Understanding Machine Learning & Neural Network

A Hardware Perspective

Chixiao Chen

Overview

- Linear Classifier
  - Score function
  - Loss function and softmax classifier
  - Gradient Descent

- Neural Network

Discussion on HW 2

- Problem 2

Assuming you are asked to design a specific processor to implementing matrix-vector multiplication based on the basic RV32 architecture. In other words, preform the following computing in a single cycle processor,

$$\underbrace{\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}}_{1\times 3} \cdot \underbrace{\begin{bmatrix} 2 & 1 & 3 \\ 3 & 3 & 2 \\ 4 & 1 & 2 \end{bmatrix}}_{3\times 3} = \begin{bmatrix} 1\cdot 2 + 2\cdot 3 + 3\cdot 4 \\ 1\cdot 1 + 2\cdot 3 + 3\cdot 1 \\ 1\cdot 3 + 2\cdot 2 + 3\cdot 2 \end{bmatrix} = \underbrace{\begin{bmatrix} 20 \\ 10 \\ 13 \end{bmatrix}}_{1\times 3}$$

Figure 2: A matrix-vector product example

Register `s0` is the base pointer to save the following sequence $\{1, 2, 3, 2, 3, 4, 1, 3, 1, 3, 2, 2\}$ in words (32bit). Register `t0` is the base pointer to store the output results. For the multiplication, please refer to the RV32M extension in appendix I. For simplicity, we only use the unsigned multiplication instruction `MUL` here.

(a) Write assembly codes to implement the matrix-vector product based on RV32I/RV32M instruction set. (hint: please use branch to minimize the code length.)

(b) Tailor a specific instruction set which only needs to support the above code. Use a table to illustrate all the instructions and their opcode/operand fields.

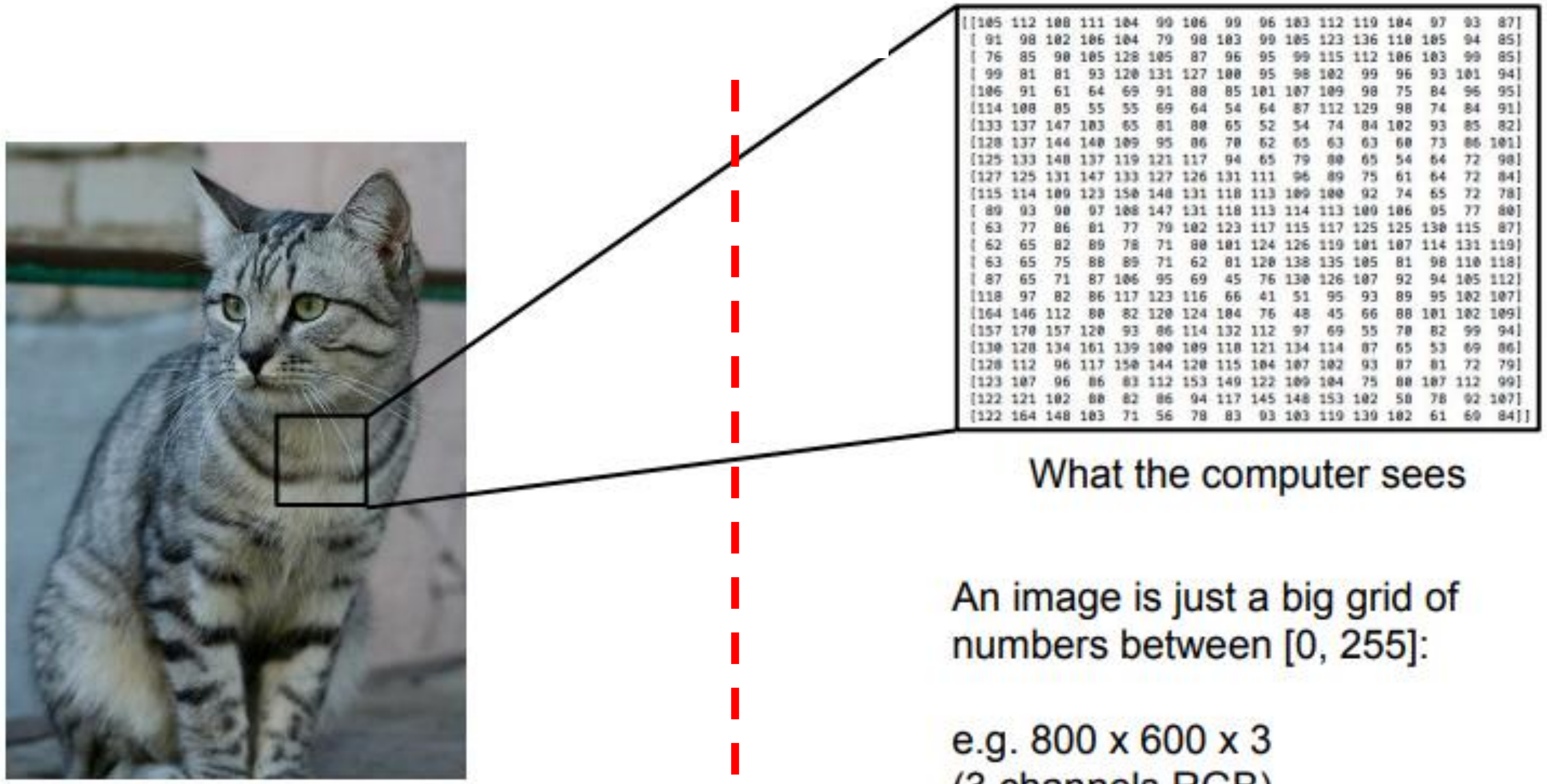
(c) Write an RTL code to realize all the instructions used in the above table. (hint: no need for pipeline here, and reuse the code in HW#1.)

Linear Classifier

Classification

- The most common and simple task in ML

- Problem: from human to data



(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

Why Hard ?

- Hard to get criteria

- Harder if occlusion

An Image Classifier

- Method 1: Define explicit feature (Unfortunately not working)

- Method 2: Machine Learning / Data-driven approach
  - Collect data set and train a model

```python
def classify_image(image):
  # Some magic here?
  return class_label
```

```python
def train(images, labels):
  # Machine learning!
  return model
```

Memorize all data and labels

```python
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

Predict the label of the most similar training image

Common Data set

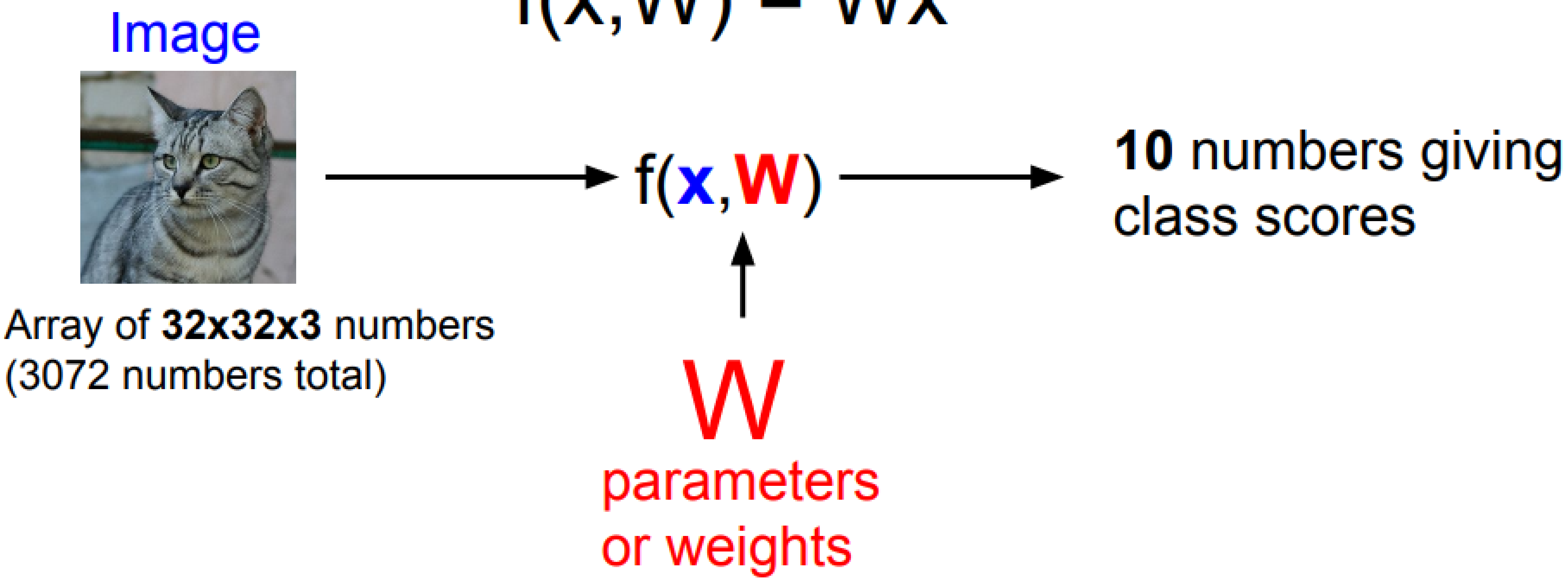
- MNIST
  - Handwrite digits
  - 10 classes

- CIFAR 10 (100)
  - Small but real pictures

Linear Classifier

- Use CIFAR 10 as examples
- What’s the size of x , w and f ?

$$f(x,W) = Wx$$



Score Function

- In practice, we may need bias
- f() is also called score function

$$\underbrace{\boxed{f(x,W)}}_{10\text{x}1} = \underbrace{\boxed{W}}_{10\text{x}3072}\overbrace{\boxed{x}}^{3072\text{x}1} + \underbrace{\boxed{b}}_{10\text{x}1}$$

Image

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

$$\text{Image} \longrightarrow f(\mathbf{x},\mathbf{W}) \longrightarrow \text{10 numbers giving class scores}$$

$$\uparrow$$

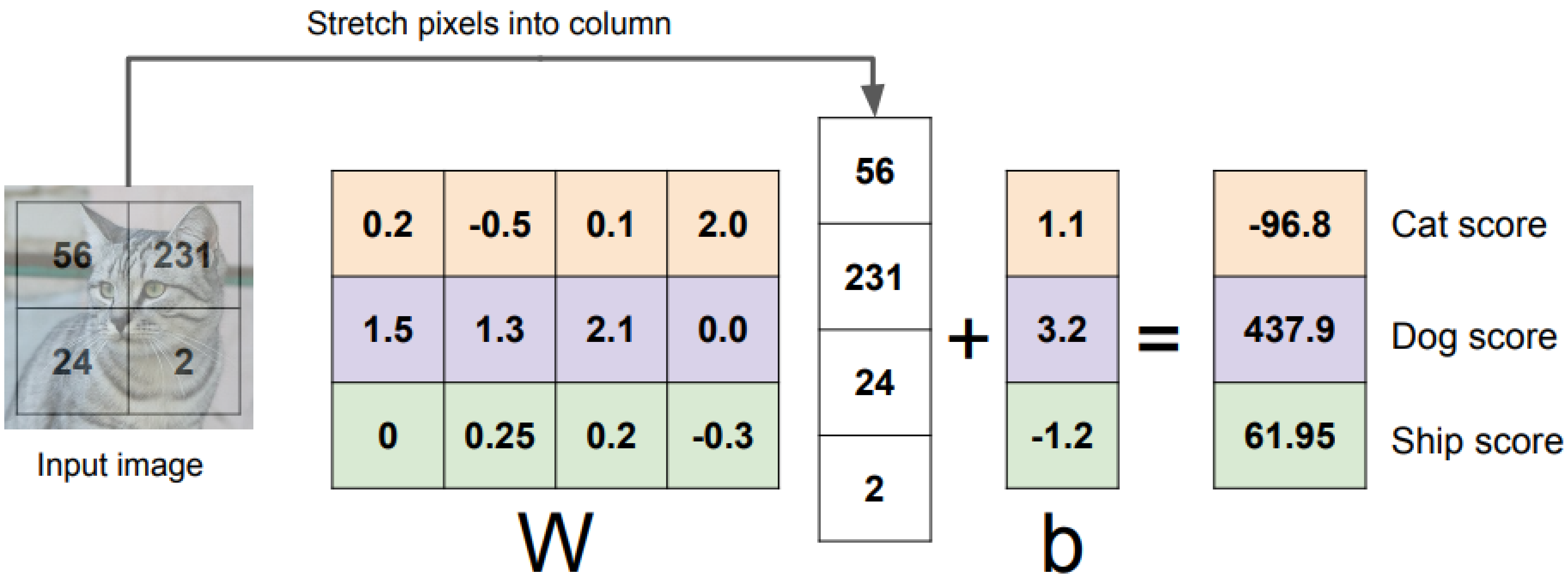
W

parameters
or weights

10 numbers giving
class scores

A simple example

- Assuming we have an image with 4 pixels, and 3 classes (cat/dog/ship)
- Pick up the maximum.



Loss function

- A loss function tells how good a classifier is

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and
y_i is (integer) label

Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



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

Softmax Classifier

- Other loss function, such as multiclass SVM, is not included in the class.
- Motivation: interpret raw classifier scores as probabilities

- Also known as logistic regression classifier

- Softmax uses a cross-entropy loss

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

Softmax Classifier Example - I

Want to interpret raw classifier scores as **probabilities**

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

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ Softmax Function

Probabilities must be >= 0

cat	**3.2**		**24.5**
car	5.1	$\xrightarrow{\text{exp}}$	164.0
frog	-1.7		0.18

unnormalized probabilities

Softmax Classifier Example - II

Want to interpret raw classifier scores as **probabilities**

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

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ Softmax Function

		exp →	Probabilities must be >= 0	normalize →	Probabilities must sum to 1
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00
			unnormalized probabilities		probabilities

Softmax Classifier Example - III

Want to interpret raw classifier scores as **probabilities**

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

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

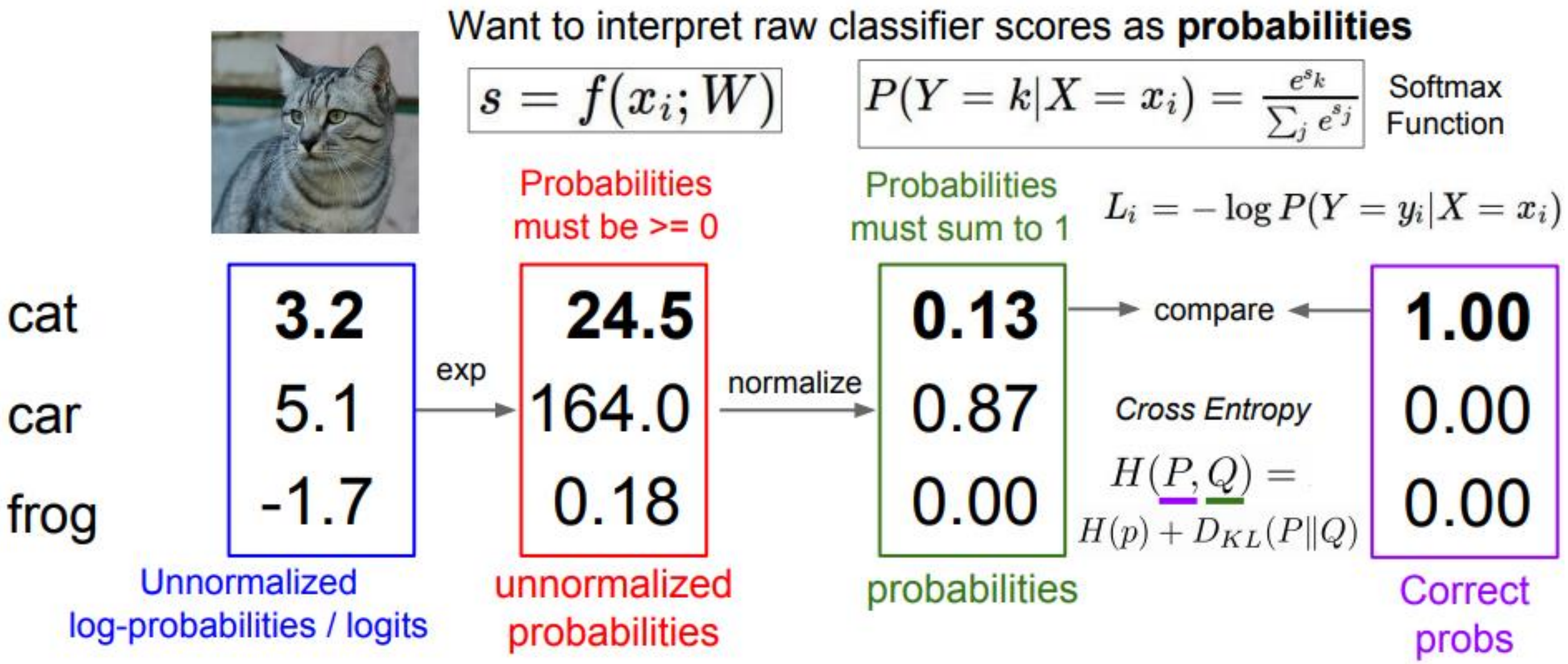
$$L_i = -\log P(Y = y_i | X = x_i)$$

	Unnormalized log-probabilities / logits	$\xrightarrow{\text{exp}}$	unnormalized probabilities	$\xrightarrow{\text{normalize}}$	probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

$\rightarrow L_i = -\log(0.13)$
$= \mathbf{2.04}$

Maximum Likelihood Estimation
Choose probabilities to maximize the likelihood of the observed data

Softmax Classifier Example - III



A computing trick

- Exponential computing might generate very big numbers, exceeding the integer range
- Scaling is before exponential domain

$$\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = \frac{Ce^{f_{y_i}}}{C\sum_j e^{f_j}} = \frac{e^{f_{y_i}+\log C}}{\sum_j e^{f_j+\log C}}$$

- Sample code: also work in HW!

```python
f = np.array([123, 456, 789]) # example with 3 classes and each having large scores
p = np.exp(f) / np.sum(np.exp(f)) # Bad: Numeric problem, potential blowup

# instead: first shift the values of f so that the highest number is 0:
f -= np.max(f) # f becomes [-666, -333, 0]
p = np.exp(f) / np.sum(np.exp(f)) # safe to do, gives the correct answer
```

How to find the best weight ?

- Recap: dataset, score function, loss function

- How to find the best score function / weights?
- Normally, random initiate provide 10%~15% accuracy for 10 class, like guess

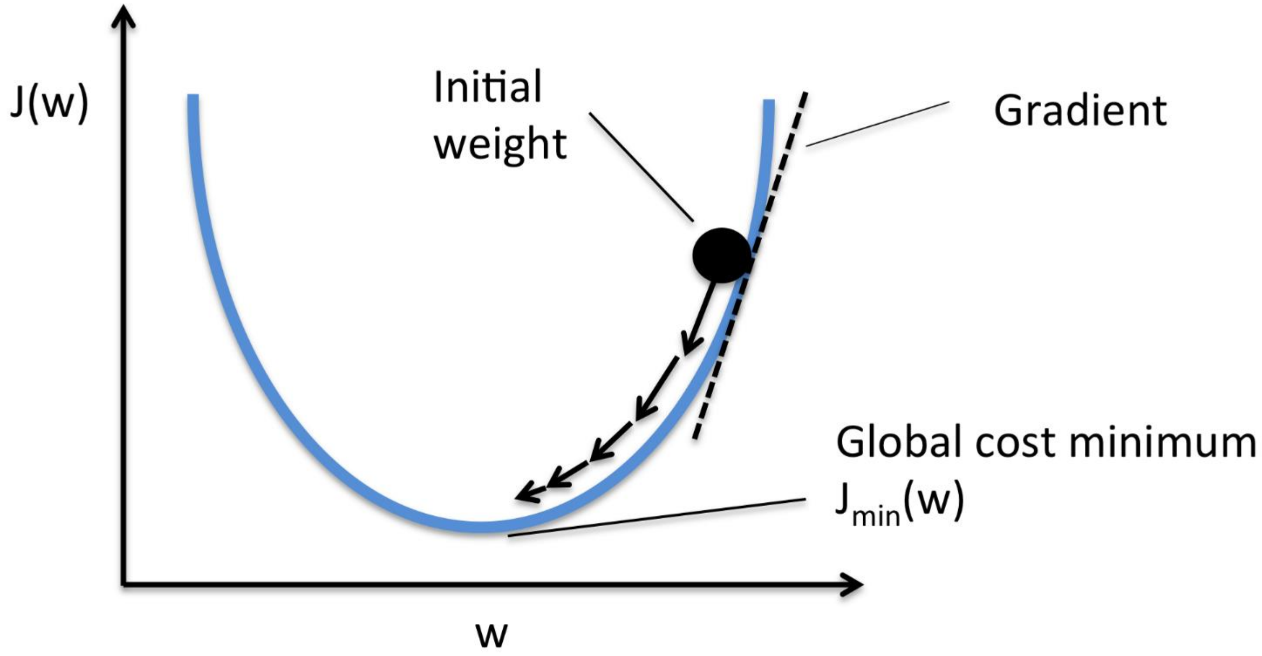
- Find the min. (__________) ?

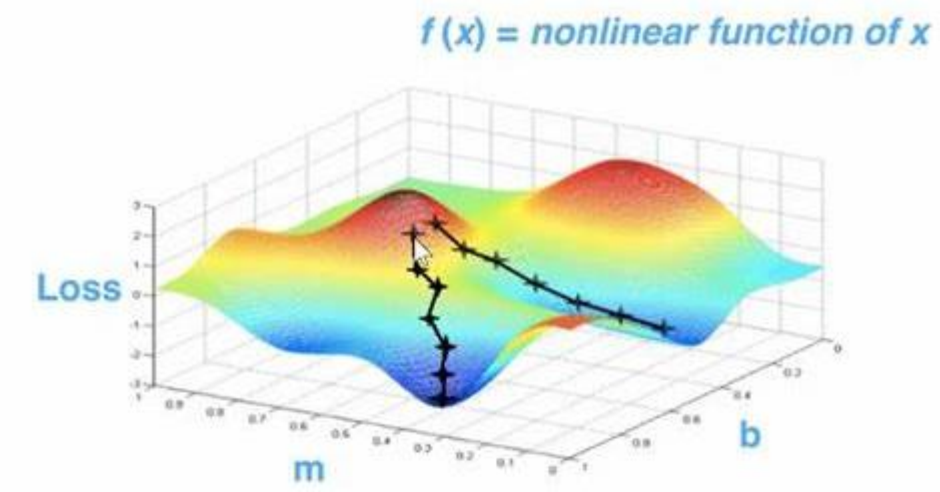
Gradient Descent

- Derivative / Gradient

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- The slope in any direction is the dot product of the direction with the gradient The direction of steepest descent is the negative gradient





Numerical Gradient Computing

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (first dim)**:**

[0.34 + **0.0001**,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25322

gradient dW:

[**-2.5**,
?,
?,

(1.25322 - 1.25347)/0.0001
= -2.5

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

?,
?,…]

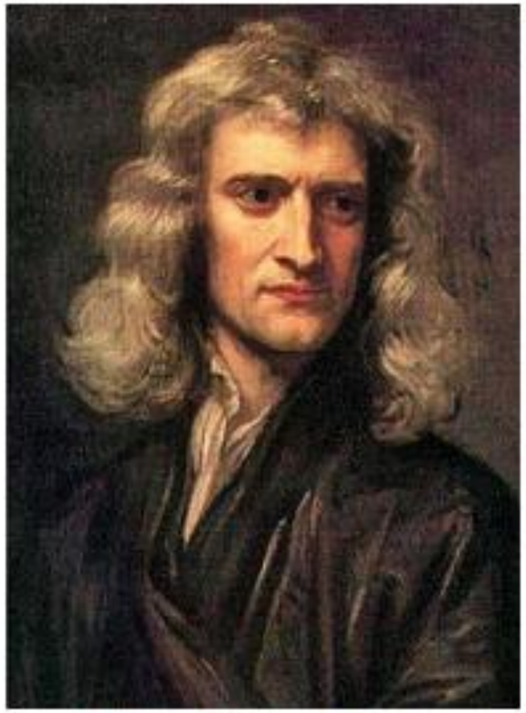
Analytical Gradient Computing

- We all learned calculus.

- Linear derivative is super easy:

$$f(a)=ax+b, f'(a)=x$$

- What about softmax?



This image is in the public domain

This image is in the public domain

Derivative of softmax classifier

$$P_k = \frac{e^{f_k}}{\sum_j e^{f_j}} \qquad L_i = -\sum_k p_{i,k} \log P_k \qquad f_m = (x_i W)_m$$

when $k = m$, $$\frac{\partial P_k}{\partial f_m} = \frac{e^{f_k} \sum_j e^{f_j} - e^{f_k} \cdot e^{f_k}}{(\sum_j e^{f_j})^2} = P_k(1 - P_k)$$

when $k \neq m$, $$\frac{\partial P_k}{\partial f_m} = -\frac{e^{f_k} e^{f_m}}{(\sum_j e^{f_j})^2} = -P_k P_m$$

then:

$$\frac{\partial L_i}{\partial f_m} = -\sum_k p_{i,k} \frac{\partial \log P_k}{\partial f_m}$$

$$= -\sum_k p_{i,k} \frac{1}{P_k} \frac{\partial P_k}{\partial f_m}$$

$$= -\sum_{k=m} p_{i,k} \frac{1}{P_k} P_k(1 - P_k) + \sum_{k \neq m} p_{i,k} \frac{1}{P_k} P_k P_m$$

$$= \sum_{k \neq m} p_{i,k} P_m - \sum_{k=m} p_{i,k}(1 - P_k)$$

$$= \begin{cases} P_m & , \quad m \neq y_i \\ P_m - 1 & , \quad m = y_i \end{cases}$$

$$\boxed{= P_m - p_{i,m}}$$

Last:

$$\frac{\partial L_i}{\partial W_k} = \frac{\partial L_i}{\partial f_m} \frac{\partial f_m}{\partial W_k} = x_i^{\ T}(P_m - p_{i,m})$$

$$\nabla_{W_k} L = -\frac{1}{N} \sum_i x_i^{\ T}(p_{i,m} - P_m) + 2\lambda W_k$$

Numerical vs. Gradient

- Numerical gradient: approximate, slow, easy to write

- Analytic gradient: exact, fast, error-prone

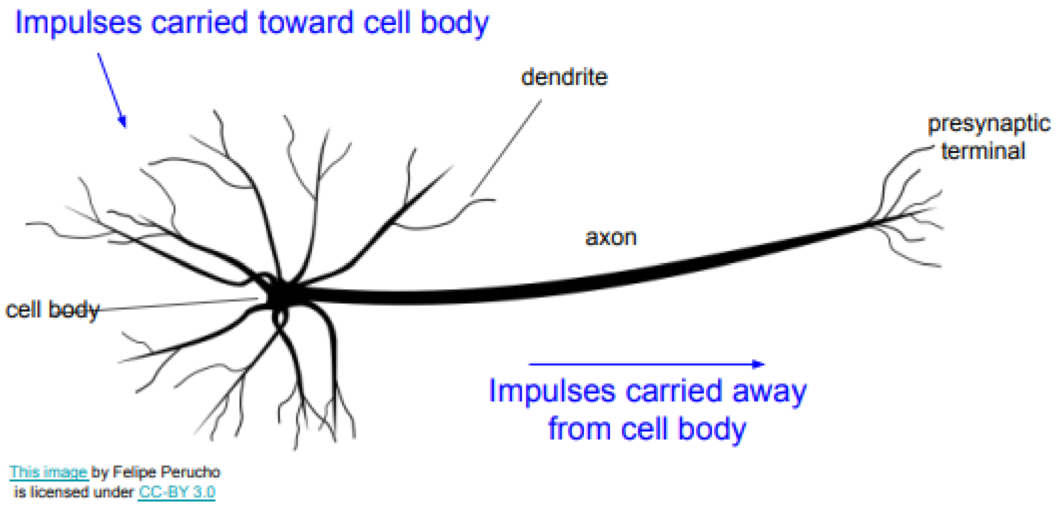
---- When you only have CPUs

---- Numerical gradient is NOT slow when you have accelerator!

Neuron Network

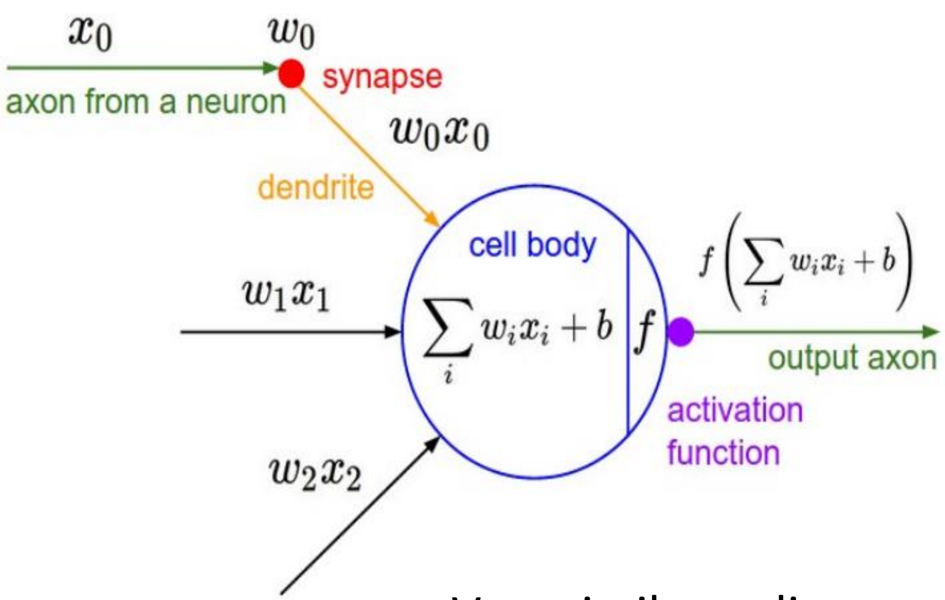
Neuron

- Neuron Structure



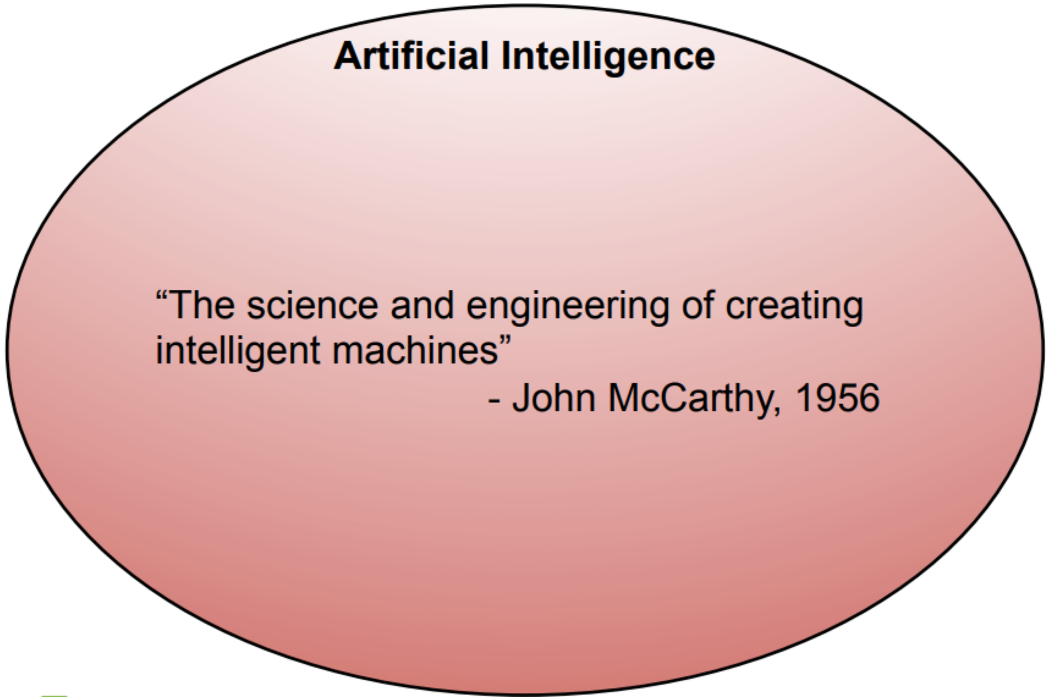
- It has many layers.

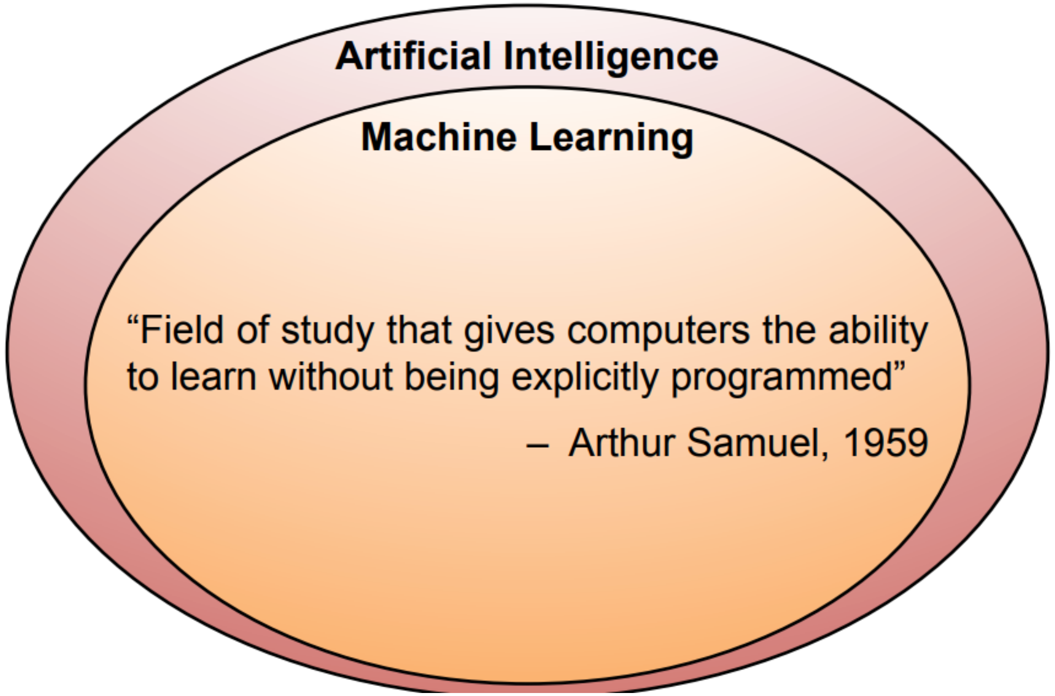
- Its math model



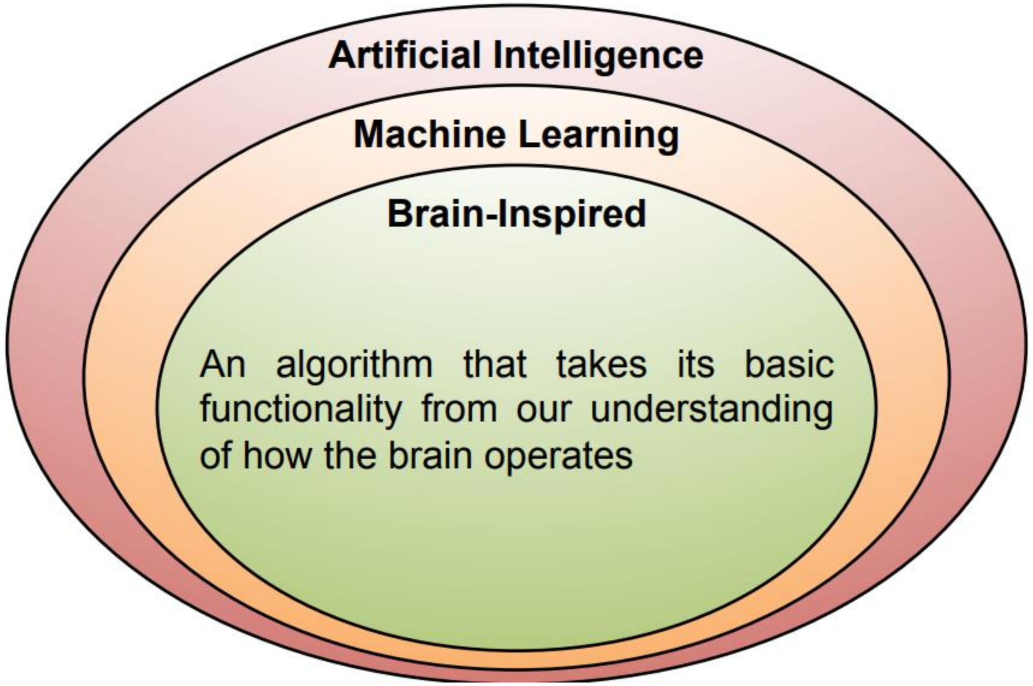
Very similar to linear classifier, **except** …

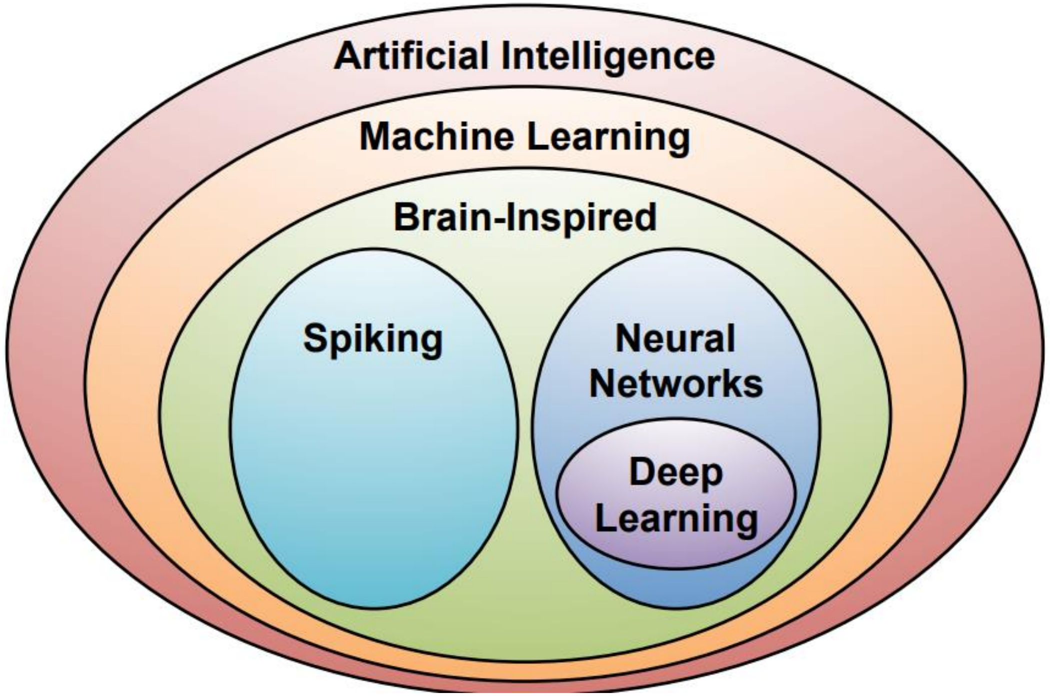
Artificial Intelligence and Machine Learning





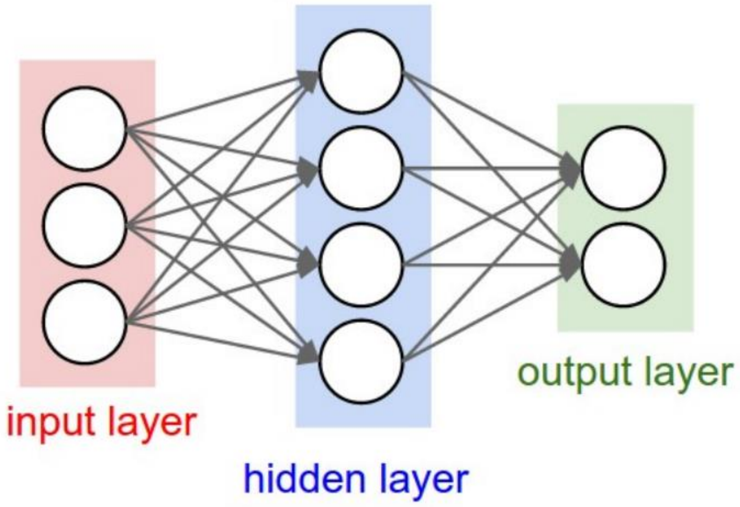
Brain Inspired and Neural Network

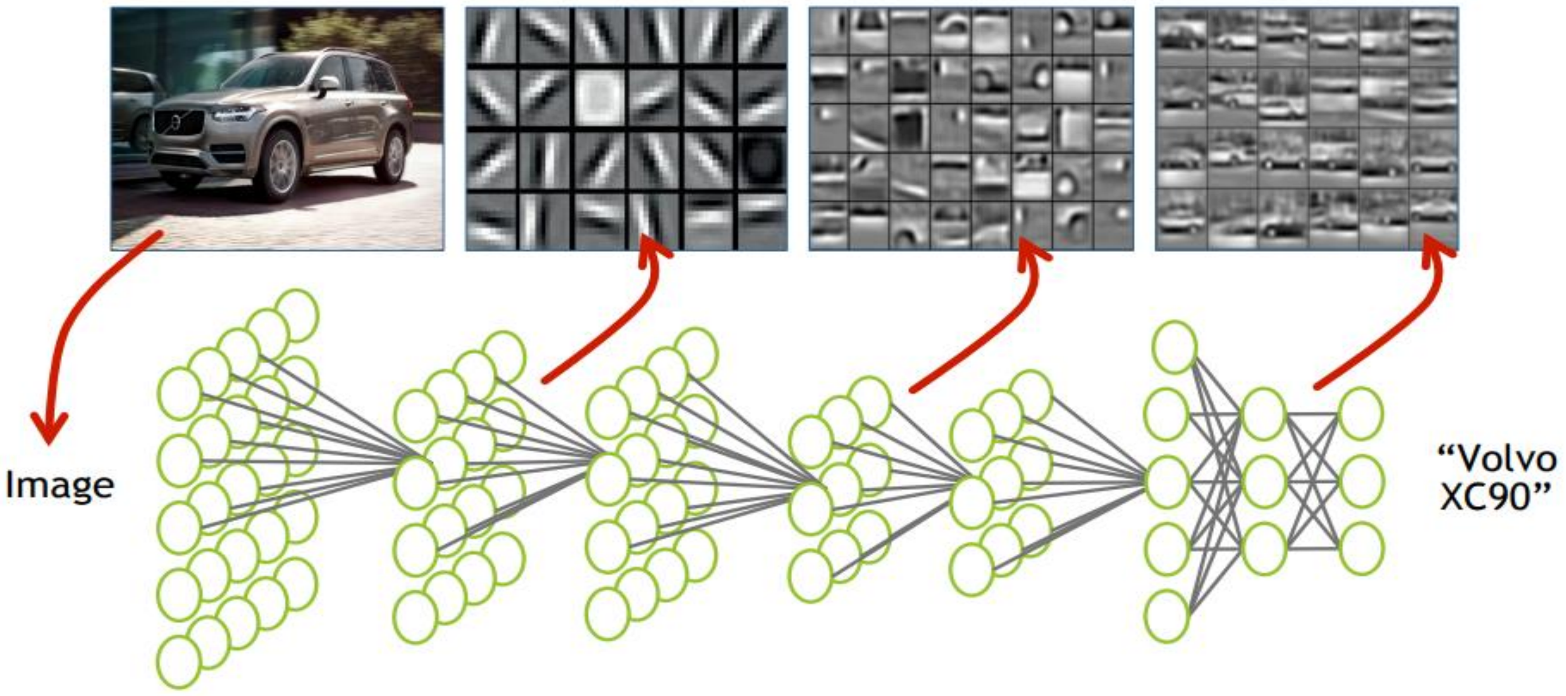




MLP and Deep Learning

- Previously, we have a name called: MLP: multi-level perceptron
- Now: Deep Learning





Back-propagation

- How can we find the derivative for multi-layer structure?

Backpropagation: a simple example

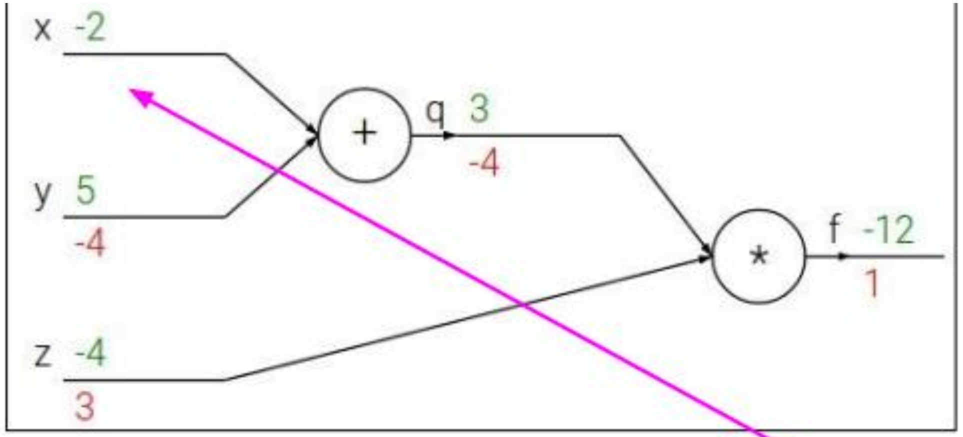
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



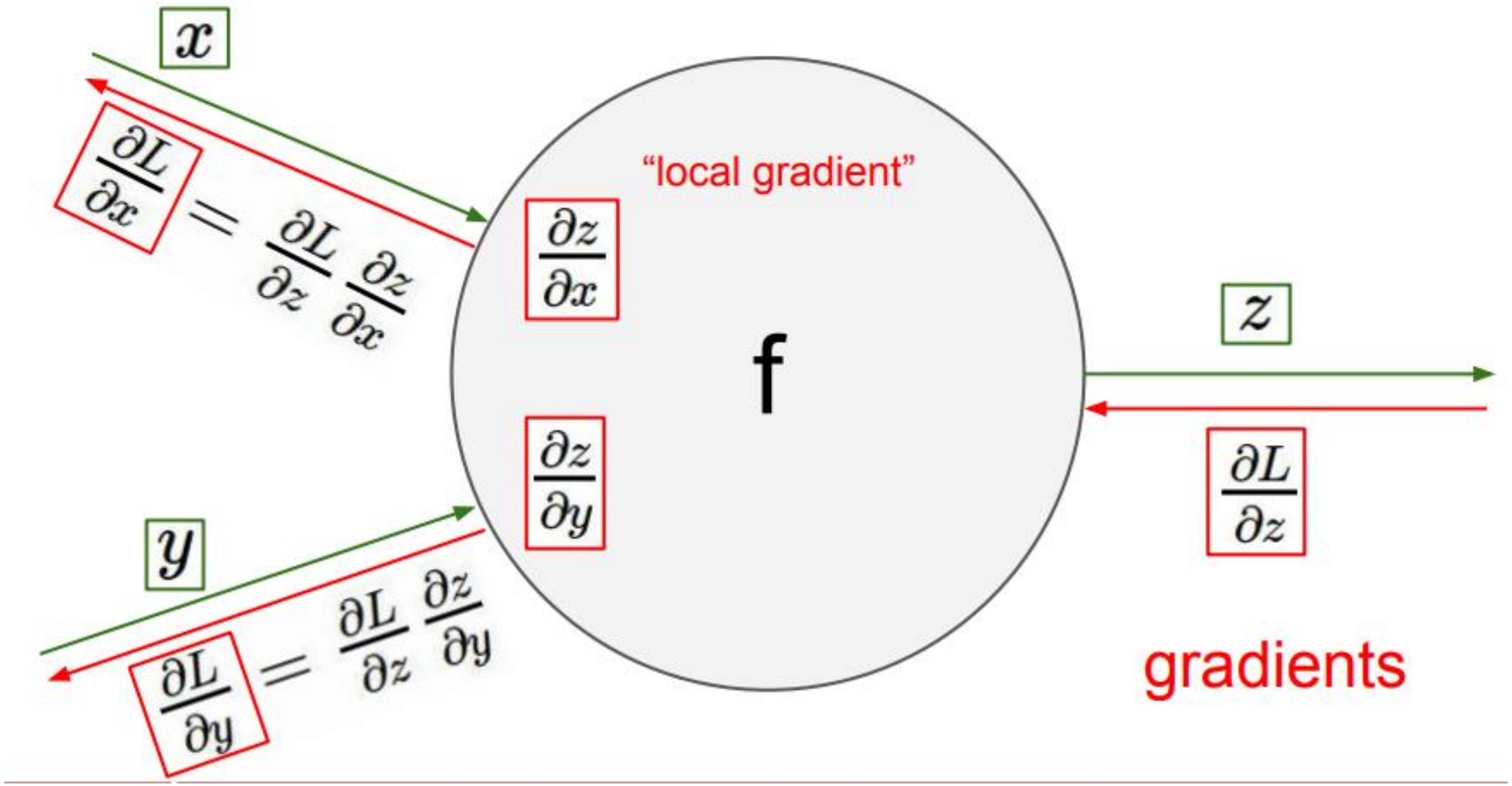
$$\boxed{\frac{\partial f}{\partial x}}$$

Chain rule:

$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}}$$

Upstream gradient   Local gradient

Chain Rule



x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

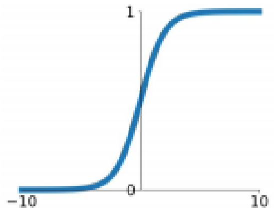
Activation Functions

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x)$$

Sigmoid is easy for derivative

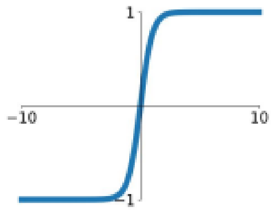
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



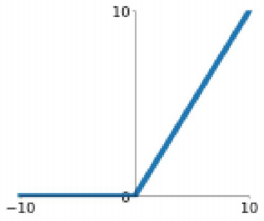
tanh

$$\tanh(x)$$



ReLU

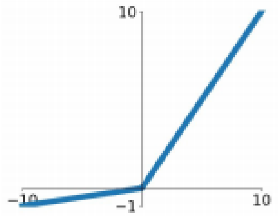
$$\max(0, x)$$



ReLU is the most common one
Used in recent CNN

Leaky ReLU

$$\max(0.1x, x)$$

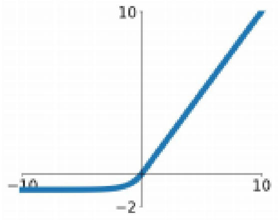


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

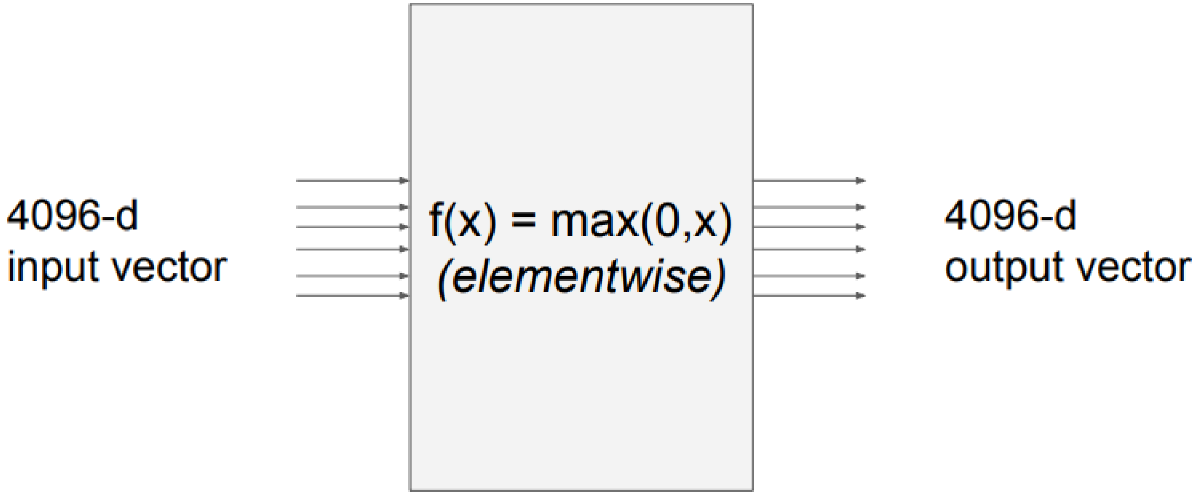
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



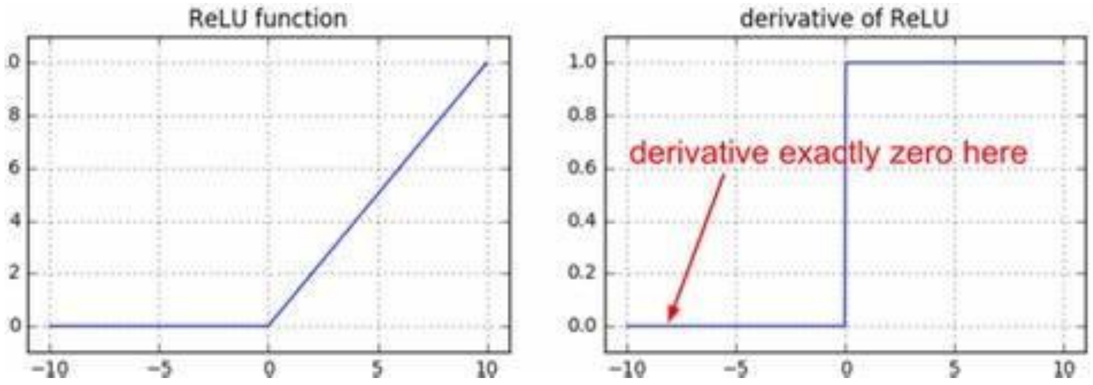
Vectorized Gradient Computing

- Jacobian Matrix

$$\mathbf{J} = \left[\frac{\partial \mathbf{f}}{\partial x_1} \quad \cdots \quad \frac{\partial \mathbf{f}}{\partial x_n} \right] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$



- How big will be the Jacobian matrix?
- Mini-batch make it worse



Using Numpy to train a 2-layer model

- For simplicity: use sigmoid as activation

```
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

Inference vs. Training

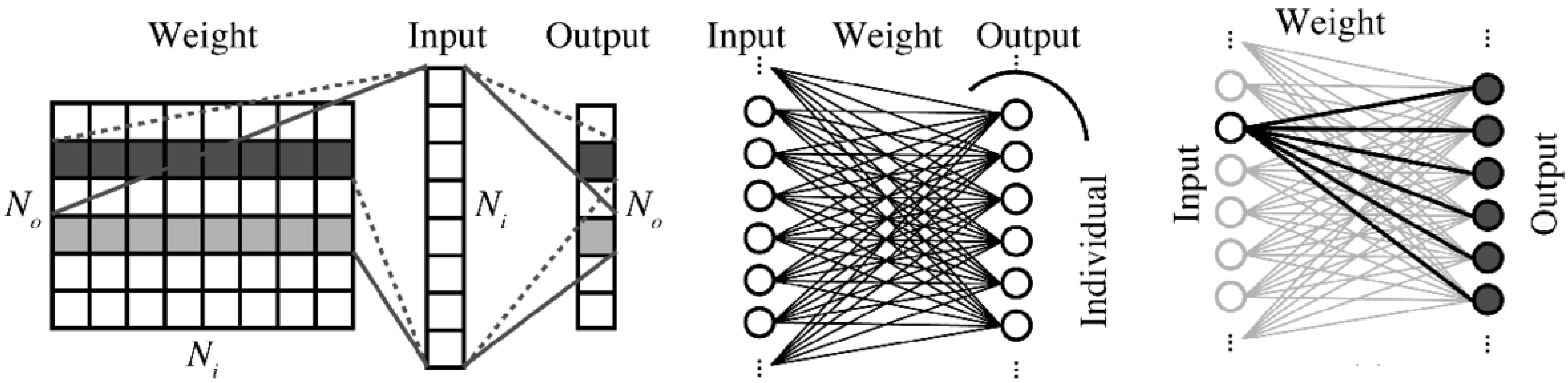
- Inference: Apply weights to determine output

- Training: Determine weights

  - Supervised: Training set has inputs and outputs, i.e., labeled
  - Unsupervised: Training set is unlabeled
  - Semi-supervised: Training set is partially labeled
  - Reinforcement: Output assessed via rewards and punishments

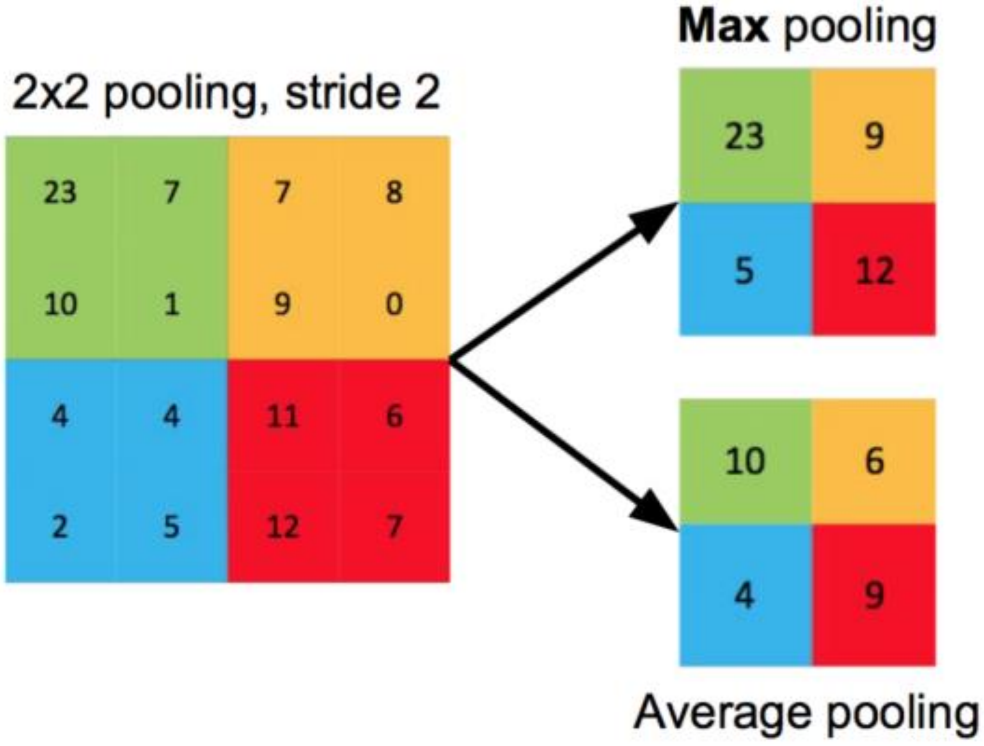
Fully Connected Layer

- Just as the linear classifier
- Even today, the last stage of DNNs are still FC layers



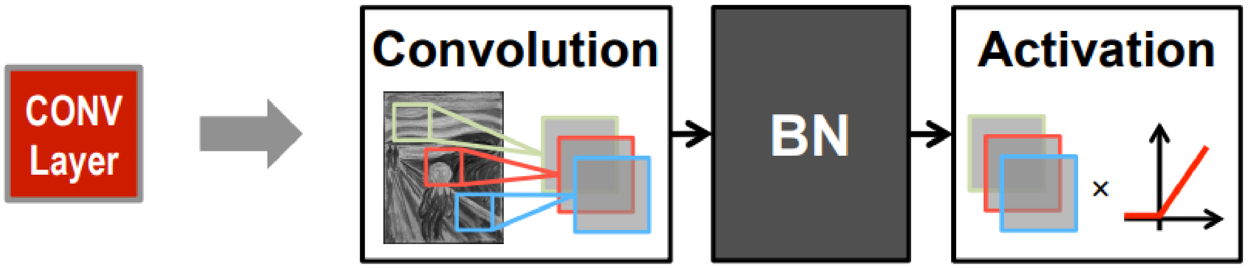
Polling Layer

- Reduce resolution of each channel independently
- Overlapping or non-overlapping → depending on stride
- Increases translation-invariance and noise-resilience



Normalization Layer

- Batch Normalization (BN)
  - Normalize activations towards mean=0 and std. dev.=1 based on the statistics of the training dataset
  - Believed to be key to getting high accuracy and faster training on very deep neural networks.
  - Makes training harder, but almost do not affect inference.



data mean

learned scale factor

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \beta$$

learned shift factor

data std. dev.

small const. to avoid numerical problems

Summary

Next time: CNN