Applied Computer Vision

David Vernon
Carnegie Mellon University Africa

vernon@cmu.edu
www.vernon.eu

Lecture 27

Computer Vision and Deep Learning II

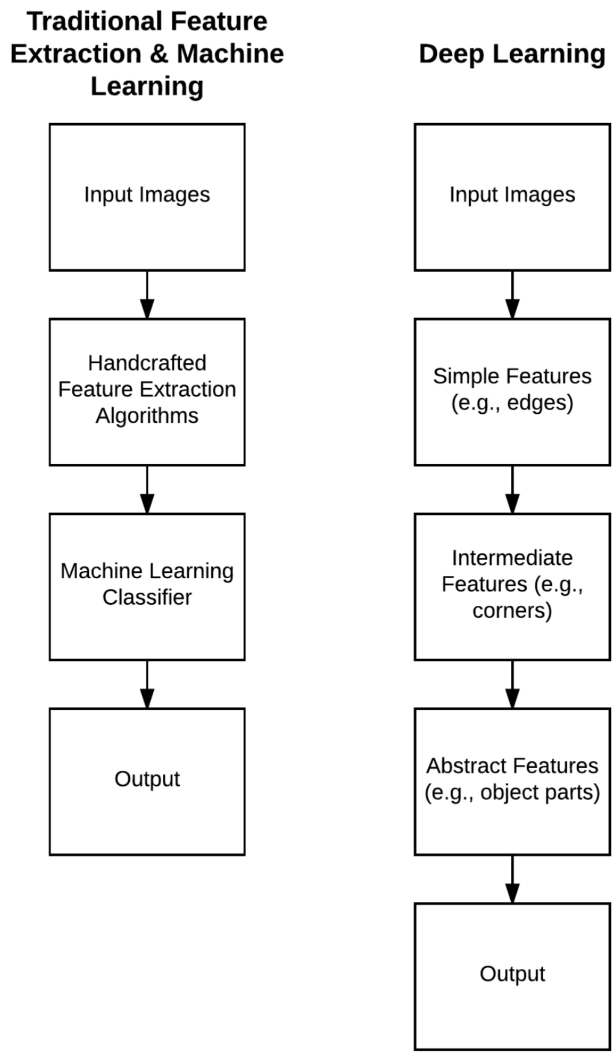
Based mostly on material in

Deep Learning for Computer Vision with Python,
A. Rosebrock, PyImageSearch, 2017

Tools: Python, OpenCV, Keras, TensorFlow

https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/

Machine Learning vs. Deep Learning



Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Deep Learning

How deep is deep?

“How many layers does a neural network need to be considered *deep?”*

No consensus … But

“My personal opinion is that any network with greater than two hidden layers can be considered ‘deep’.“

Adrian Rosebrock

Deep Learning

Why did artificial neural networks not take off during the 1990s?

1. Our labelled datasets were thousands of times too small

2. Our computers were millions of times too slow

3. We initialized the network weights in a stupid way

4. We used the wrong type of nonlinearity activation function

Geoffrey Hinton. What Was Actually Wrong With Backpropagation in 1986?
https://www.youtube.com/watch?v=VhmE_UXDOGs. 2016

Deep Learning

However, today we have:

1. Faster computers

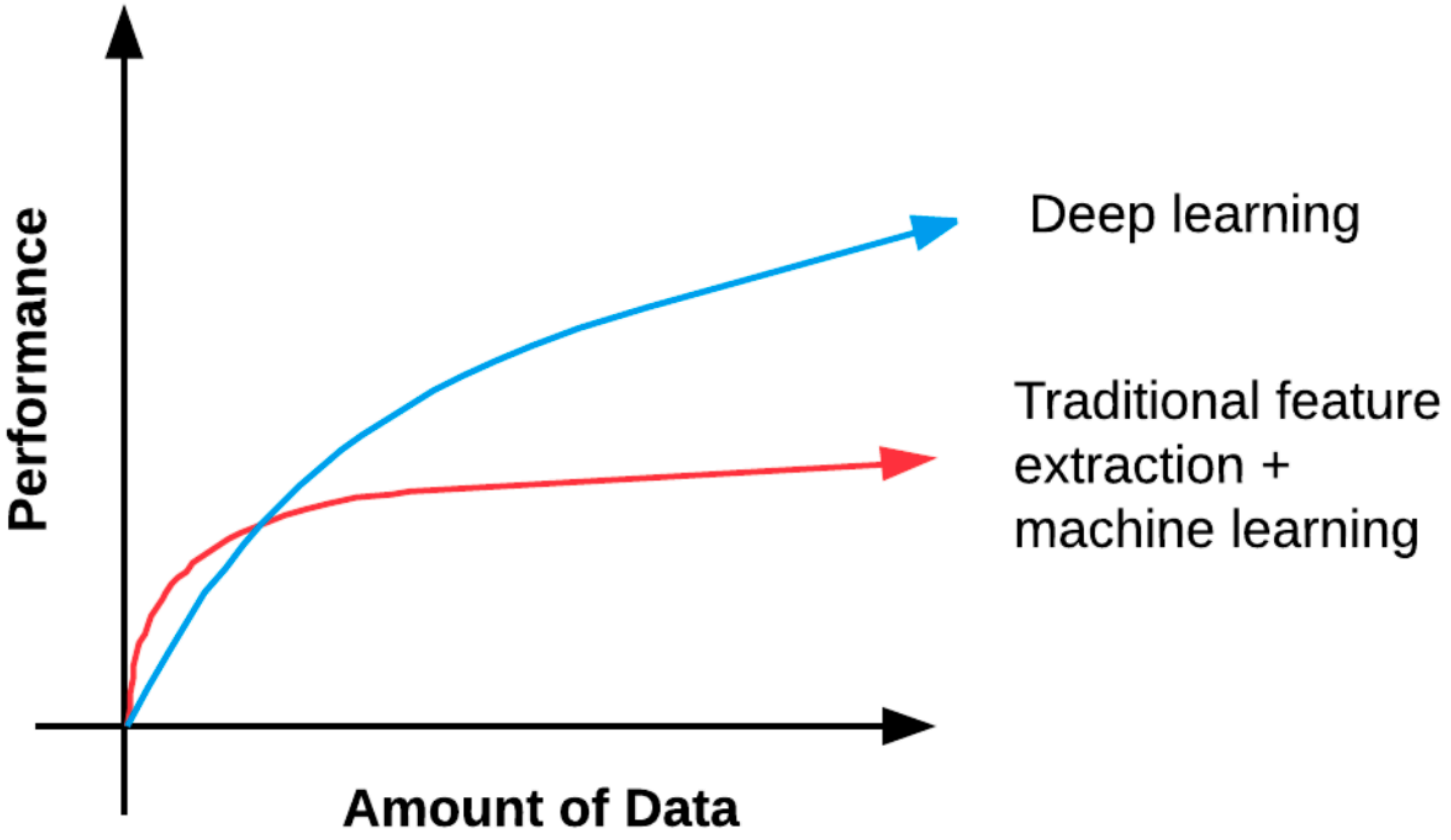
2. Highly optimized hardware (i.e., GPUs)

3. Large, labelled datasets in the order of millions of images

4. A better understanding of weight initialization functions and what does/does not work

5. Superior activation functions

Deep Learning



Andrew Ng. *What data scientists should know about deep learning*. https : / / www . slideshare.net/ExtractConf. 2015

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Deep Learning

Four steps in deep learning:

1. Create or acquire your data set

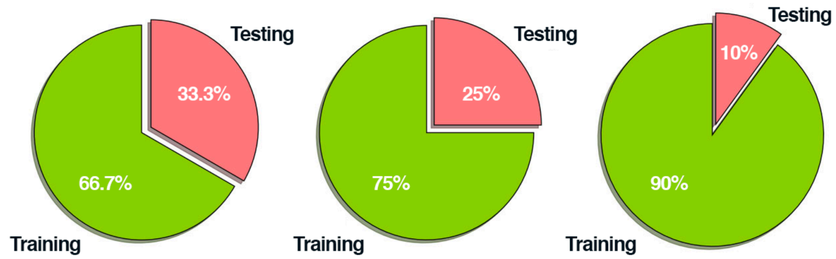
   – Typically, 1000 images per class/category

2. Split your data set

   – Training set
   – Validation set
   – Testing set

Must be independent

Used to adjust hyperparameters
Typically, 10-20% of the data set



   – Needs just a single line of code using the scikit-learn library

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Deep Learning

3. Train your network

   – Gradient descent, usually stochastic gradient descent (SGD)

4. Evaluate

   – Compare model predictions with ground truth from test data set

   – Compute metrics to quantify performance: precision, recall, and f-measure

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Deep Learning

Goal: a model that can generalize

- Perform well on data that is not part of the training, validation, test data sets
- Avoid over-fitting
    - excellent performance on training set
    - poor performance on test set

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Datasets for Image Classification

MNIST

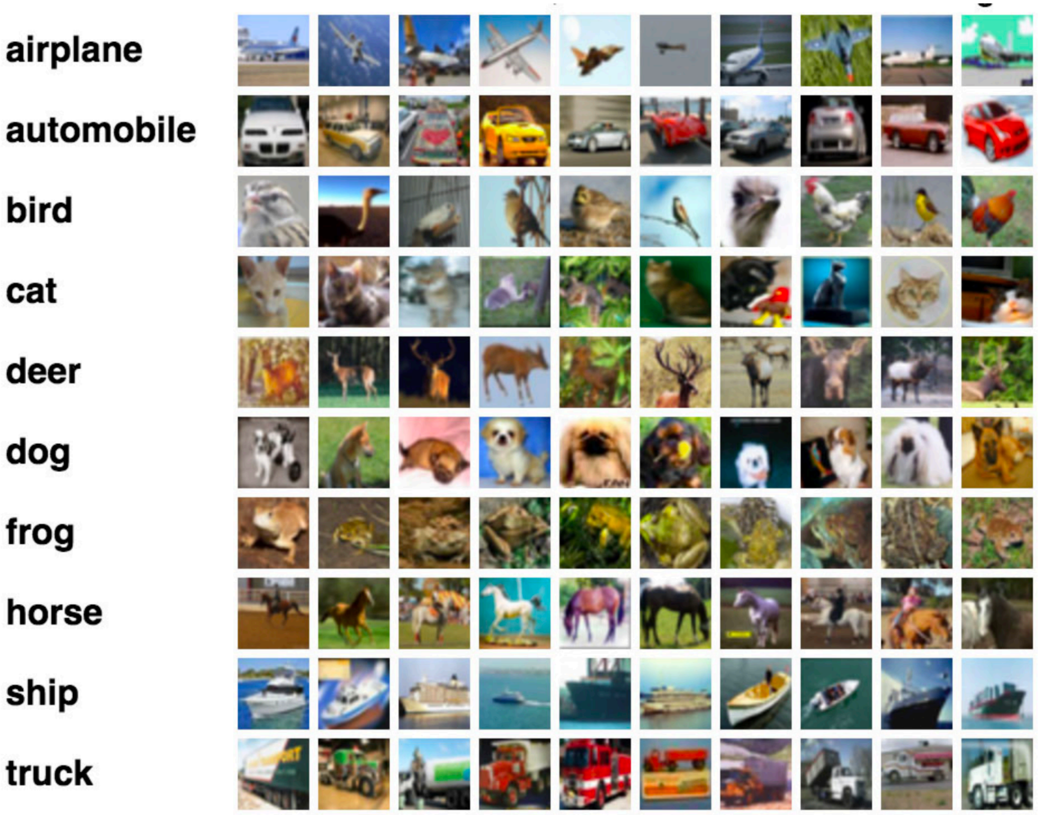
- Modified National Institute of Standards and Technology
- 60,000 training images; 10,000 testing images
- 28 x 28 greyscale
- Black background, grey to white foreground
- Easy to obtain 97% classification accuracy

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Datasets for Image Classification

CIFAR-10

- 60,000 images
- 32 x 32 x 3 (RGB) … 3072 element feature vector
- 10 classes: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks



Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Datasets for Image Classification

SMILES

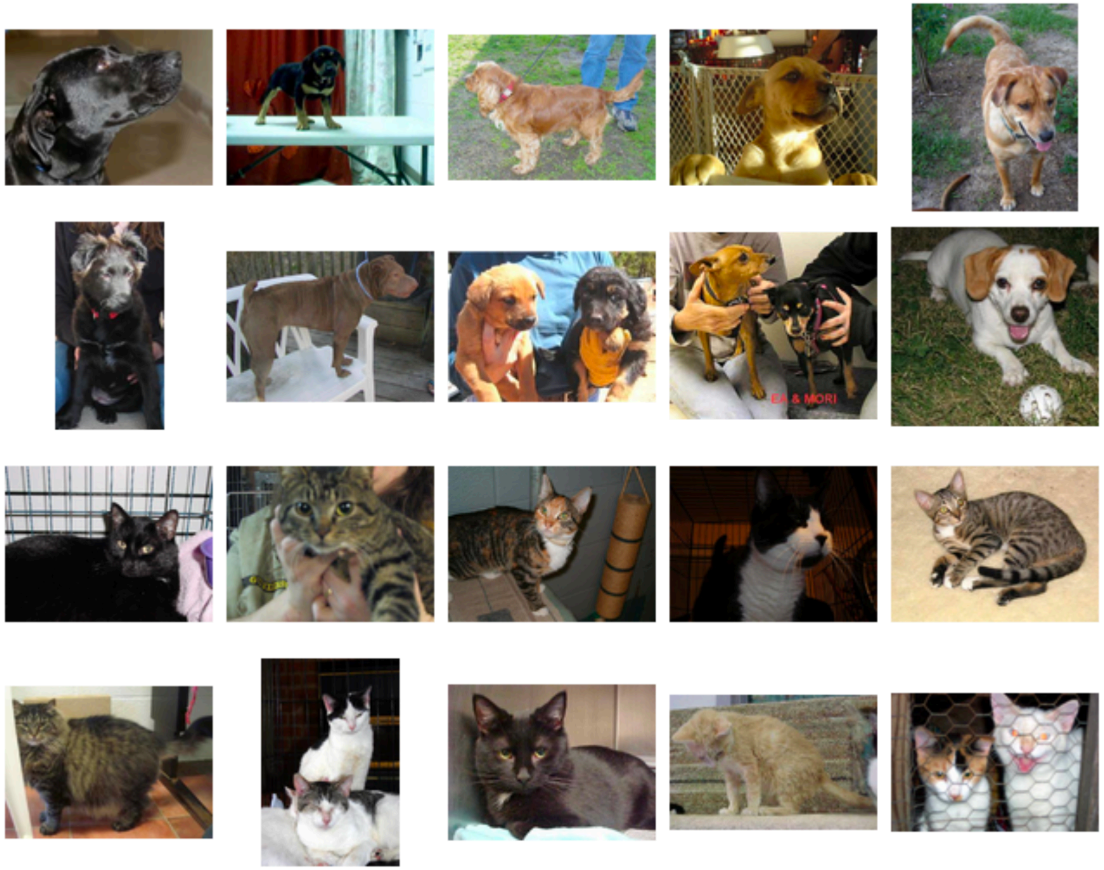
- Daniel Hromada. https://github.com/hromi/SMILEsmileD
- Preprocessed to ensure that only relevant data is present.

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Datasets for Image Classification

Kaggle Competition: Dogs vs. Cats

- 25,000 images
- Varying image resolution



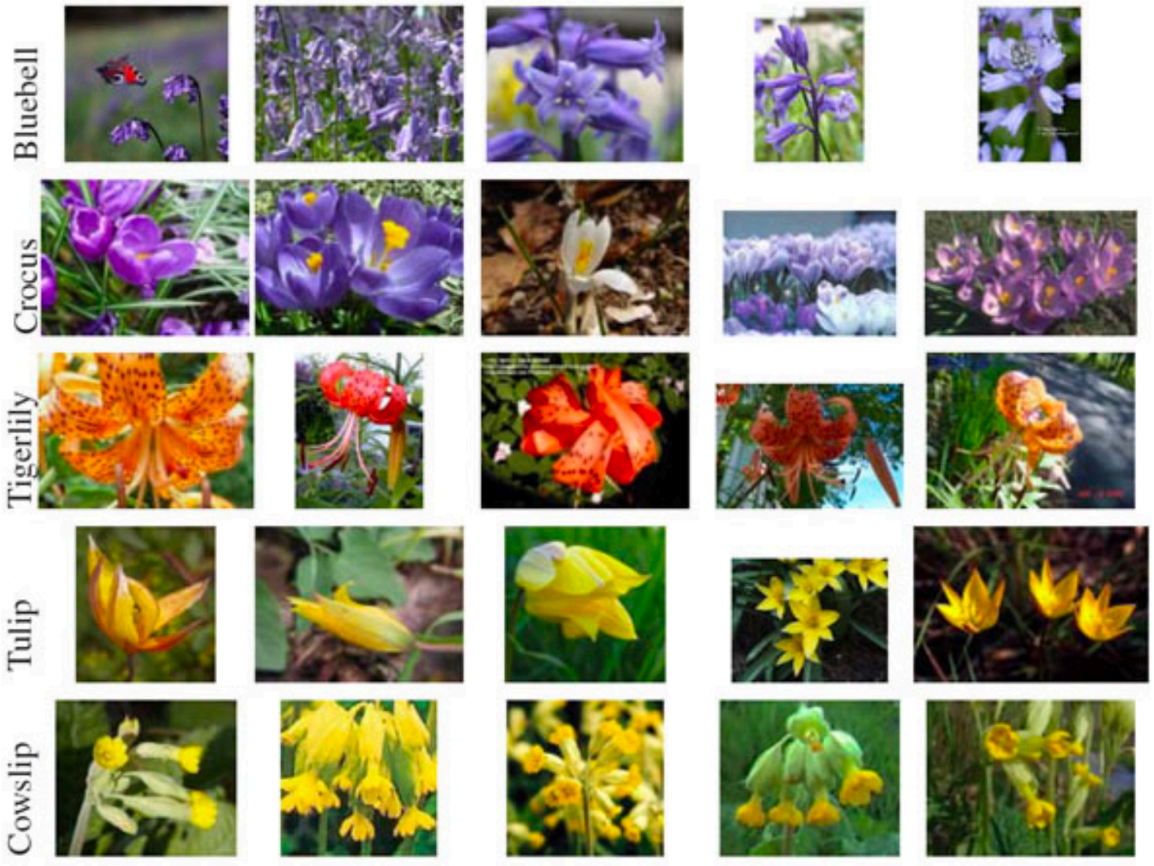
Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Datasets for Image Classification

Flowers-17

- 17 classes
- 80 images per class
- Challenging:
  - Variation in scale
  - Variation in viewpoint
  - Variation in lighting
  - Only 80 image per class

- Rule of thumb: 1,000 – 5000 images per class when training a deep neural network



Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Datasets for Image Classification

Adience

- 26,580 images
- Age and gender prediction

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Datasets for Image Classification

Stanford Cars Dataset

– 16,185 images
– 196 vehicle makes and models categories

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Datasets for Image Classification

Kaggle: Facial Expression Recognition Challenge FER-13

- 35,888 images
- 7 categories
  1. Anger
  2. Disgust
  3. Fear
  4. Happy
  5. Sad
  6. Surprise
  7. Neutral
- Disgust and Fear sometimes combined due to class imbalance

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Datasets for Image Classification

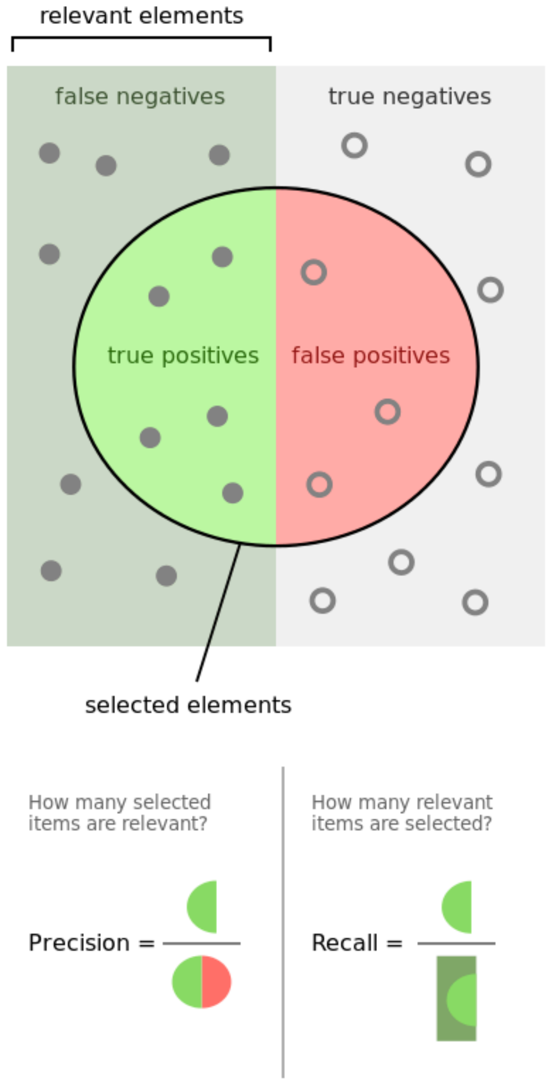
Others

- Tiny ImageNet 200
  - 200 categories; 500 training images, 50 validation, 50 testing, per class
- CALTECH-101
  - 8,677 images; 101 categories
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
  - 1,000 categories, 1.2 million training images, 50,000 validation, 100,000 testing

Classifier Performance Metrics

- Precision
  The number of correct positive results divided by the number of all positive results returned by the classifier

- Recall
  The number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive)

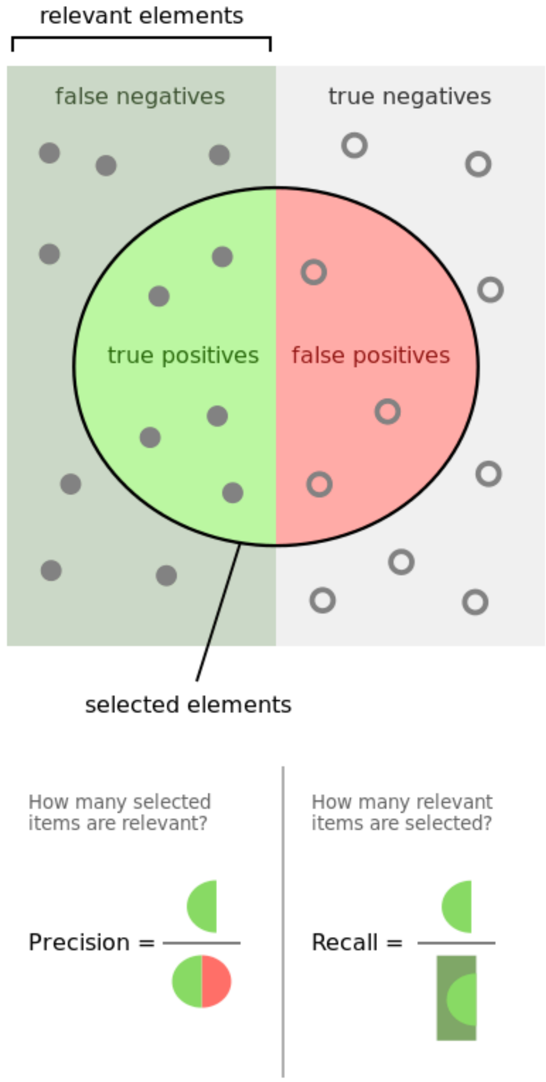


Credit: https://en.wikipedia.org/wiki/F1_score

Classifier Performance Metrics

- F1-Score
  The harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

$$F_1 = \left(\frac{\text{recall}^{-1} + \text{precision}^{-1}}{2} \right)^{-1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$



Credit: https://en.wikipedia.org/wiki/F1_score

Parametric Models & Parameterized Learning

"A learning model that summarizes data with a set of parameters of fixed size (independent of the number of training examples) is called a parametric model.

No matter how much data you throw at the parametric model, it won't change its mind about how many parameters it needs."

Russell and Norvig (2009)

Parametric Models & Parameterized Learning

Four components of parameterized learning

1. Data
2. Scoring Function
3. Loss Function
4. Weights and Biases

Parametric Models & Parameterized Learning

Four components of parameterized learning

1. Data

- Data point (intensity images, feature images)
- Class labels
- Multi-dimensional design matrix X
  - Each row x_i represents a data point (e.g. n x m x 3 RGB pixels of image i)
  - Each column corresponds to a different feature
- Vector y, where y_i provides the class label for the i-th example in the data set

Parametric Models & Parameterized Learning

Four components of parameterized learning

2. Scoring Function $f(x_i)$
   - Maps data to class labels
   - `INPUT_IMAGES => F(INPUT_IMAGES) => OUTPUT_CLASS_LABELS`

Parametric Models & Parameterized Learning

Four components of parameterized learning

3. Loss function L

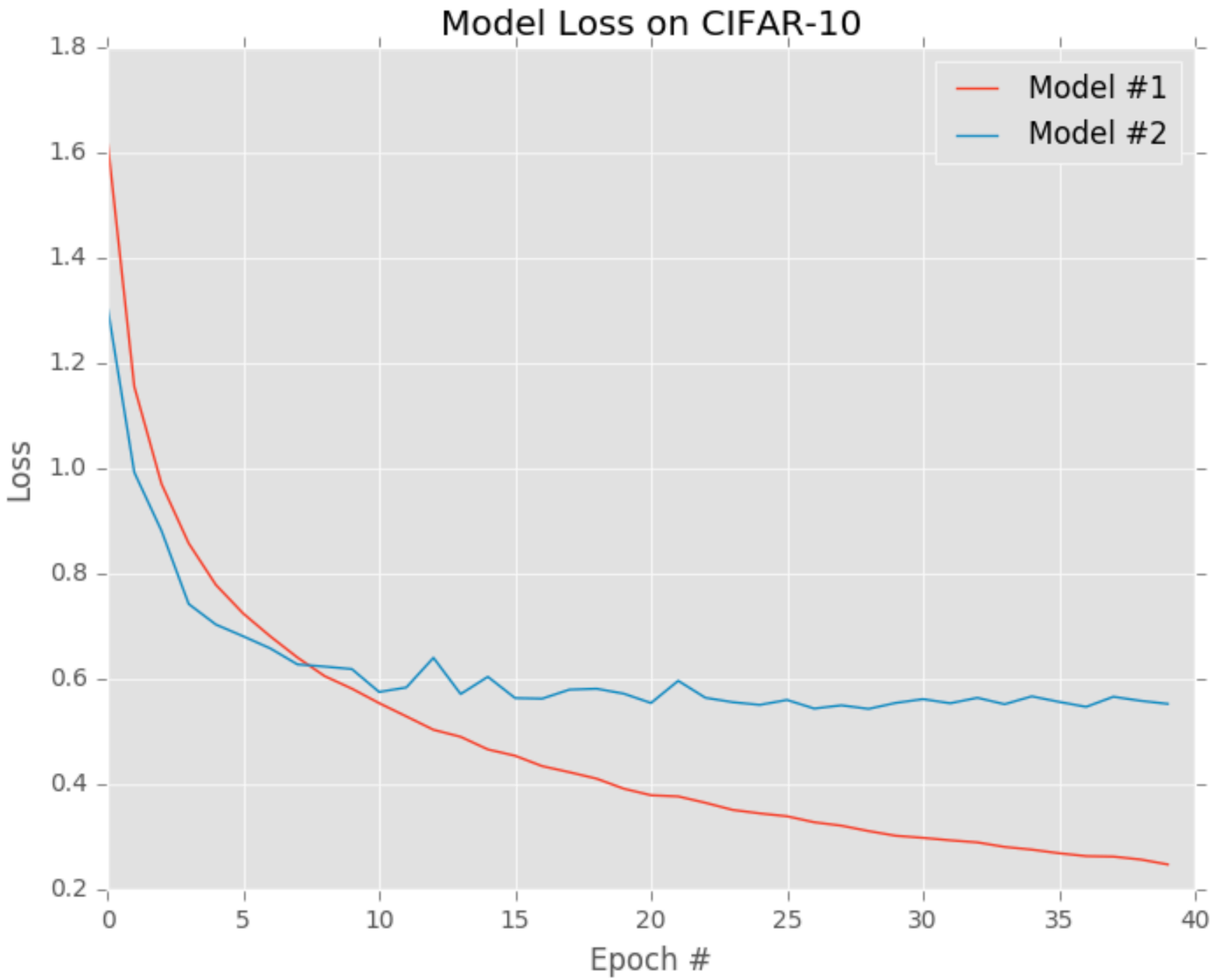
    - Quantifies how well predicted class labels agree with ground-truth labels

    - Low loss implies high level of agreement (and vice versa)

    - Goal of training is to minimize the loss function

        - And, hence, increase classification accuracy

Parametric Models & Parameterized Learning



Parametric Models & Parameterized Learning

Four components of parameterized learning

4. Weights and Biases

- Weight matrix W
- Bias vector b

Allows us to shift the scoring function without influencing the weight matrix

- They are parameters of the scoring function

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

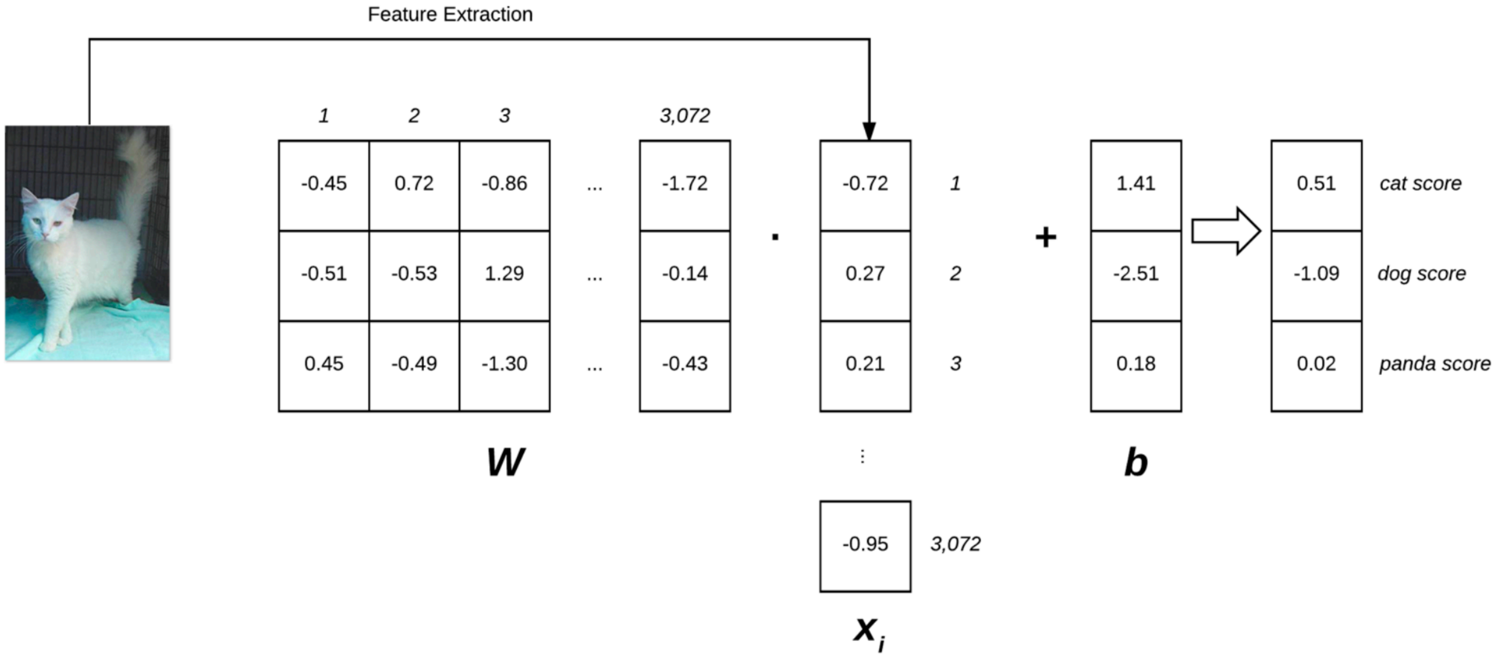
e.g. a simple linear classifier
(not a neural network)

Parametric Models & Parameterized Learning

Four components of parameterized learning

4. Weights and Biases

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



Parametric Models & Parameterized Learning

Four components of parameterized learning

4. Weights and Biases

- For simplicity, omit b and write as follows

$$s = f(x_i, W)$$

- Thus, we can get the predicted score of the j-th class using the i-th data point

$$s_j = f(x_i, W)_j$$

Parametric Models & Parameterized Learning

Four components of parameterized learning

4. Weights and Biases

- We can get also the predicted score of the y_i-th class (the correct class)

  s_{y_i}

- Goal of training is to find the W and b that minimizes the loss function for some test image x_i

Optimization

Parametric Models & Parameterized Learning

Advantages

- Size of the $\overbrace{\text{weight matrix and bias vector}}^{\text{model}}$ $\ll$ size of training data set
- Classifying new test data is fast

Cross-entropy Loss and Softmax Classifiers

Softmax classifiers provide probabilities for each class

- The softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sum to one

$$\overbrace{e^{s_{y_i}} \Big/ \underbrace{\sum_j e^{s_j}}_{\text{Normalization}}}^{\text{Normalized probability of correct label}}$$

Unnormalized probability

- In other words, the softmax function produces a probability distribution

See http://cs231n.github.io/linear-classify/ for the derivation of this expression

Cross-entropy Loss and Softmax Classifiers

Cross-entropy loss

- Cross entropy indicates the distance between what the model believes the output distribution should be, and what the original distribution really is

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

Loss for one data point (i.e. image)

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i$$

Loss for all images in the training, validation, or test sets

- Loss functions almost always have an additional regularization term

See http://cs231n.github.io/linear-classify/ for the information theory and probability theory views

Cross-entropy Loss and Softmax Classifiers

	Scoring Function
Dog	-3.44
Cat	1.16
Panda	s_{y_i} 3.91

s_j

Input Image

	Scoring Function	Unnormalized Probabilities
Dog	-3.44	0.03
Cat	1.16	3.19
Panda	3.91	49.90

e^{sj}

	Scoring Function	Unnormalized Probabilities	Normalized Probabilities
Dog	-3.44	0.0321	0.0006
Cat	1.16	3.1899	0.0601
Panda	3.91	49.8990	0.9393

$e^{s_{y_i}} / \sum_j e^{s_j}$

	Scoring Function	Unnormalized Probabilities	Normalized Probabilities	Negative Log Loss
Dog	-3.44	0.0321	0.0006	
Cat	1.16	3.1899	0.0601	
Panda	3.91	49.8990	0.9393	**0.0626**

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

Repeat this for all images in the training set, Take the average to obtain the overall cross-entropy loss for the training set $L = \frac{1}{N}\sum_{i=1}^{N} L_i$

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

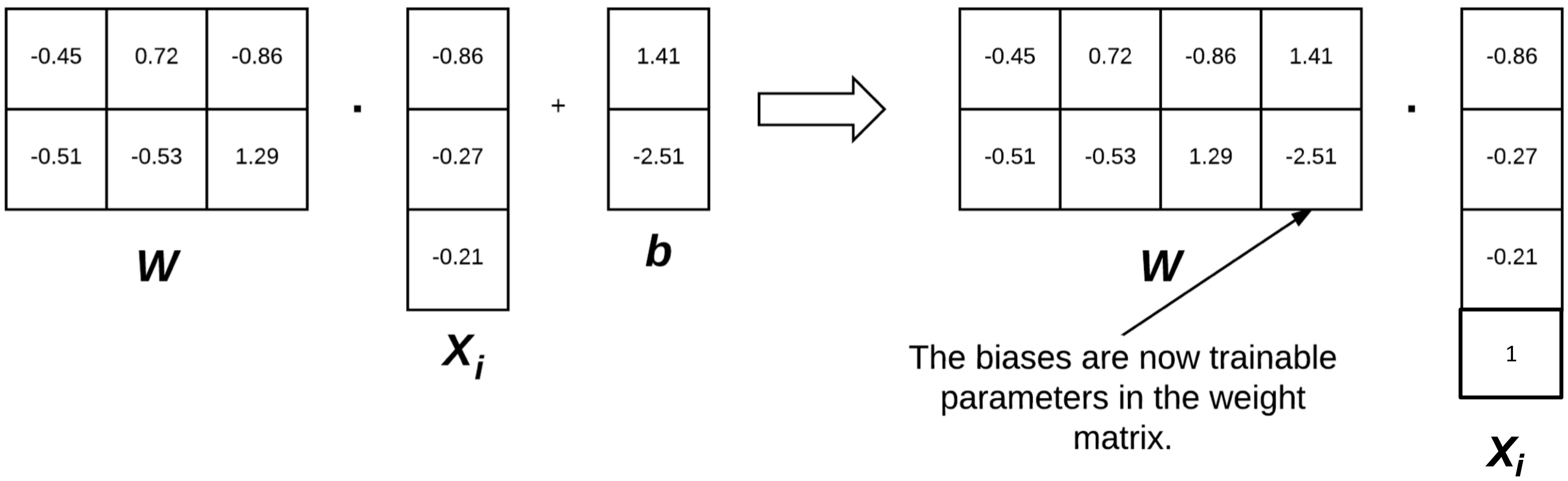
Optimization Methods and Regularization

Aside: The Bias Trick

- Recall the scoring function $f(x_i, W, b) = W x_i + b$
- We can combine W and b to avoid having to keep track of two separate variables
- We add an extra column to the input data X with constant value 1 that corresponds to the bias variable
- This allows us to rewrite our scoring function $s = f(x_i, W) = W x_i$
- Treat the bias as a learnable parameter within the weight matrix

Optimization Methods and Regularization

Aside: The Bias Trick

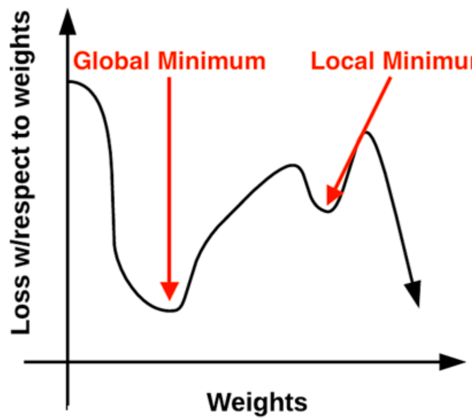


Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Optimization Methods and Regularization

Gradient Descent – Iterative Optimization

- Iteratively evaluate your parameters
- Compute your loss
- Take a small step in the direction that will minimize your loss using the gradient of of the loss function



Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Optimization Methods and Regularization

Gradient Descent – Iterative Optimization

- Iteratively evaluate your parameters
- Compute your loss
- Take a small step in the direction that will minimize your loss using the gradient of of the loss function

```
while True:
     Wgradient = evaluate_gradient(loss, data, W)
     W += -alpha * Wgradient
```

$$W = W - \alpha \nabla_W f(W)$$

Learning rate α

Controls the size of the step

By far the most important hyperparameter

Too big: bouncing around the loss landscape

Too small: may take far too many iterations (epochs)

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Optimization Methods and Regularization

Stochastic Gradient Descent SGD

- Gradient descent computes the gradient and updates the weight matrix after a computation involving all data points in the training data set in each epoch
- Stochastic gradient descent computes the gradient updates the weight matrix using small batches of training data
- SGD is one of the most important techniques in DL
- Batch size hyperparameter: 32, 64, 128, 256, ….

Optimization Methods and Regularization

Extensions to Stochastic Gradient Descent SGD

- Momentum … to accelerate SGD

$$V = \gamma V - \alpha \nabla_{\mathbf{W}} f(W) \qquad\qquad W = W + V$$

  - Commonly set to 0.9
  - Sometimes set to 0.5 until learning stabilizes and then increase to 0.9

- Nesterov accelerated gradient .

  - Corrective update to the momentum-derived step

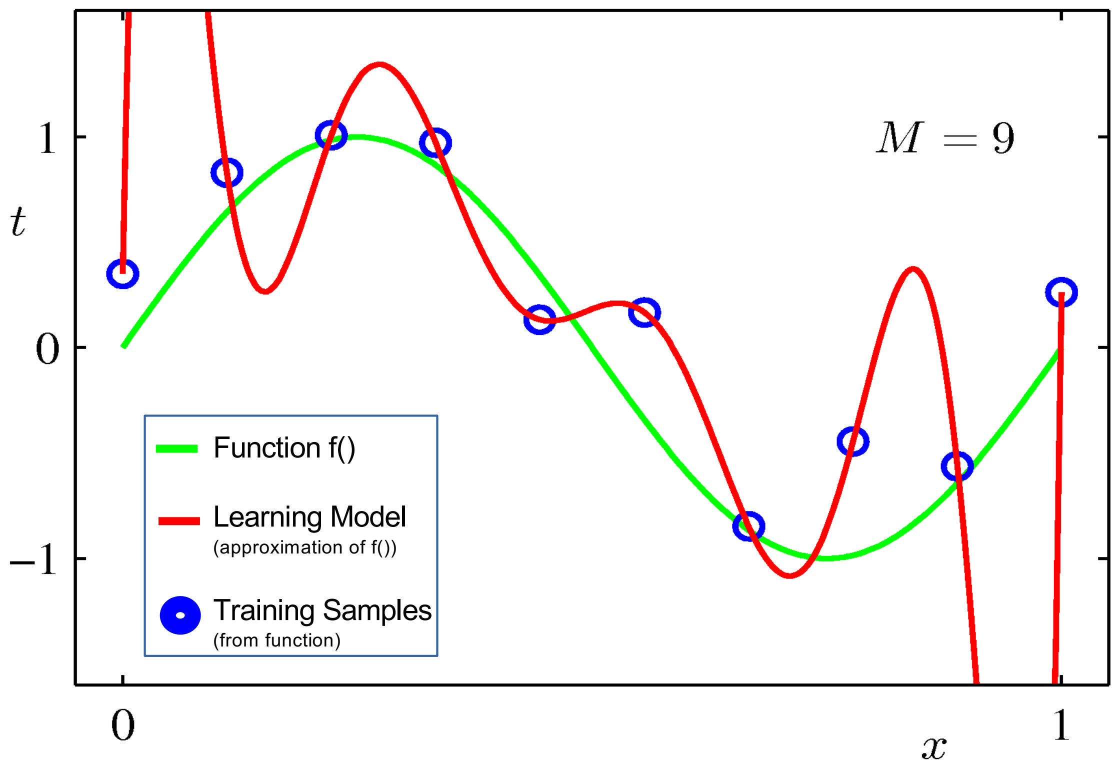
Optimization Methods and Regularization

Regularization

- A way of ensuring the model generalizes well, i.e. good test accuracy
- Possibly at the expense of accuracy with the training set
- Reduces over-fitting
- One of the most important hyperparameters

Optimization Methods and Regularization

Graphical Example: function approximation (via regression)



Source: [PRML, Bishop, 2006]

Credit: Toby Breckon, Durham University

Optimization Methods and Regularization

Regularization

- A way of ensuring the model generalizes well, i.e. good test accuracy
- Possibly at the expense of accuracy with the training set
- Reduces over-fitting
- One of the most important hyperparameters

Optimization Methods and Regularization

Regularization ... recall cross-entropy loss

– If we have a weight matrix W that achieve perfect classification of every image in the training set, the the loss $L = 0$ for all L_i

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

Loss for one data point (i.e. image)

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

Loss for all images in the training, validation, or test sets

– But is W unique? Is there a better W that will improve the models' ability to generalize and reduce overfitting?

Optimization Methods and Regularization

Define a regularization penalty $R(W)$

– If we have a weight matrix W that achieve perfect classification of every image in the training set, the the loss $L = 0$ for all L_i

$$R(W) = \sum_i \sum_j W_{i,j}^2$$

L2 regularization penalty: discourages large weights in W
Spreads "responsibility" for classification more evenly.
i.e. uses all dimensions rather than a few with large values
and thereby reduces overfitting and improves generalization

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} [-log(e^{s_{y_i}} / \sum_j e^{s_j})] + \lambda \sum_i \sum_j W_{i,j}^2$$

Optimization Methods and Regularization

Define a regularization penalty $R(W)$

- Also used with weight update

$$R(W) = \sum_i \sum_j W_{i,j}^2$$

L2 regularization penalty: discourages large weights in W
Spreads "responsibility" for classification more evenly.
i.e. uses all dimensions rather than a few with large values
and thereby reduces overfitting and improves generalization

$$W = W - \alpha \nabla_{\mathbf{W}} f(W) \longrightarrow W = W - \alpha \nabla_{\mathbf{W}} f(W) + \lambda R(W)$$

Optimization Methods and Regularization

Define a regularization penalty $R(W)$

– Different regularization functions

$$R(W) = \sum_i \sum_j W_{i,j}^2$$

L2 regularization penalty
Also called weight decay

$$R(W) = \sum_i \sum_j |W_{i,j}|$$

L1 regularization penalty

$$R(W) = \sum_i \sum_j \beta W_{i,j}^2 + |W_{i,j}|$$

Elastic net

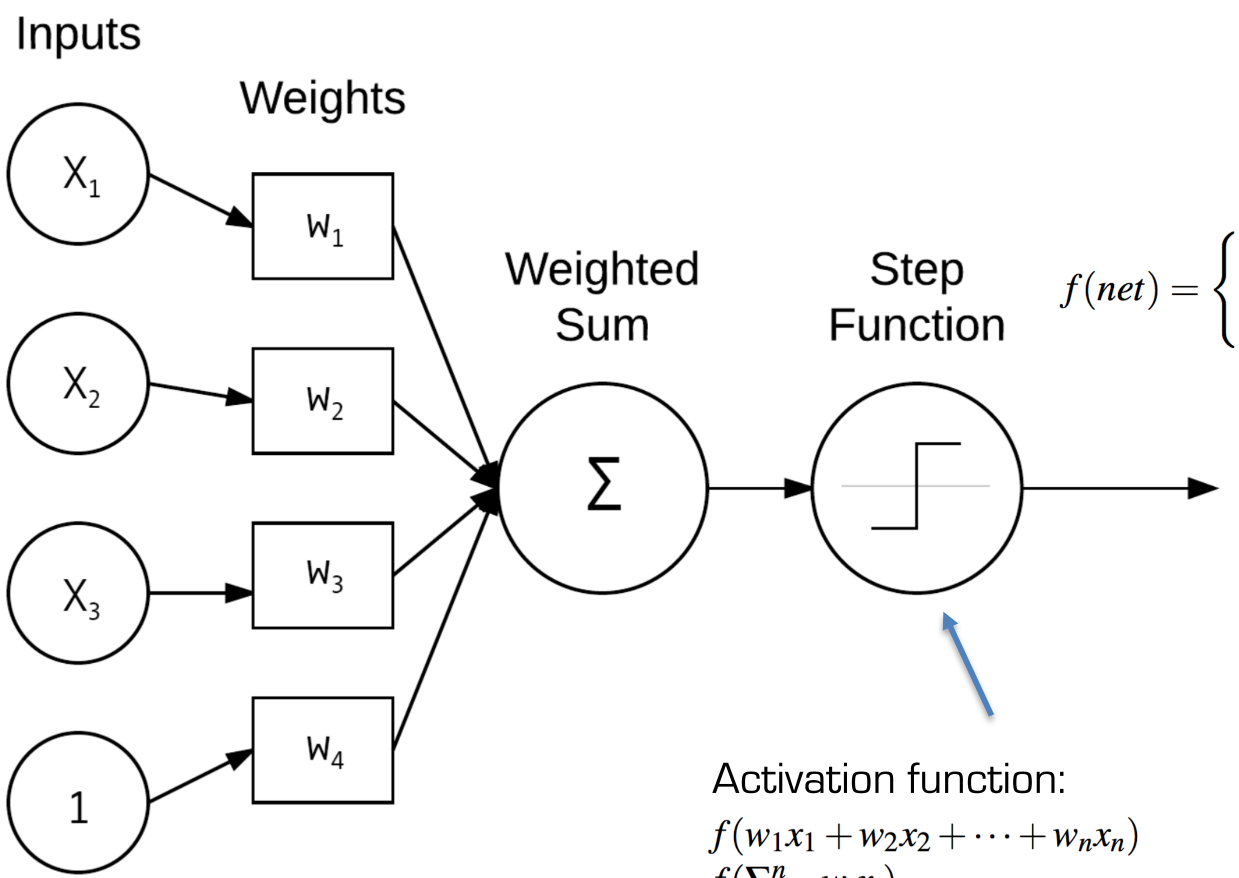
Optimization Methods and Regularization

Define a regularization penalty $R(W)$

- The trick is to tune the λ hyperparameter to uses just the right amount of regularization

- Other approaches later that modify the architecture, e.g. dropout

Fundamentals of Artificial Neural Networks



$$f(net) = \begin{cases} 1 & \text{if } net > 0 \\ 0 & \text{otherwise} \end{cases}$$

Activation function:

$f(w_1x_1 + w_2x_2 + \cdots + w_nx_n)$

$f(\sum_{i=1}^{n} w_ix_i)$

Or simply, $f(net)$, where $net = \sum_{i=1}^{n} w_ix_i$

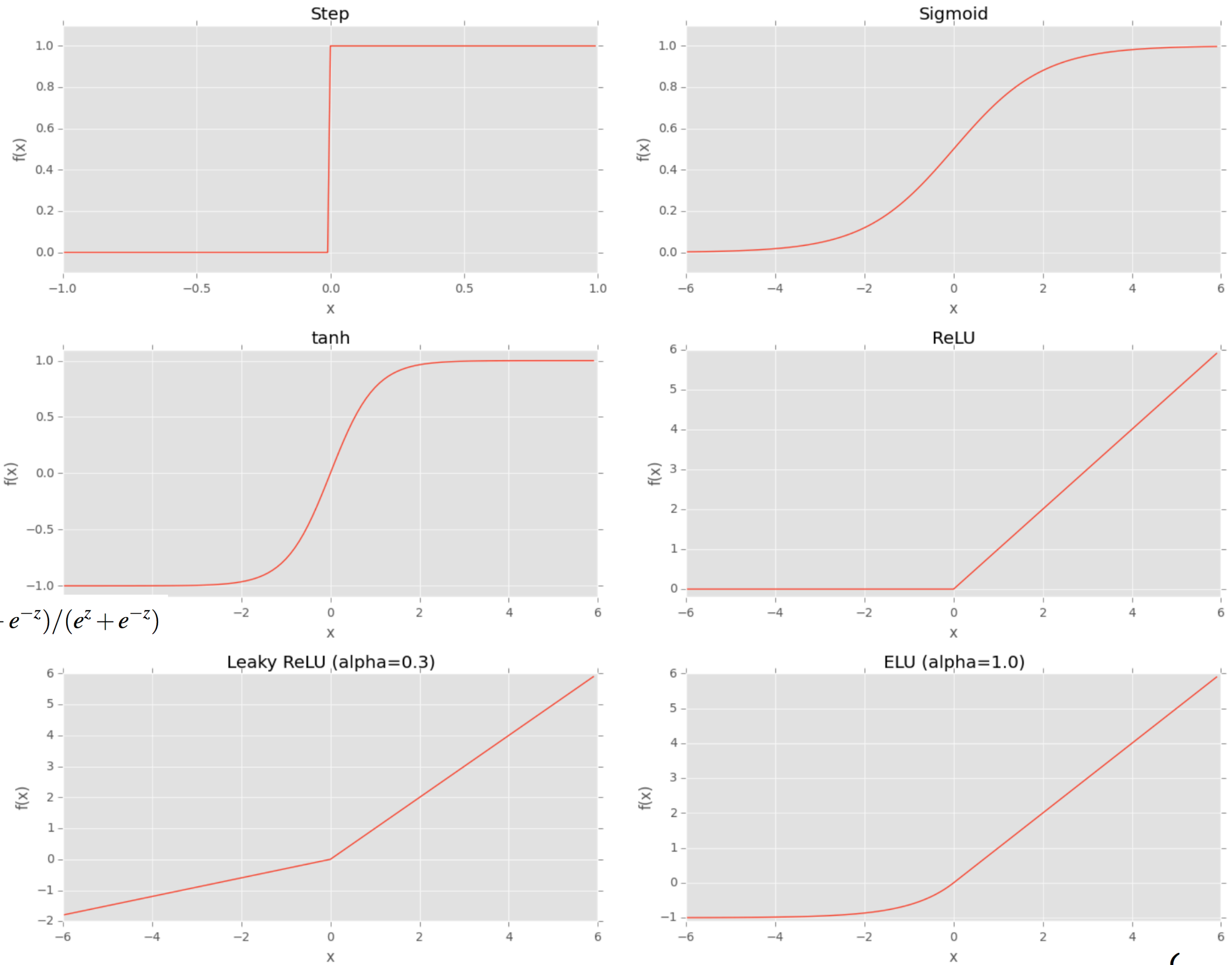
Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Fundamentals of Artificial Neural Networks

$$f(net) = \begin{cases} 1 & \text{if } net > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$t = \sum_{i=1}^{n} w_i x_i$$

$$s(t) = 1/(1+e^{-t})$$



Rectified Linear Unit (ReLU)

$$f(z) = tanh(z) = (e^{z} - e^{-z})/(e^{z} + e^{-z})$$

$$f(x) = max(0, x)$$

$$f(net) = \begin{cases} net & \text{if } net >= 0 \\ \alpha \times net & \text{otherwise} \end{cases}$$

$$f(net) = \begin{cases} net & \text{if } net >= 0 \\ \alpha \times (exp(net) - 1) & \text{otherwise} \end{cases}$$

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Fundamentals of Artificial Neural Networks

Classical activation functions

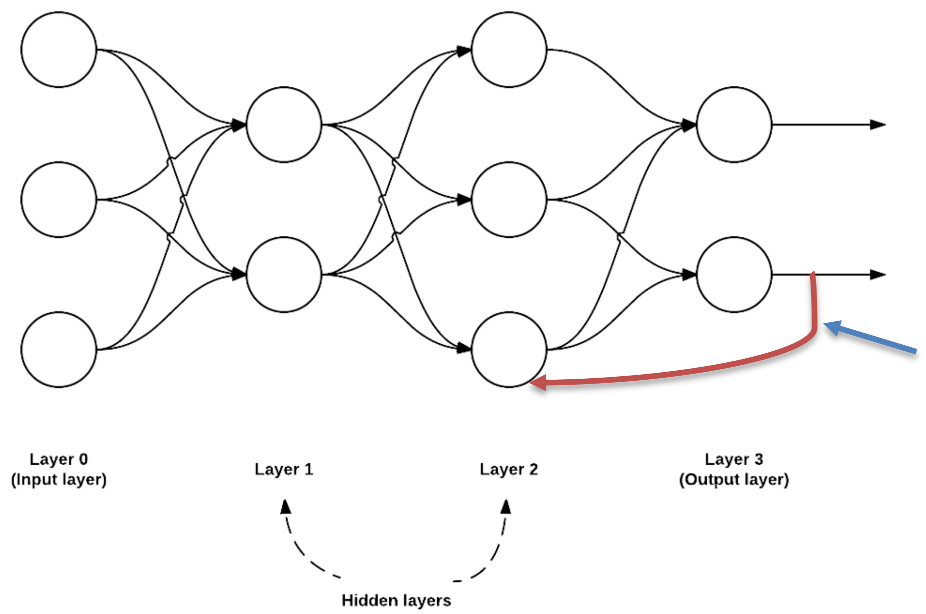
- step
- sigmoid
- tanh
- …

Modern activation functions

- ReLU ← Start with this, tune and optimize
- Leaky ReLU,
- ELU ← then substitute in this
- …

Fundamentals of Artificial Neural Networks

Feedforward Network Architectures



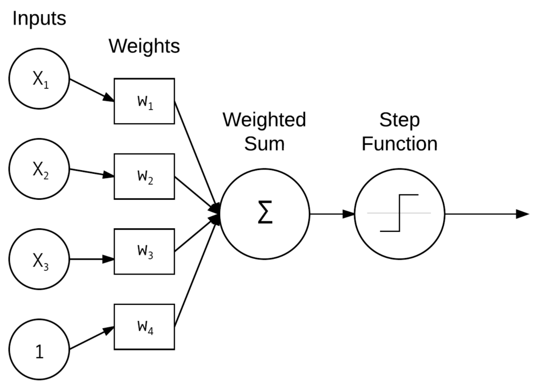
Recurrent neural networks have output connections that feed back into the inputs

Convolutional neural networks (CNNs) are a special case of feedforward neural networks

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Fundamentals of Artificial Neural Networks

Perceptron … simple one layer binary classifier (output is 0 or 1)



Each time the network has seen the full training set, we say an epoch has passed

To train a perceptron:

Iteratively feed the network with training data multiple times

1. Initialize our weight vector $\boldsymbol{w}$ with small random values
2. Until Perceptron converges:
   (a) Loop over each feature vector x_j and true class label d_i in our training set D
   (b) Take $\boldsymbol{x}$ and pass it through the network, calculating the output value: $y_j = f(\boldsymbol{w}(\boldsymbol{t}) \cdot \boldsymbol{x}_j)$
   (c) Update the weights $\boldsymbol{w}$: $w_i(t+1) = w_i(t) + \eta(d_j - y_j)x_{j,i}$ for all features $0 <= i <= n$

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Fundamentals of Artificial Neural Networks

Perceptron … simple one layer binary classifier (output is 0 or 1)

1. Initialize our weight vector $\boldsymbol{w}$ with small random values
2. Until Perceptron converges:
   (a) Loop over each feature vector x_j and true class label d_i in our training set D
   (b) Take $\boldsymbol{x}$ and pass it through the network, calculating the output value: $y_j = f(\boldsymbol{w(t)} \cdot \boldsymbol{x}_j)$
   (c) Update the weights $\boldsymbol{w}$: $w_i(t+1) = w_i(t) + \eta(d_j - y_j)x_{j,i}$ for all features $0 <= i <= n$

Learning rate:
Critical to set
the right rate:

Determines if the output classification
is correct or not

Terminate when
  all training samples are classified correctly or
  a preset number of epochs has been reached

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Fundamentals of Artificial Neural Networks

Backpropagation and Multi-layer Perceptron Networks

The backpropagation algorithm consists of two phases:

1. The forward pass
   - Inputs are passed through the network and output predictions obtained (the propagation phase)

This means the activation function must be differentiable

2. The backward pass
   - Compute the gradient of the loss function E at the final layer (i.e., predictions layer) of the network
   - Use this gradient to recursively apply the chain rule to update the weights in our network (the weight update phase).

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Fundamentals of Artificial Neural Networks

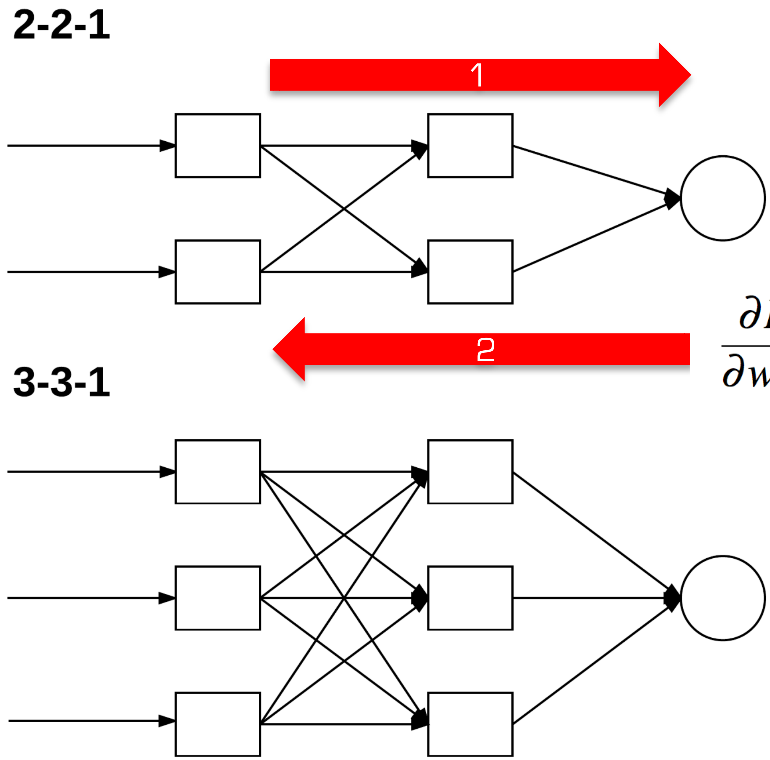
Backpropagation and Multi-layer Perceptron Networks

x_0	x_1	y
0	0	0
0	1	1
1	0	1
1	1	0

x_0	x_1	x_2
0	0	1
0	1	1
1	0	1
1	1	1

XOR function

XOR inputs
with bias column



XOR multilayer perceptron

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{i,j}}$$

XOR multilayer perceptron with bias trick

For a detailed description of how backpropagtion works, see http://neuralnetworksanddeeplearning.com/chap2.html
For a worked example, see https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Fundamentals of Artificial Neural Networks

MNIST

- 60,000 training images; 10,000 testing images
- 28 x 28 greyscale

Sigmoid activation function

Softmax activation function to generate class probabilities

- 784 – 256 – 128 – 10 feedforward neural network: 92% accuracy

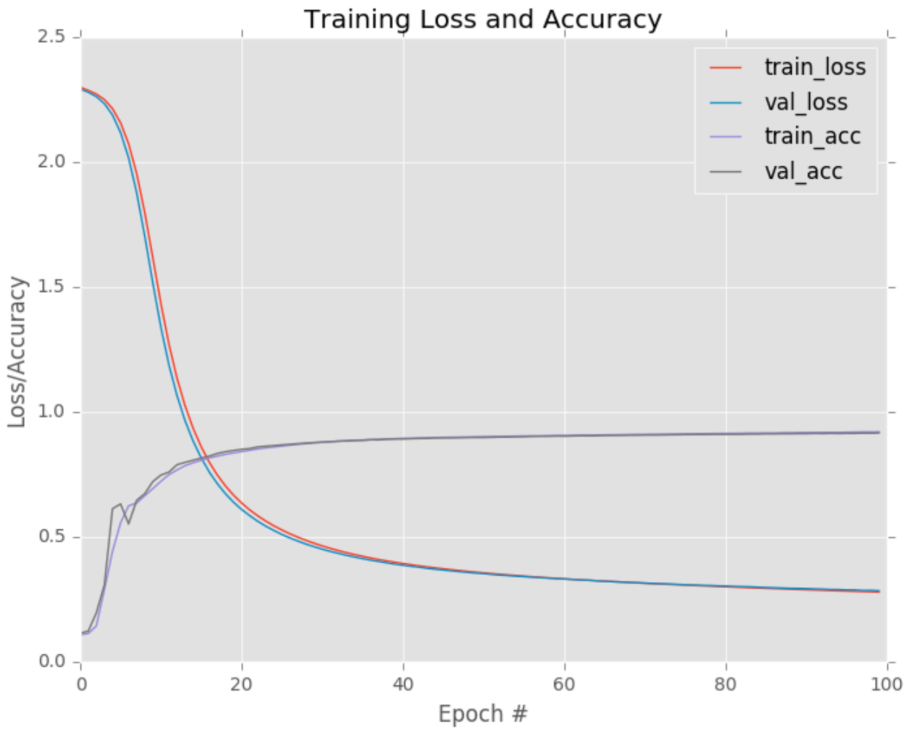
- CNN: > 98% accuracy

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Fundamentals of Artificial Neural Networks

MNIST

	precision	recall	f1-score	support
0.0	0.94	0.96	0.95	1726
1.0	0.95	0.97	0.96	2004
2.0	0.91	0.89	0.90	1747
3.0	0.91	0.88	0.89	1828
4.0	0.91	0.93	0.92	1686
5.0	0.89	0.86	0.88	1581
6.0	0.92	0.96	0.94	1700
7.0	0.92	0.94	0.93	1814
8.0	0.88	0.88	0.88	1679
9.0	0.90	0.88	0.89	1735
avg / total	0.92	0.92	0.92	17500



Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

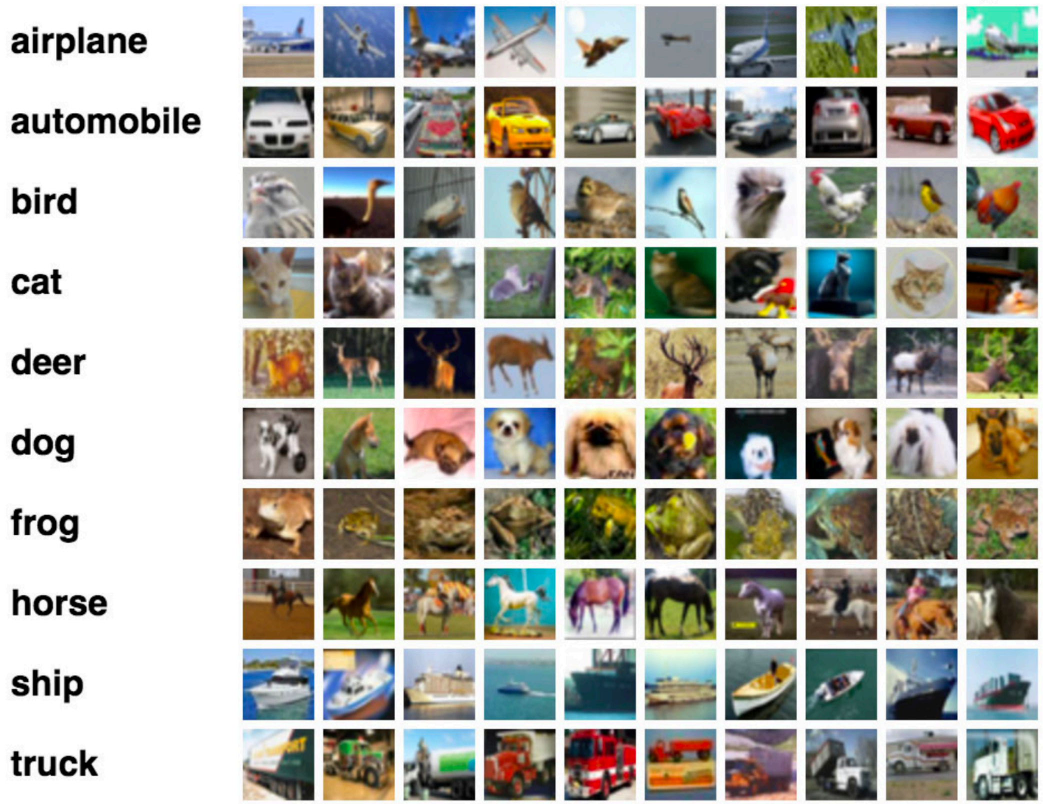
Fundamentals of Artificial Neural Networks

CIFAR-10

- 60,000 images
- 32 x 32 x 3 (RGB) ... 3072 element feature vector
- 10 classes

- 3072 – 1024 – 512 – 10
  57% accuracy

- Need a convolutional neural network (CNN) to get better accuracy (79% - 93%)



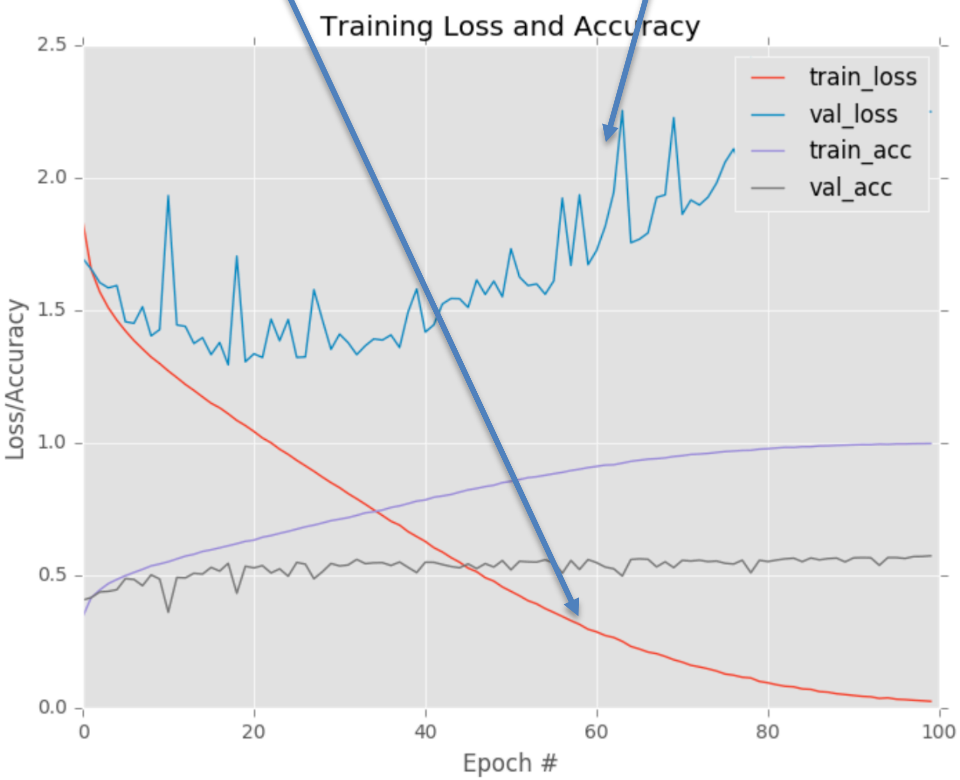
Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Fundamentals of Artificial Neural Networks

CIFAR-10

	precision	recall	f1-score	support
airplane	0.63	0.66	0.64	1000
automobile	0.69	0.65	0.67	1000
bird	0.48	0.43	0.45	1000
cat	0.40	0.38	0.39	1000
deer	0.52	0.51	0.51	1000
dog	0.48	0.47	0.48	1000
frog	0.64	0.63	0.64	1000
horse	0.63	0.62	0.63	1000
ship	0.64	0.74	0.69	1000
truck	0.59	0.65	0.62	1000
avg / total	0.57	0.57	0.57	10000

Dramatic over-fitting:
Training loss falls but validation loss rises



Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Fundamentals of Artificial Neural Networks

The four ingredients in designing a neural network

1. Dataset
   - At least 1000 images per class

2. Loss functions
   - For example, cross-entropy loss
     - Number of classes = 2: Binary cross-entropy loss
     - Number of classes > 2: Categorical cross-entropy loss

3. Model / Architecture
   - Number of data points, classes, similarity of classes, intra-class variance

4. Optimization method
   - For example, Stochastic Gradient Descent (SGD)
   - Set learning rate, regularization strength, number of epochs, momentum value, Nesterov acceleration, …

Fundamentals of Artificial Neural Networks

Weight initialization

- Uniform distribution of random values
  - Equal probability of every value in range
- Normal distribution of random values
  - Gaussian distribution of probability
- Alternatives
  - LeCun Uniform and Normal initialization
  - Glorot/Xavier Uniform and Normal initialization (default in the Keras library)
  - He et al. / Kaiming / MSRA Uniform and Normal initialization