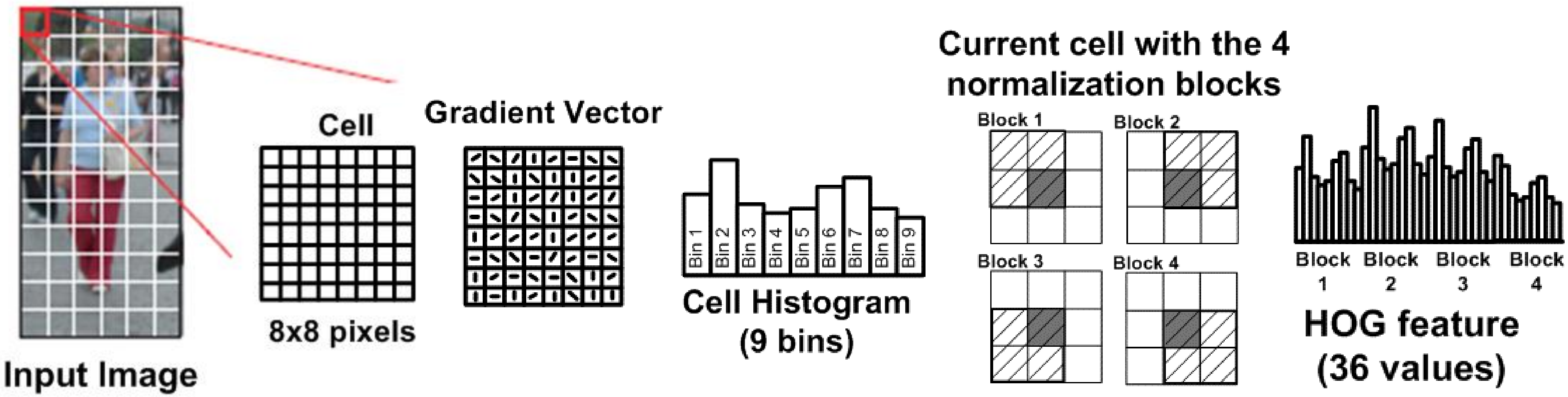




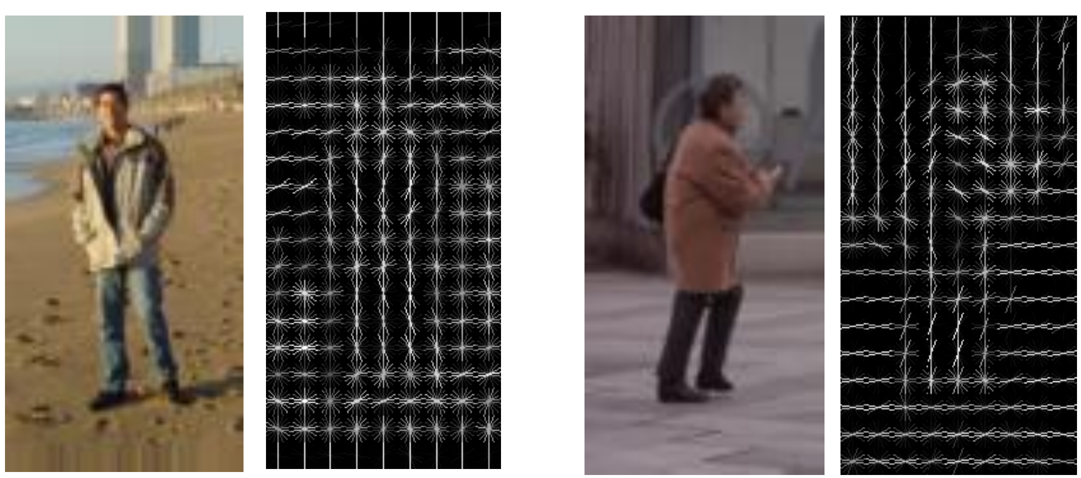
Hubel and Wiesel (1950s – Nobel Prize): https://youtu.be/Cw5PKV9Rj3o

Example Hand-Crafted Features

Histogram of Oriented Gradients (HOG)



Examples



Learned Weights

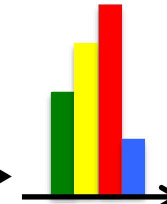
[Dalal & Triggs, CVPR 2005]

Classification Pipeline (Inference)

Image

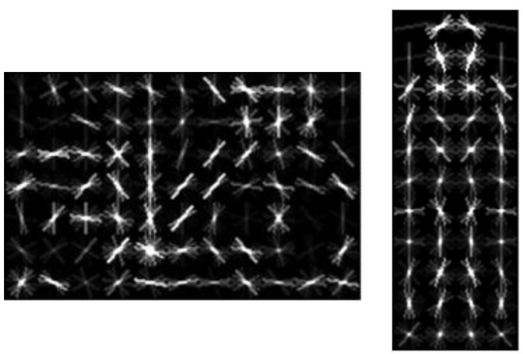
pixels → **Feature Extraction** → Features (**x**) → **Classification** ***(w^Tx)*** → Scores

Trained weights (**w**)



Scores per class
(select class based on max or threshold)

Handcrafted Features (e.g. HOG)

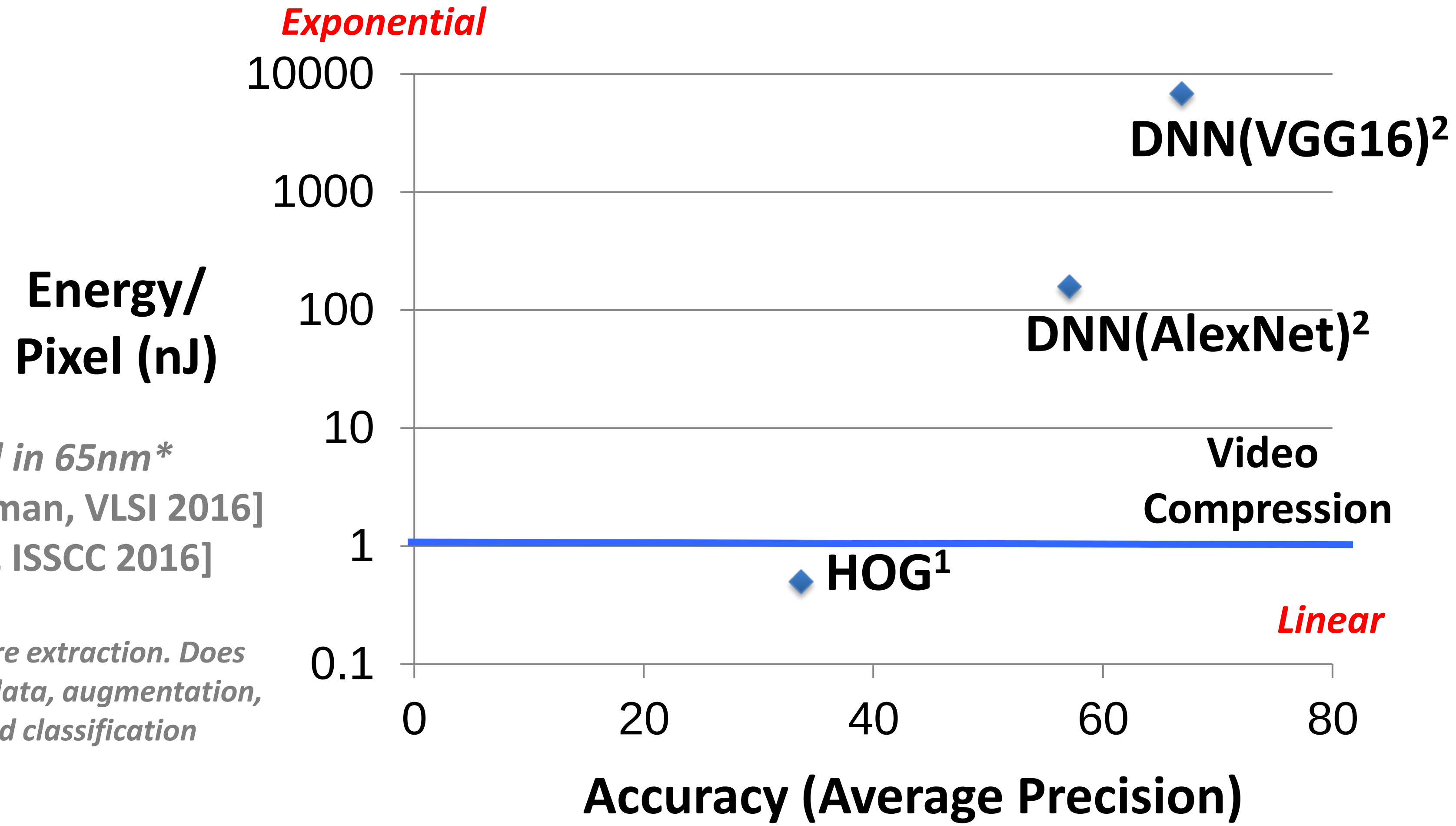


Learned Features (e.g. DNN)

In PyTorch: `torch.nn.Conv2d`

$$\textbf{Score} = \Sigma_n x_i w_i$$

Features: Energy vs. Accuracy



*Measured in 65nm**
1. [Suleiman, VLSI 2016]
2. [Chen, ISSCC 2016]

** Only feature extraction. Does not include data, augmentation, ensemble and classification energy, etc.*

Measured in on VOC 2007 Dataset
1. DPM v5 [Girshick, 2012]
2. Fast R-CNN [Girshick, CVPR 2015]

[Suleiman et al., ISCAS 2017]

Summary

- Image Classification Task
  - Input: Image → Output: label (class scores)
- Steps to training and testing a classifier
  - regularization
- Example of a simple linear classifier
- Feature extraction

PyTorch Summary

- Dataset: torchvision.datasets, torch.utils.data.DataLoader
- Construct model: torch.nn
  - Linear layer: torch.nn.Linear
  - Feature extraction: torch.nn.Conv2d
  - Activations: torch.nn.ReLU
- Train the model:
  - Loss function: torch.nn.CrossEntropyLoss
  - Optimizer: torch.optim.Adam
- One training step:
  - output = model(input)
  - loss = loss_fn(output, target)
  - optimizer.zero_grad()
  - loss.backward()
  - optimizer.step()

https://github.com/pytorch/examples/blob/master/mnist/main.py

https://pytorch.org/tutorials/

Key Concepts and Terms

- Image Classification
- Training, Testing, Validation
- Linear Classifier → Weights and Bias
- Loss function → Softmax
- Generalization, Overfitting
- Regularization
- Hyper-parameters
- Epoch, Batch
- Gradient Descent, Learning Rate, Adam
- Feature Extraction

In Lab 1 and walk through the PyTorch code to see if you can identify these concepts

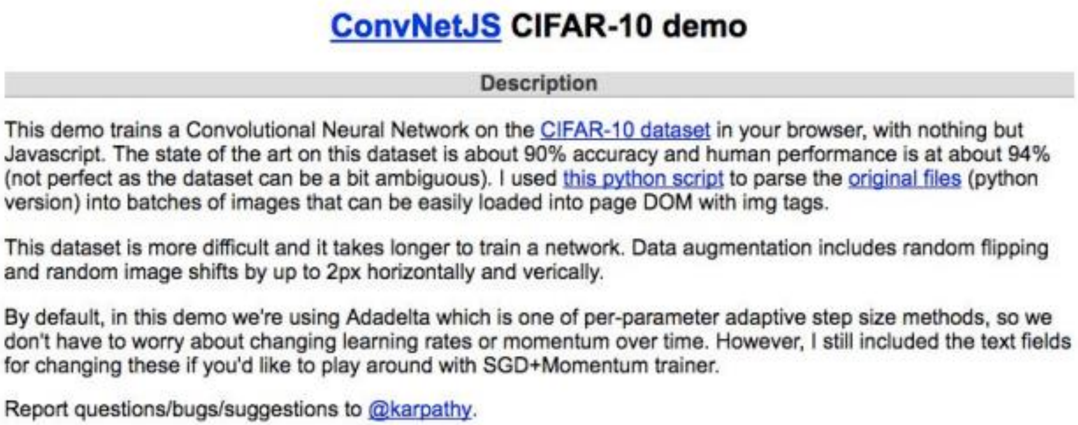
References

- For a more in-depth treatment, please see
  - MIT's Machine Learning Courses (6.036/6.876)
    - https://introml.mit.edu/
  - MIT's Computer Vision Course (6.819/6.869)
    - http://6.869.csail.mit.edu/fa18/
  - Class notes from Stanford's CNN Course (cs231n)
    - http://cs231n.stanford.edu/syllabus.html
    - http://cs231n.github.io/classification/
    - http://cs231n.github.io/linear-classify/

References

- Textbook: Chapters 1 & 2
  - https://www.morganclaypool.com/doi/abs/10.2200/S01004ED1V01Y202004CAC050
- Stanford cs231n
  - http://cs231n.github.io/classification/
  - http://cs231n.github.io/linear-classify/
- http://www.deeplearningbook.org/
  - Chapter 5 http://www.deeplearningbook.org/contents/ml.html
- Other Works Cited in Recitation
  - CIFAR-10 Dataset: https://www.cs.toronto.edu/~kriz/cifar.html
  - L. Zitnick, "Which way forward? AI + vision," CVPR Workshop, 2017
  - A. Suleiman*, Y.-H. Chen*, J. Emer, V. Sze, "Towards Closing the Energy Gap Between HOG and CNN Features for Embedded Vision," IEEE International Symposium of Circuits and Systems, 2017.
  - N. Dalal, B. Triggs, "Histograms of oriented gradients for human detection," Computer Vision and Pattern Recognition, 2005

Demo of CIFAR-10 CNN Training



ConvNetJS CIFAR-10 demo

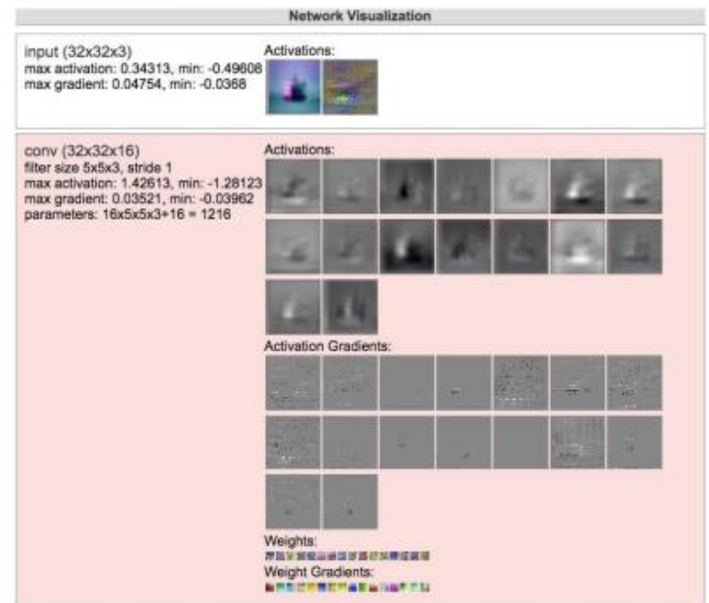
Description

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and verically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.



http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html