Dive into Deep Learning in 1 Day

1 Basics · 2 Convnets · 3 Computation · 4 Sequences

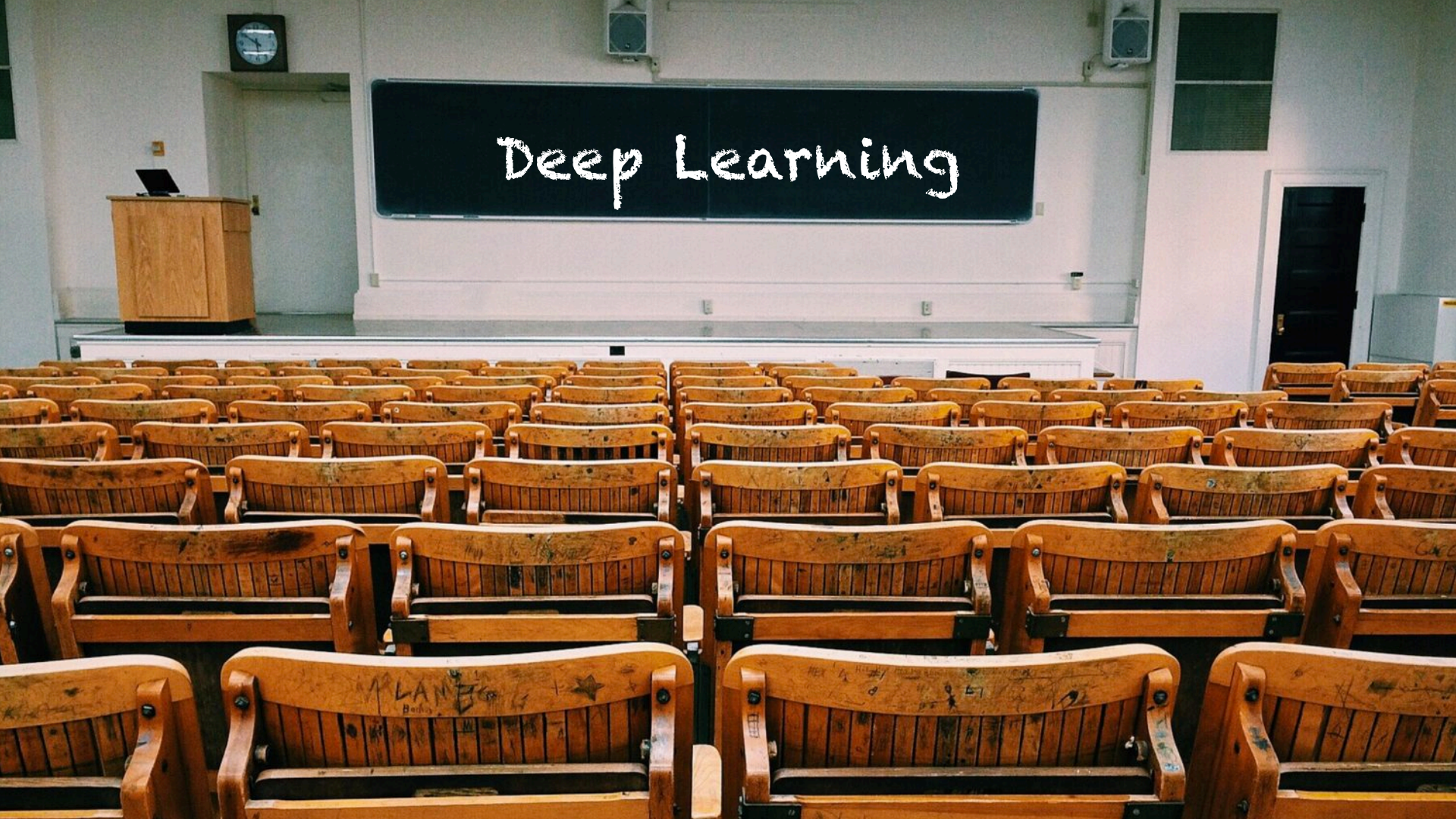
AMLC 2019

Mu Li and Alex Smola
amlc-d2l.corp.amazon.com

aws

Outline

- **Introduction to Deep Learning**
- **Installation** (Cloud, Laptop, GitHub)
- **Linear Algebra and Deep Numpy**
- **Autograd**
- **Linear Regression**
- **Optimization**
- **Softmax Classification**
- **Multilayer Perceptron (train MNIST)**

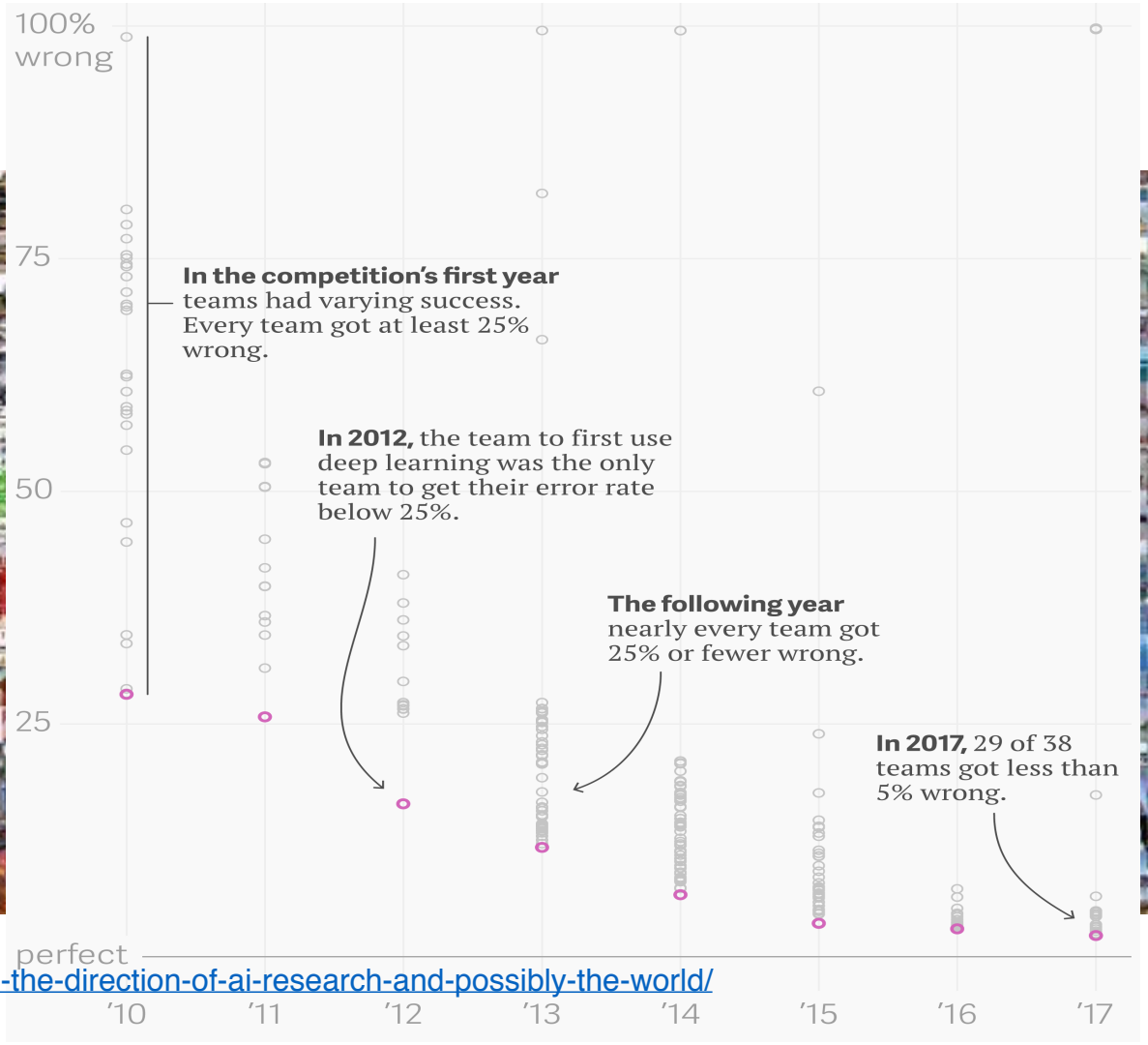

Deep Learning

Classify Images

http://www.image-net.org/

aws

Classify Images

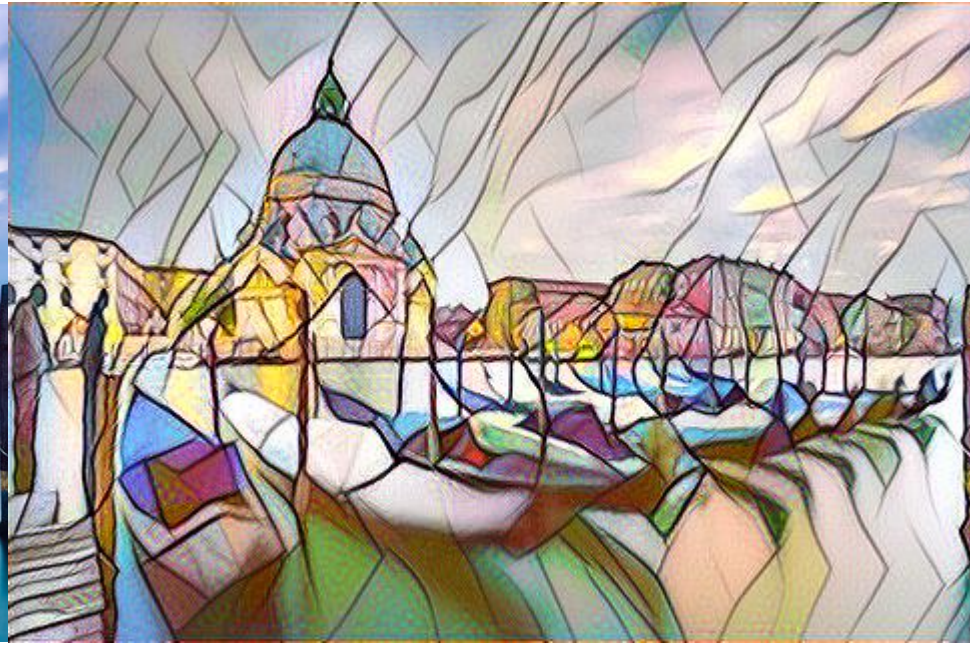


Yanofsky, Quartz
https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world/

Detect and Segment Objects

https://github.com/matterport/Mask_RCNN

Style transfer

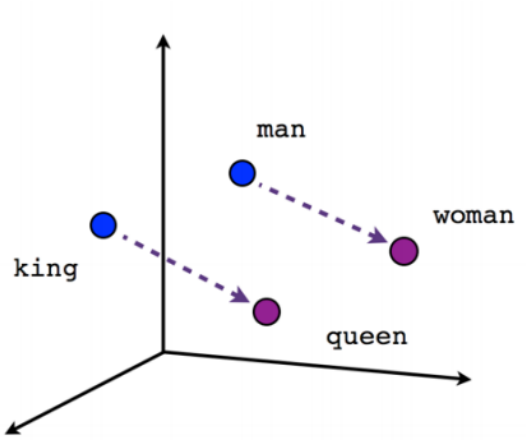


https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/

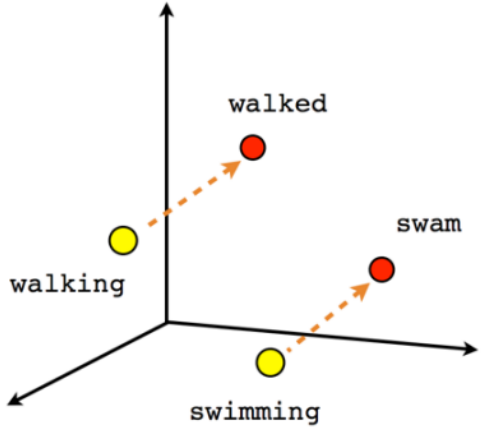
Synthesize Faces

Karras et al, arXiv 2019

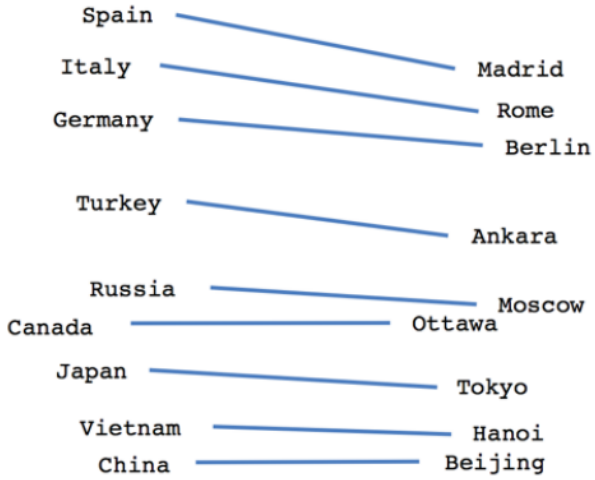
Analogies



Male-Female



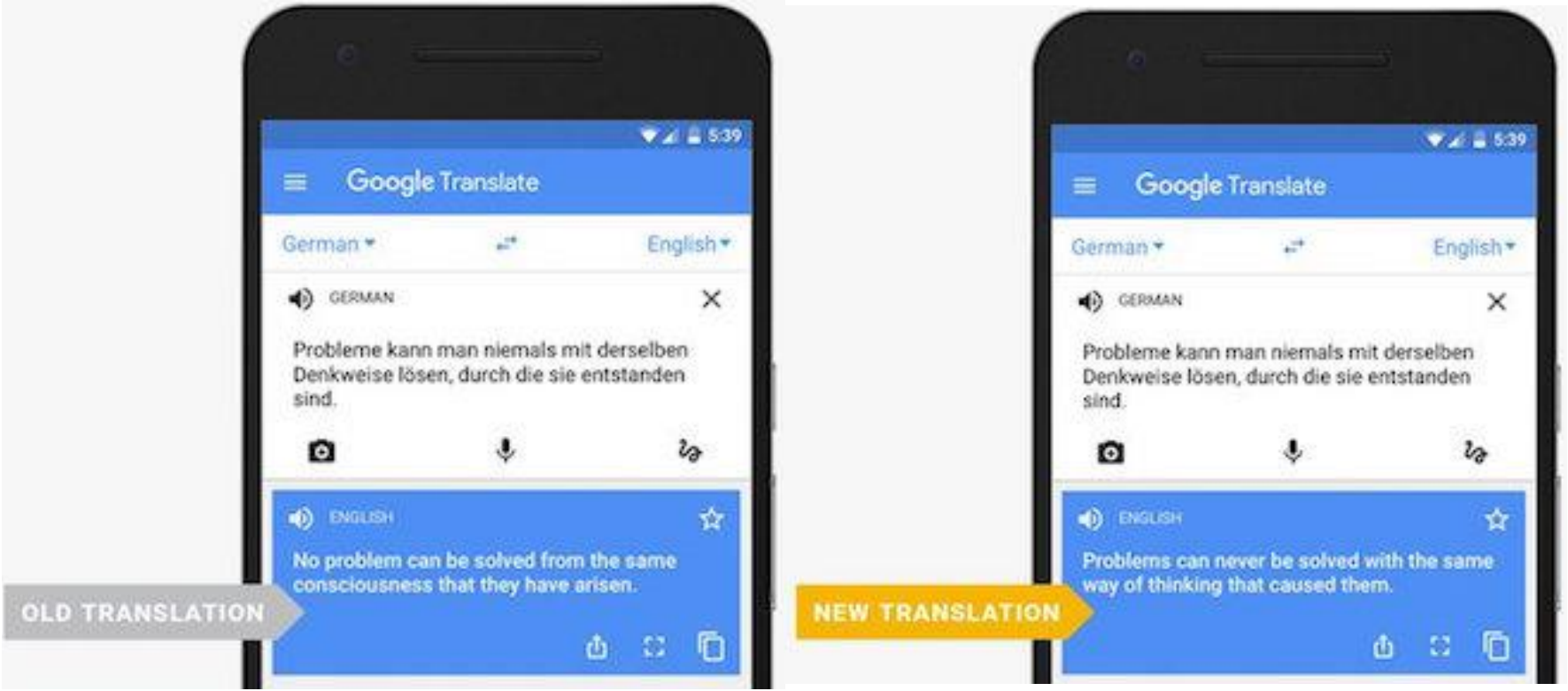
Verb tense



Country-Capital

https://www.tensorflow.org/tutorials/word2vec

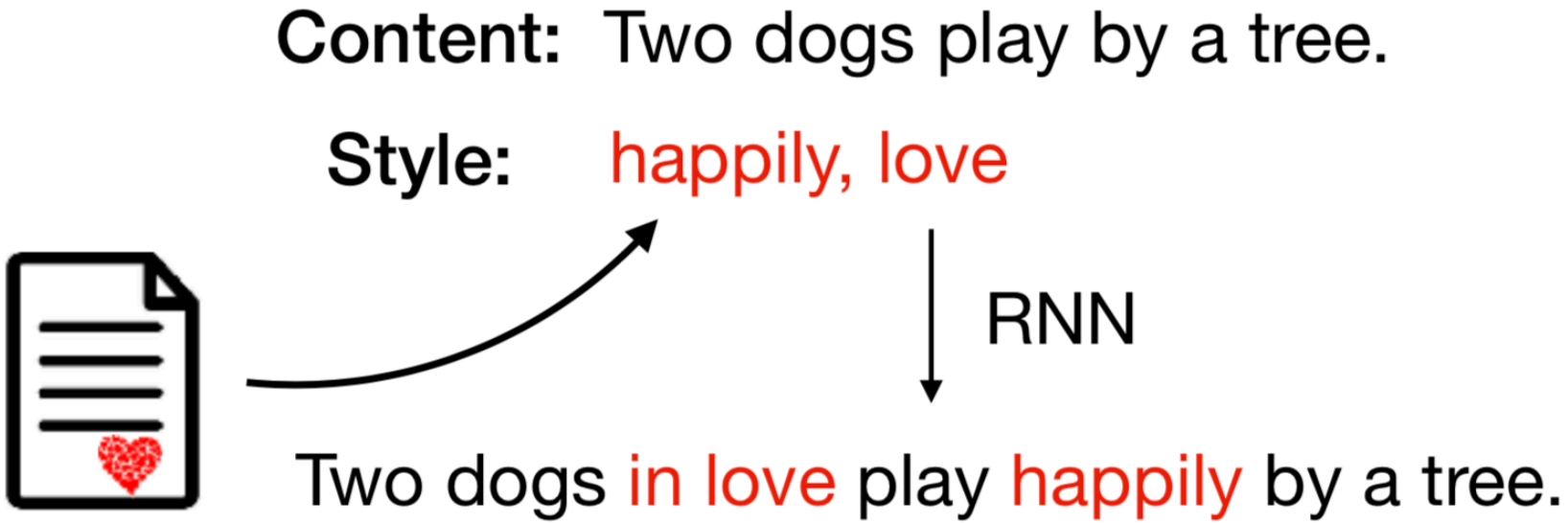
Machine Translation



https://www.pcmag.com/news/349610/google-expands-neural-networks-for-language-translation

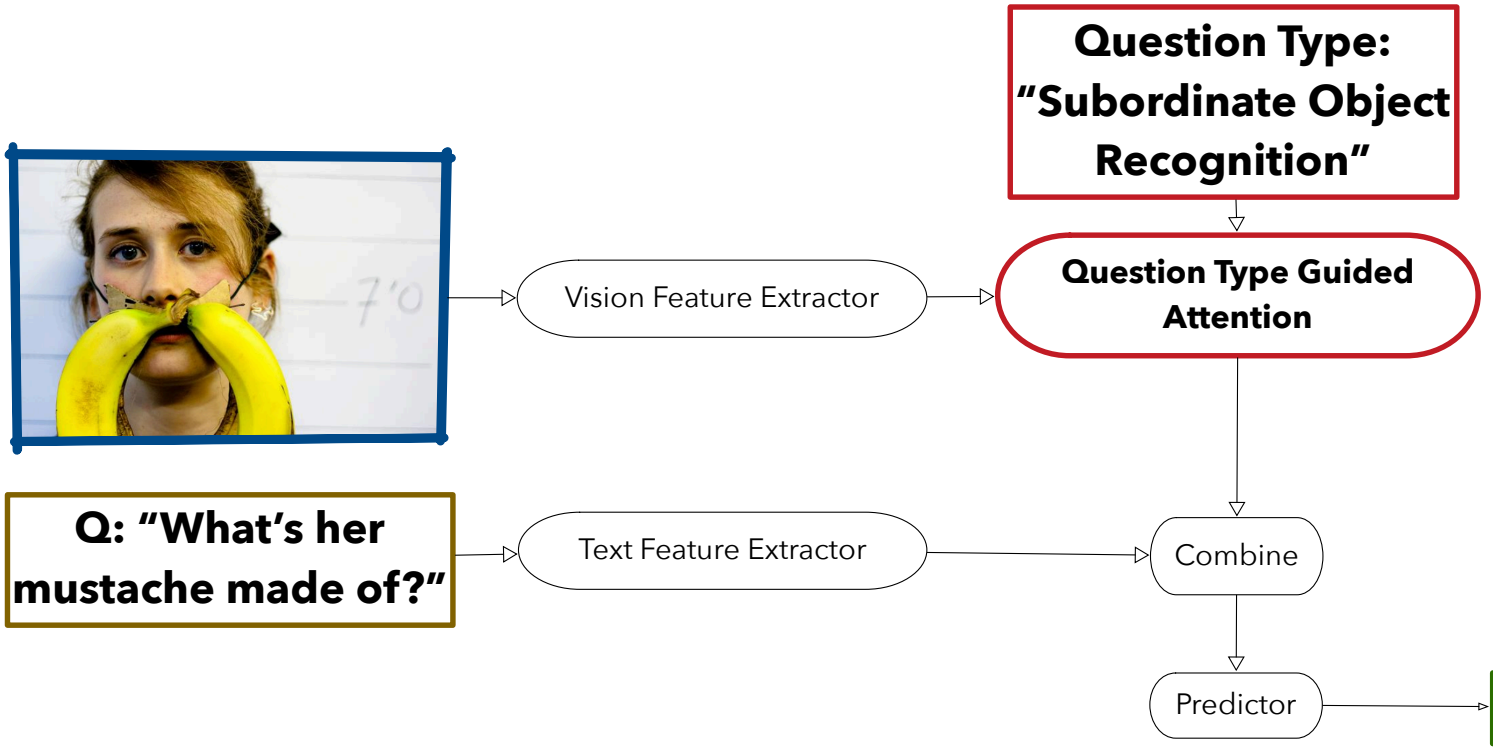
aws

Text synthesis



Li et al, NACCL, 2018

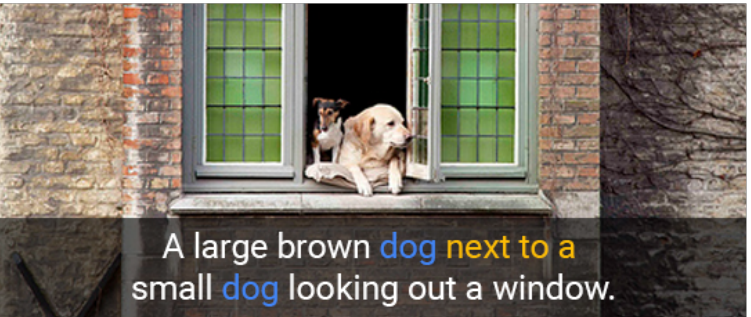
Question answering



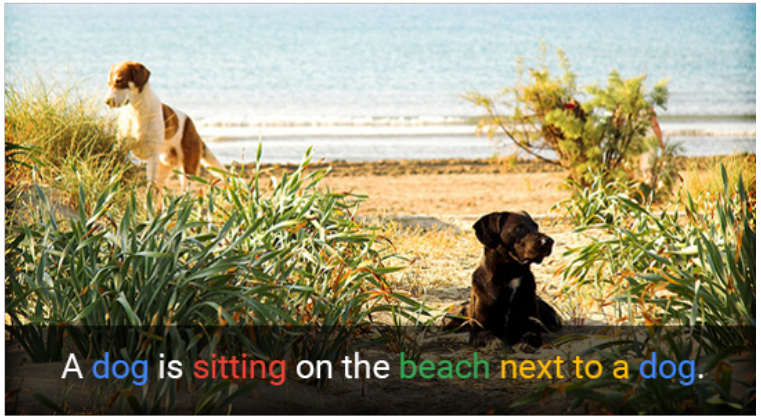
Shi et al, 2018, Arxiv

Image captioning

Human captions from the training set

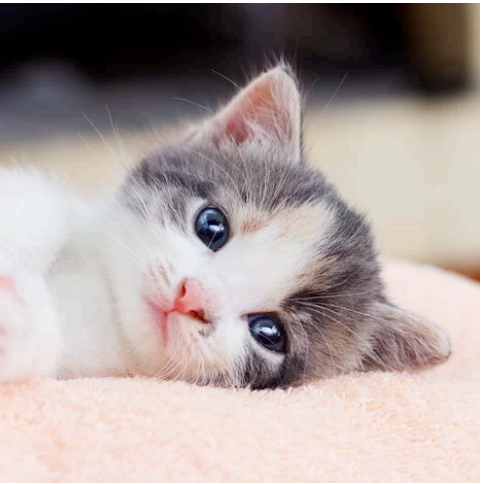


Automatically captioned



Shallue et al, 2016
https://ai.googleblog.com/2016/09/show-and-tell-image-captioning-open.html

Problems we will solve today: Classification



cat

dog

rabbit

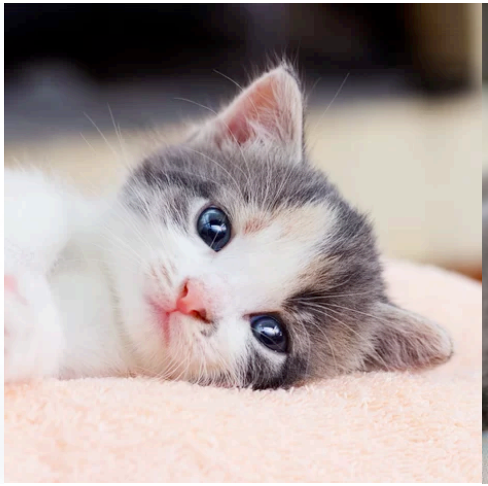
gerbil

Given image x estimate label y

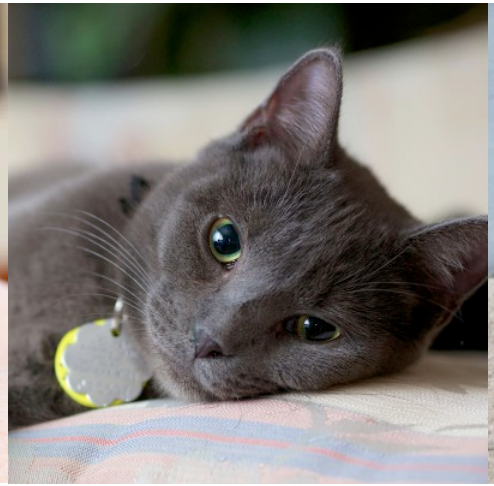
$$y = f(x) \text{ where } y \in \{1, \ldots N\}$$

Problems we will solve today: Regression

0.4kg



2kg



4kg

10kg

Given image x estimate label y

$$y = f(x) \text{ where } y \in \mathbb{R}$$

aws

Problems we will solve today: Sequence Models

SYSTEM PROMPT (HUMAN-WRITTEN)

A train carriage containing controlled nuclear materials was stolen in Cincinnati today. Its whereabouts are unknown.

MODEL COMPLETION (MACHINE-WRITTEN, FIRST TRY)

The incident occurred on the downtown train line, which runs from Covington and Ashland stations.

In an email to Ohio news outlets, the U.S. Department of Energy said it is working with the Federal Railroad Administration to find the thief.

“The theft of this nuclear material will have significant negative consequences on public and environmental health, our workforce and the economy of our nation,” said Tom Hicks, the U.S. Energy Secretary, in a statement. “Our top priority is to secure the theft and ensure it doesn’t happen again.”

GPT2, 2019


Installation

Tools

- **Python**
  - Everyone is using it in machine learning & data science
  - Conda package manager (for simplicity)
- **Jupyter**
  - So much easier to keep track of your experiments
  - Obviously you should put longer code into modules
- **Reveal (for notebook slides)**

  numpy.mxnet.io
  https://github.com/d2l-ai/d2l-en/tree/numpy2

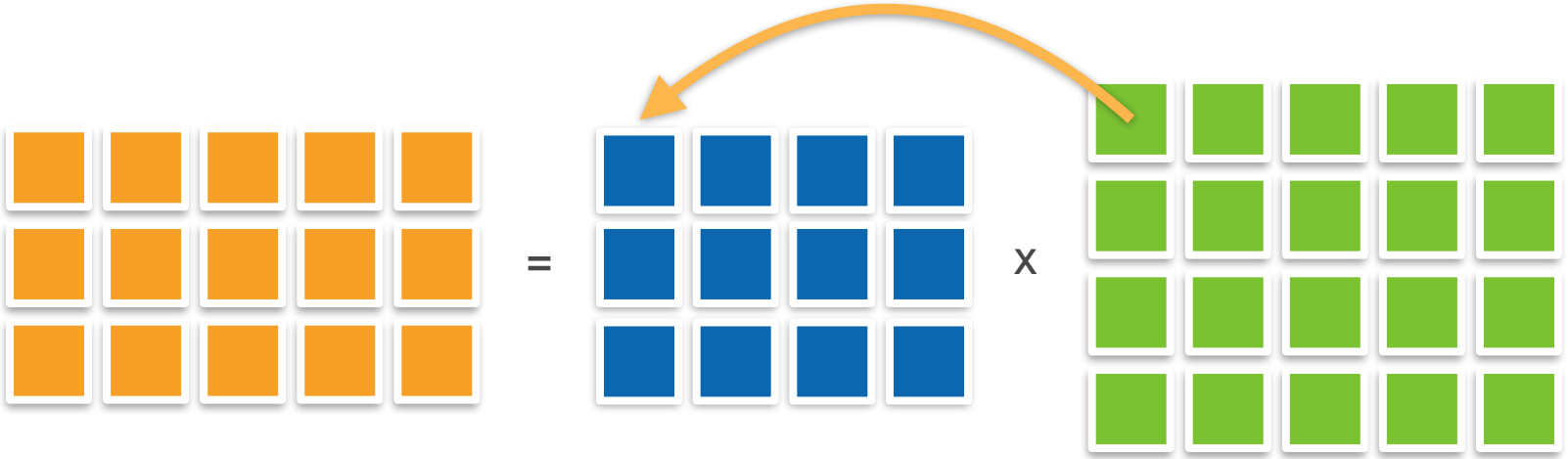
Linux and MacOS

```
conda create --name deepnumpy
conda activate deepnumpy
conda install pip
conda install jupyter
conda install rise

## get latest mxnet build with deep numpy
https://apache-mxnet.s3-us-west-2.amazonaws.com/dist/
python/numpy/latest/mxnet_cu100mkl-1.5.0-py2.py3-none-
manylinux1_x86_64.whl

pip install git+https://github.com/d2l-ai/d2l-en@numpy2
```

Linear Algebra



aws

Scalars

- **Simple operations**

$$c = a + b$$

$$c = a \cdot b$$

$$c = \sin a$$

- **Length**

$$|a| = \begin{cases} a & \text{if} a > 0 \\ -a & \text{otherwise} \end{cases}$$

$$|a + b| \leq |a| + |b|$$

$$|a \cdot b| = |a| \cdot |b|$$

Vectors

- **Simple operations**

$$c = a + b \quad \text{where } c_i = a_i + b_i$$
$$c = \alpha \cdot b \quad \text{where } c_i = \alpha b_i$$
$$c = \sin a \quad \text{where } c_i = \sin a_i$$

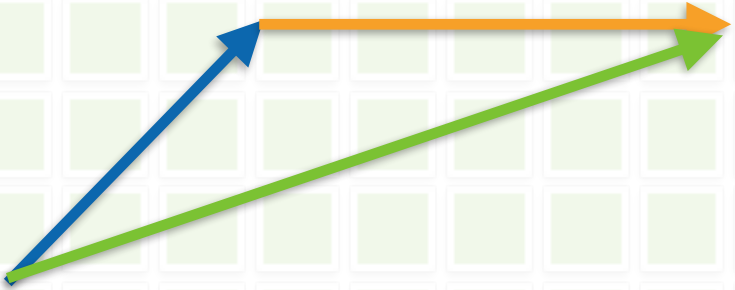
- **Length**

$$\|a\|_2 = \left[\sum_{i=1}^{m} a_i^2\right]^{\frac{1}{2}}$$

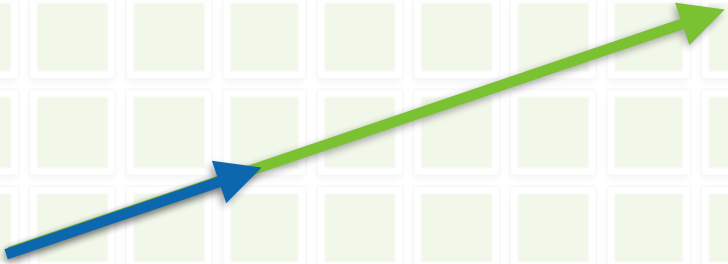
triangle inequality

$$\|a\| \geq 0 \text{ for all } a$$
$$\|a + b\| \leq \|a\| + \|b\|$$
$$\|a \cdot b\| = |a| \cdot \|b\|$$

Vectors



$$c = a + b$$

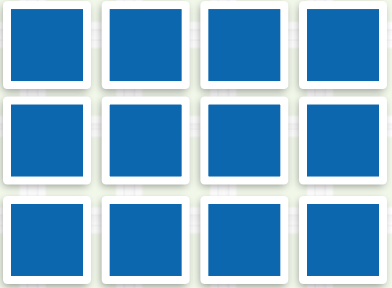


$$c = \alpha \cdot b$$

dot product

$$a^\top b = \sum_i a_i b_i$$

Matrices



- **Operations**

$$C = A + B \quad \text{where } C_{ij} = A_{ij} + B_{ij}$$

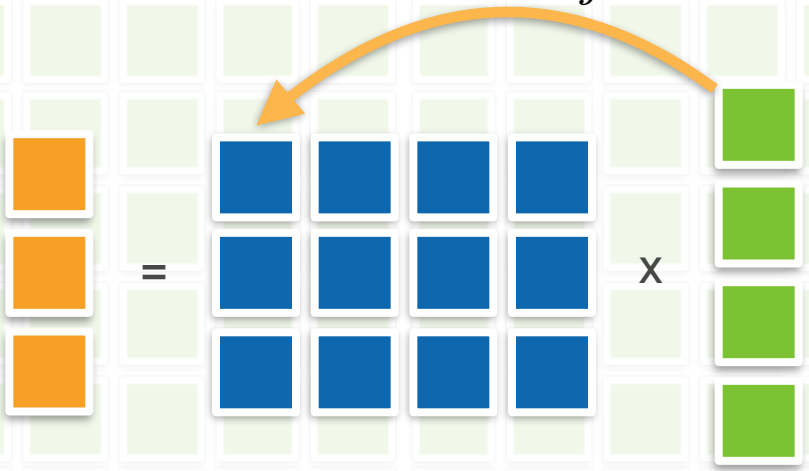
$$C = \alpha \cdot B \quad \text{where } C_{ij} = \alpha B_{ij}$$

$$C = \sin A \quad \text{where } C_{ij} = \sin A_{ij}$$

Matrices

- **Multiplications (matrix vector)**

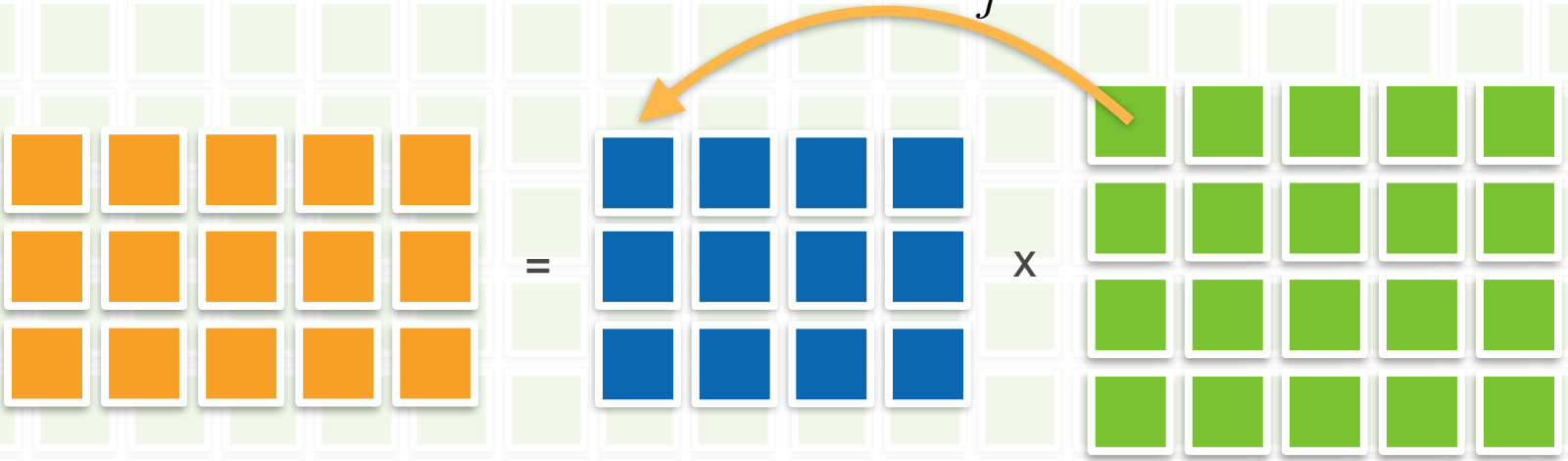
$$c = Ab \text{ where } c_i = \sum_j A_{ij} b_j$$



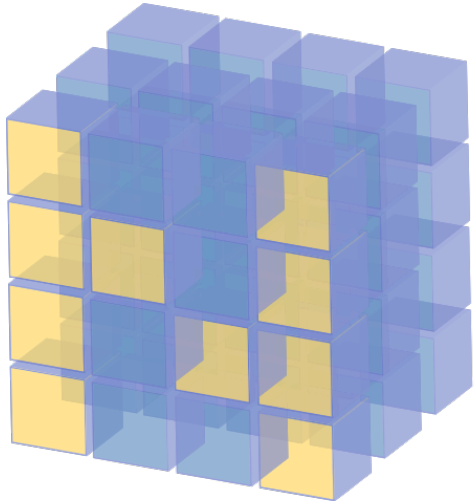
Matrices

- **Multiplications (matrix matrix)**

$$C = AB \text{ where } C_{ik} = \sum_j A_{ij} B_{jk}$$




GPUs love matrices and vectors
(they have many processors)

Deep  NumPy

numpy.d2l.ai

aws

N-dimensional Arrays

N-dimensional arrays are the main data structure for machine learning and neural networks

0-d (scalar)	1-d (vector)	2-d (matrix)
		

`1.0`	`[1.0, 2.7, 3.4]`	`[[1.0, 2.7, 3.4]`
		`[5.0, 0.2, 4.6]`
		`[4.3, 8.5, 0.2]]`

A class label	A feature vector	An example-by-feature matrix

N-dimensional Arrays

3-d

```
[[[0.1, 2.7, 3.4]
  [5.0, 0.2, 4.6]
  [4.3, 8.5, 0.2]]
 [[3.2, 5.7, 3.4]
  [5.4, 6.2, 3.2]
  [4.1, 3.5, 6.2]]]
```

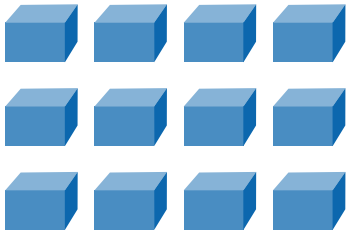
A RGB image (width x height x channels)

4-d

```
[[[[. . .
    . . .
    . . .]]]]
```

A batch of RGB images (batch-size x width x height x channels)

5-d



```
[[[[[. . .
     . . .
     . . .]]]]]
```

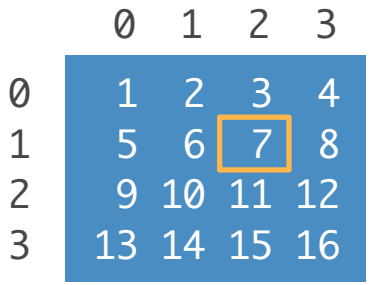
A batch of videos (batch-size x time x width x height x channels)

numpy.d2l.ai

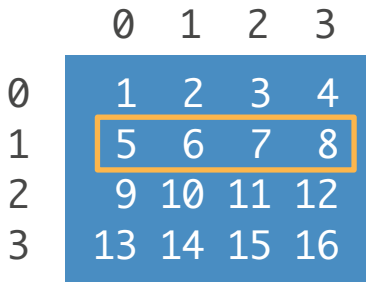
aws

Element-wise access

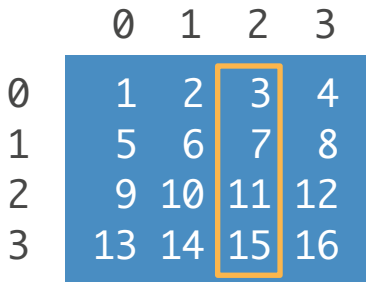
element: [1, 2]

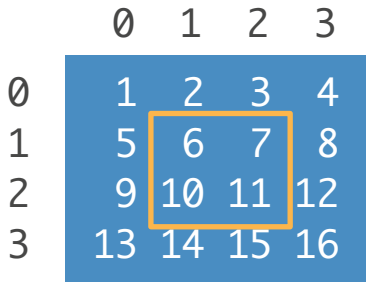


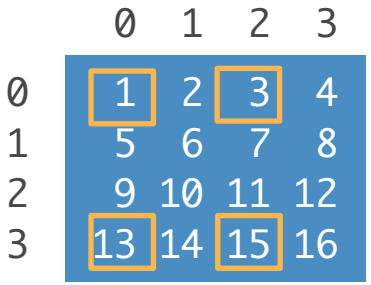
row: [1, :]



column: [1, :]







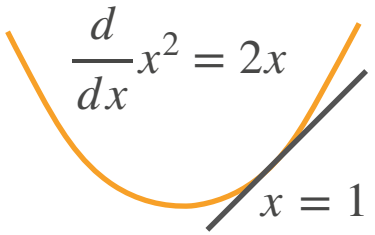
NumPy notebook

Automatic Differentiation

Calculus 101

y	a	x^n	$\exp(x)$	$\log(x)$	$\sin(x)$
$\frac{dy}{dx}$	0	nx^{n-1}	$\exp(x)$	$\frac{1}{x}$	$\cos(x)$

(slope of tangent)



y	$u + v$	uv	$y = f(u),\ u = g(x)$
$\frac{dy}{dx}$	$\frac{du}{dx} + \frac{dv}{dx}$	$\frac{du}{dx}v + \frac{dv}{dx}u$	$\frac{dy}{du}\frac{du}{dx}$

Derivatives for vectors

		Scalar	Vector
		x	$\mathbf{x}$
Scalar	y	$\frac{\partial y}{\partial x}$	$\frac{\partial y}{\partial \mathbf{x}}$
Vector	$\mathbf{y}$	$\frac{\partial \mathbf{y}}{\partial x}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$

$$\left[\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right]_{ij} = \frac{\partial y_i}{\partial x_j}$$

Examples

y	a	au	$\text{sum}(\mathbf{x})$	$\|\mathbf{x}\|^2$
$\frac{\partial y}{\partial \mathbf{x}}$	$\mathbf{0}^T$	$a\frac{\partial u}{\partial \mathbf{x}}$	$\mathbf{1}^T$	$2\mathbf{x}^T$

0 and **1** are vectors

y	$u + v$	uv	$\langle \mathbf{u}, \mathbf{v} \rangle$
$\frac{\partial y}{\partial \mathbf{x}}$	$\frac{\partial u}{\partial \mathbf{x}} + \frac{\partial v}{\partial \mathbf{x}}$	$\frac{\partial u}{\partial \mathbf{x}}v + \frac{\partial v}{\partial \mathbf{x}}u$	$\mathbf{u}^T\frac{\partial \mathbf{v}}{\partial \mathbf{x}} + \mathbf{v}^T\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$

Chain Rule

Scalars

$$y = f(u),\ u = g(x) \qquad \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x}$$

Vectors

$$\frac{\partial y}{\partial \mathbf{x}} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial \mathbf{x}} \qquad \frac{\partial y}{\partial \mathbf{x}} = \frac{\partial y}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{x}} \qquad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

$$(1,n)\quad (1,)\ (1,n) \qquad (1,n)\quad (1,k)\ \ (k,n) \qquad (m,n)\ \ (m,k)\ (k,n)$$

Automatic Differentiation

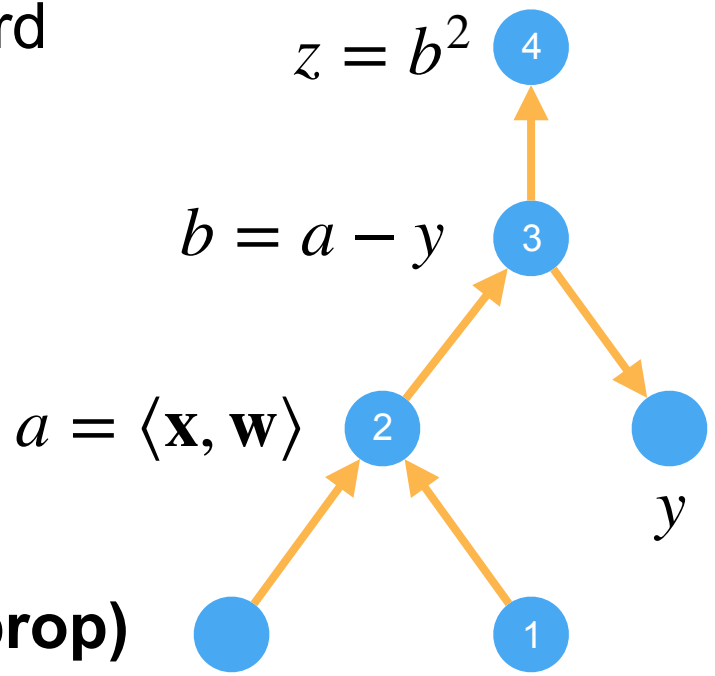
$$z = \left(\langle \mathbf{x}, \mathbf{w} \rangle - y\right)^2$$

Computing derivatives by hand is hard

Computers can automate this

Compute graph

- Build explicitly
  (TensorFlow, MXNet Symbol)
- **Build implicitly by tracing**
  (Chainer, PyTorch, DeepNumpy)



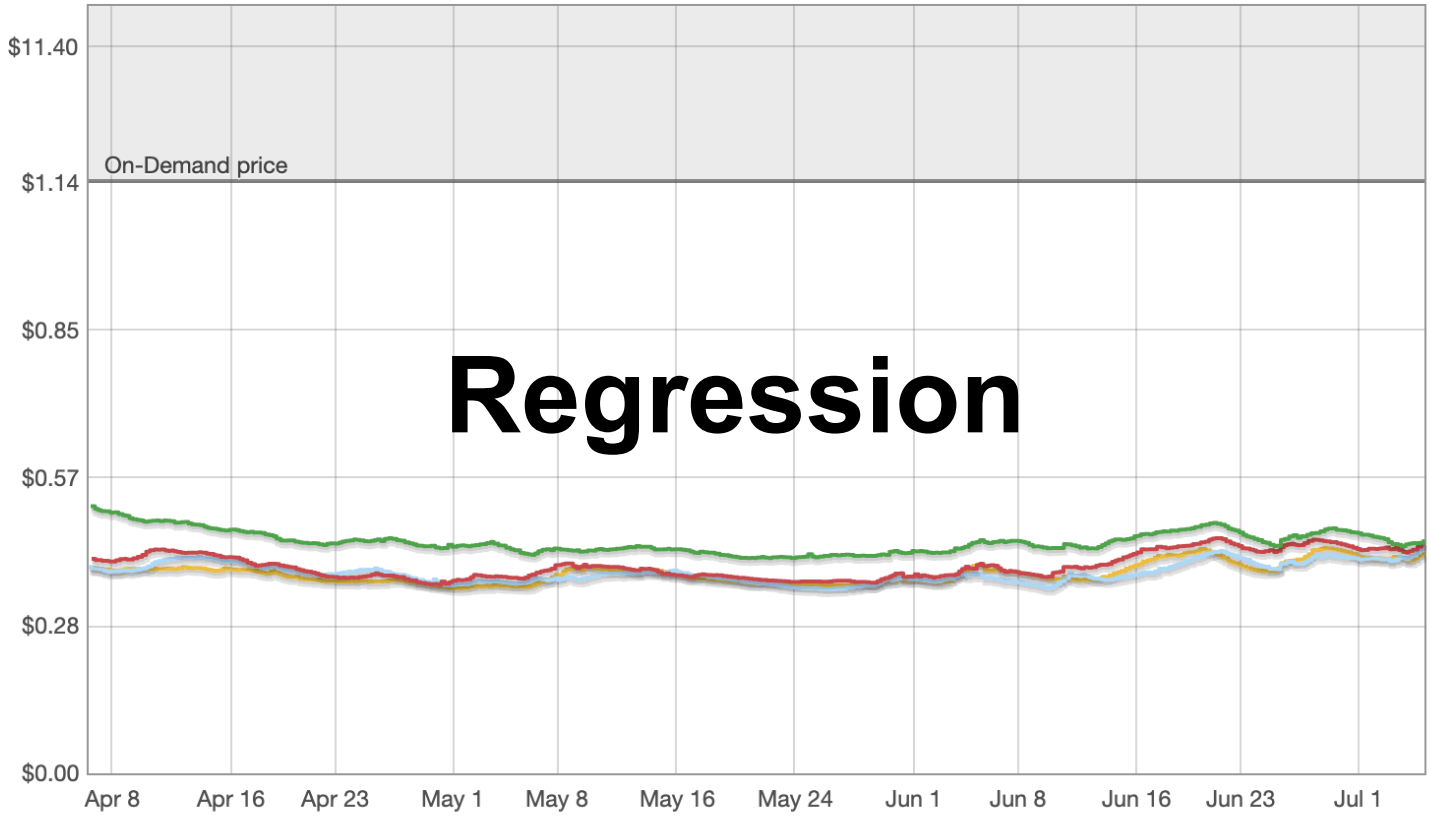
Chain rule (evaluate e.g. via backprop)

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u_n}\frac{\partial u_n}{\partial u_{n-1}}\cdots\frac{\partial u_2}{\partial u_1}\frac{\partial u_1}{\partial x}$$

AutoGrad notebook

Spot Instance Pricing History

Product: Linux/UNIX Instance type: g3.4xlarge Date range: 3 months



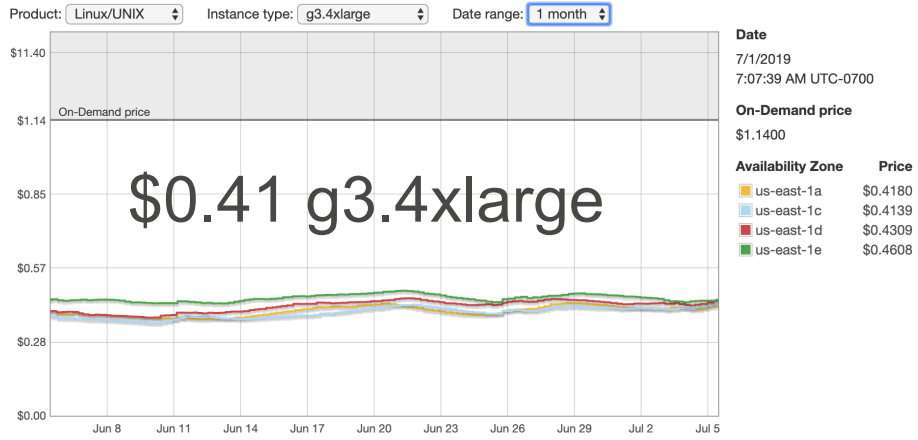
Date

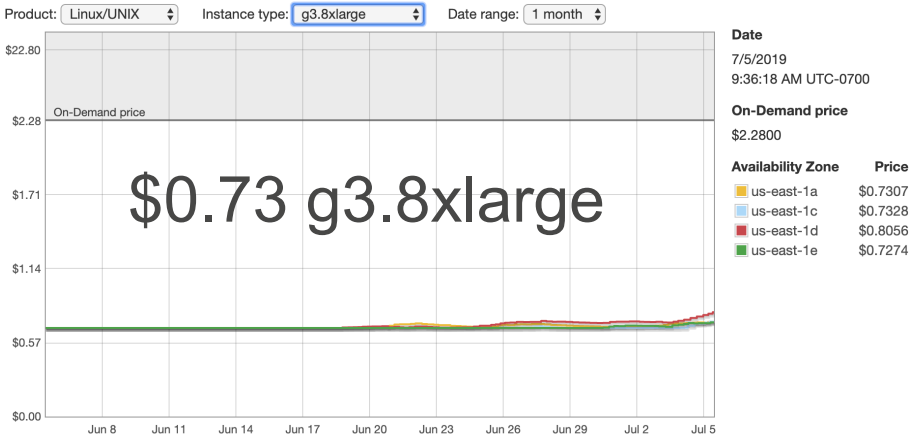
6/10/2019
9:47:38 PM UTC-0700

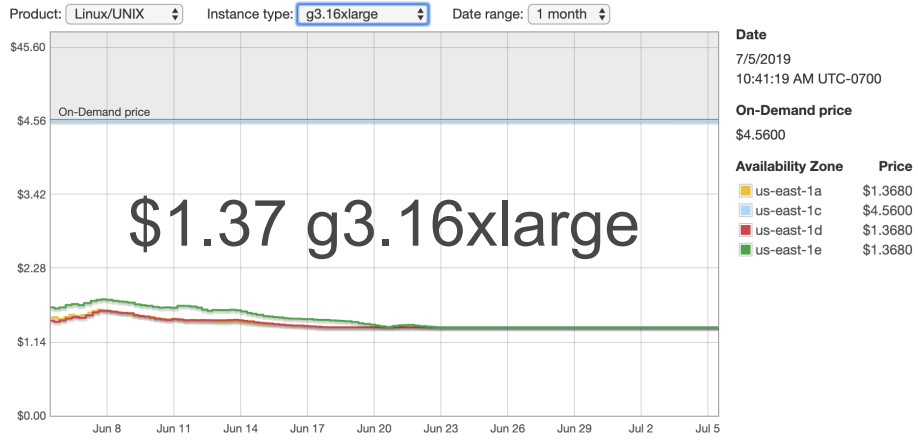
On-Demand price

$1.1400

Availability Zone	Price
us-east-1a	$0.3741
us-east-1c	$0.3710
us-east-1d	$0.3893
us-east-1e	$0.4353







Can we estimate
prices (time, server, region)?

$$p = w_{\text{time}} \cdot t + w_{\text{server}} \cdot s + w_{\text{region}}[r]$$

aws

Linear Model

- n-dimensional inputs $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$
- Linear model

$$\hat{y} = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$$

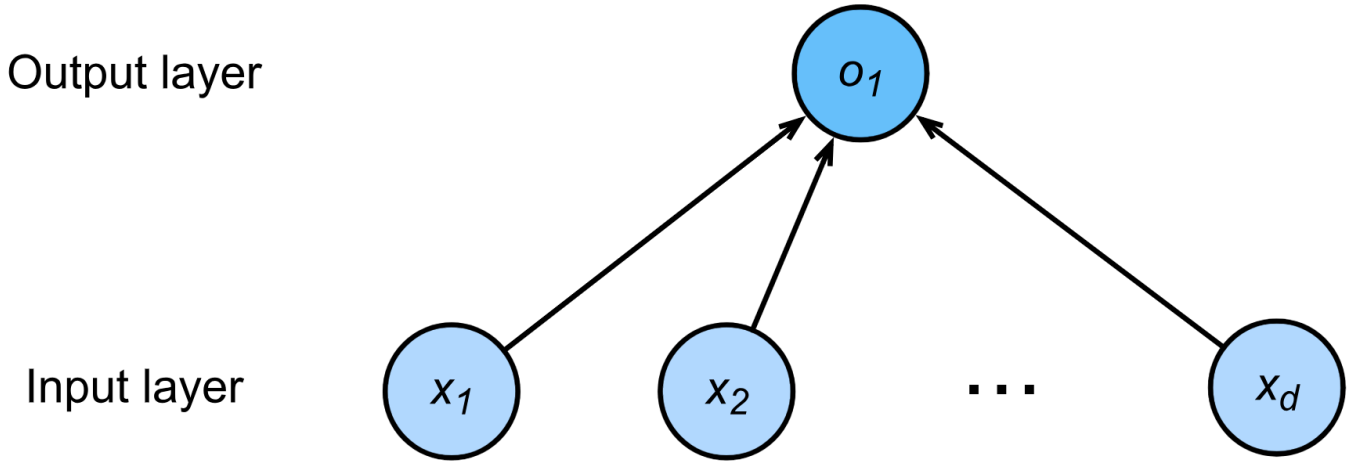
- Weights and bias $\mathbf{w} = [w_1, w_2, \ldots, w_n]^\top$ and b
- Vectorized version

$$\hat{y} = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

- Loss function (to measure goodness of fit)

$$l(y, \hat{y}) = (y - \hat{y})^2$$

Linear Model as a Single-layer Neural Network



We can stack multiple layers to get deep neural networks

Training and Test Data

- Collect multiple data points to fit parameters (spot instance prices in the past 6 months)
- Training data - the more the better

$$\mathbf{X} = \left[\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n\right]^T \qquad \mathbf{y} = \left[y_0, y_1, \ldots, y_n\right]^T$$

- Test data - when we actually want to try it out

$$\mathbf{X}' = \left[\mathbf{x}'_0, \mathbf{x}'_1, \ldots, \mathbf{x}'_{n'}\right]^T$$

We do not observe labels at time of prediction (after all, that's what we use ML for)

Training

- Objective is to minimize (regularized) training loss

$$\underset{w}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} l(y_i, \hat{y}_i)$$

- Plugging in expansion $\hat{y} = \langle \mathbf{w}, \mathbf{x} \rangle + b$ yields objective

$$\frac{1}{n} \sum_{i=1}^{n} (y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - b)^2 = \frac{1}{n} \| \mathbf{y} - \mathbf{X}\mathbf{w} - b \|^2$$

Linear Regression notebook

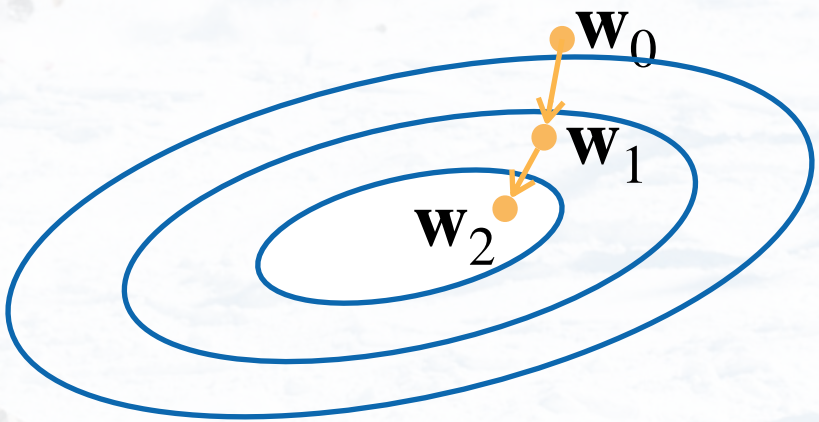
Stochastic Gradient Descent

Gradient Descent

Choose a starting point $\mathbf{w}_0$

Repeat to update weight

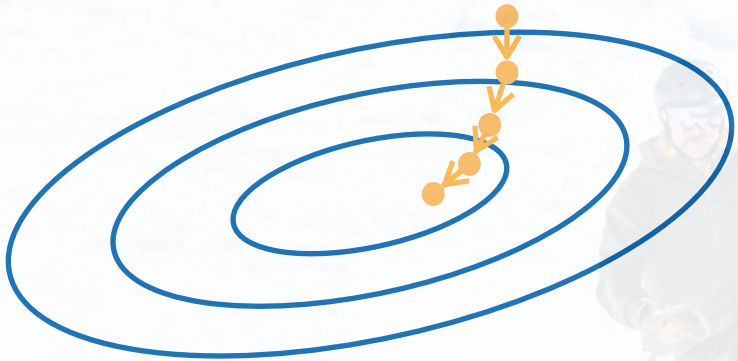
$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \partial_w l(w_{t-1})$$



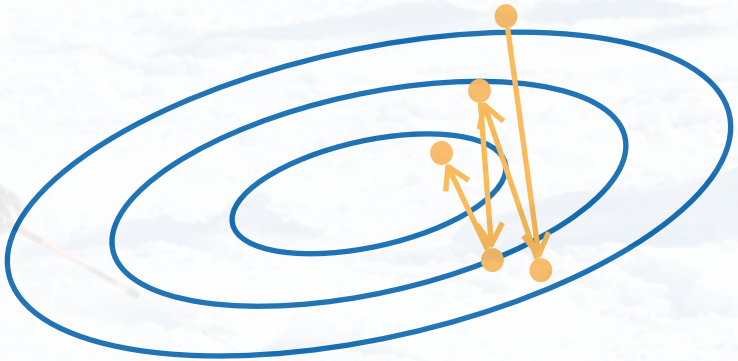
- Gradient direction indicates increase in value
- Learning rate adjusts step length

Goldilocks Learning Rate

Too small

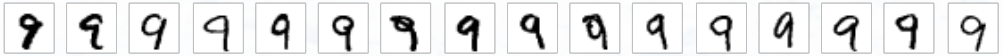


Too big



Stochastic Gradient Descent (SGD)

- Computing the gradient over all data is too slow
- Redundancy in data (e.g. many similar digits)



- Single observation is not efficient on GPU

- Sample b examples $i_1, \ldots i_b$ to approximate loss/gradient

$$\frac{1}{b}\sum_{i \in I_b} l(\mathbf{x}_i, y_i, \mathbf{w}) \text{ and } \frac{1}{b}\sum_{i \in I_b} \partial_{\mathbf{w}} l(\mathbf{x}_i, y_i, \mathbf{w})$$

b is the mini-batch size (chosen for GPU efficiency)

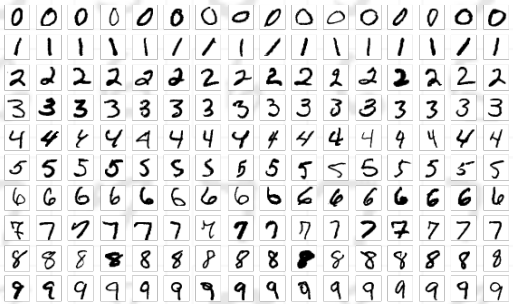

Logistic Regression

Regression vs. Classification

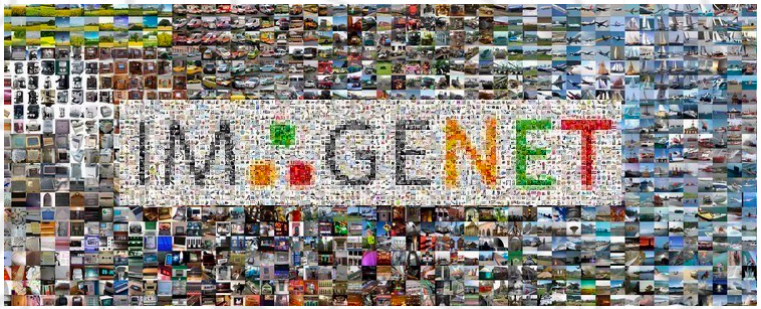
Regression estimates a continuous value

Classification predicts a discrete category

MNIST: classify hand-written digits (10 classes)



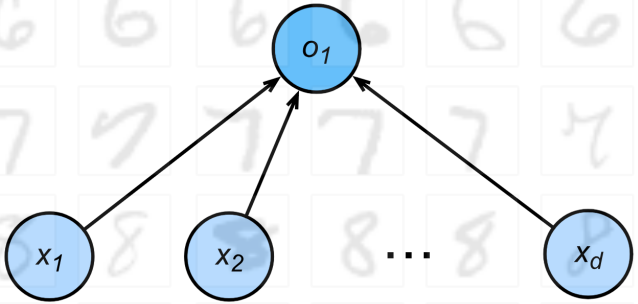
ImageNet: classify nature objects (1000 classes)



From Regression to Multi-class Classification

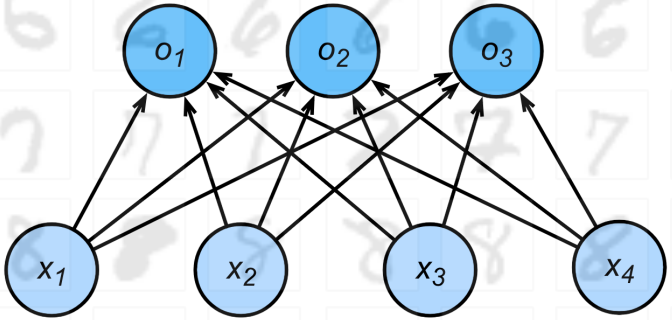
Regression

- Single continuous output
- Natural scale in $\mathbb{R}$
- Loss given e.g. in terms of difference $y - f(x)$



Classification

- Multiple classes, typically multiple outputs
- Score *should* reflect confidence …



From Regression to Multi-class Classification

Square Loss

- One hot encoding per class

$$\mathbf{y} = [y_1, y_2, \ldots, y_n]^\top$$

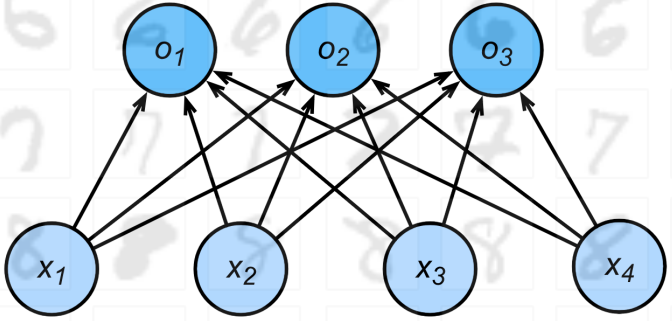
$$y_i = \begin{cases} 1 \text{ if } i = y \\ 0 \text{ otherwise} \end{cases}$$

- Train with squared loss
- Largest output wins

$$\hat{y} = \operatorname*{argmax}_i o_i$$

Classification

- Multiple classes, typically multiple outputs
- Score *should* reflect confidence …



From Regression to Multi-class Classification

Calibrated Scale

- Output matches probabilities (nonnegative, sums to 1)

$$p(y \mid o) = \mathrm{softmax}(o)$$

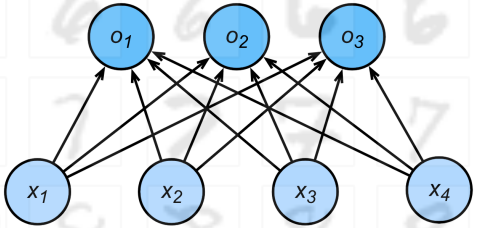
$$= \frac{\exp(o_y)}{\sum_i \exp(o_i)}$$

- Negative log-likelihood

$$-\log p(y \mid y) = \log \sum_i \exp(o_i) - o_y$$

Classification

- Multiple classes, typically multiple outputs
- Score *should* reflect confidence …



Softmax and Cross-Entropy Loss

- Negative log-likelihood (for given label y)

$$-\log p(y \mid o) = \log \sum_i \exp(o_i) - o_y$$

- Cross-Entropy Loss (for probability distribution y)

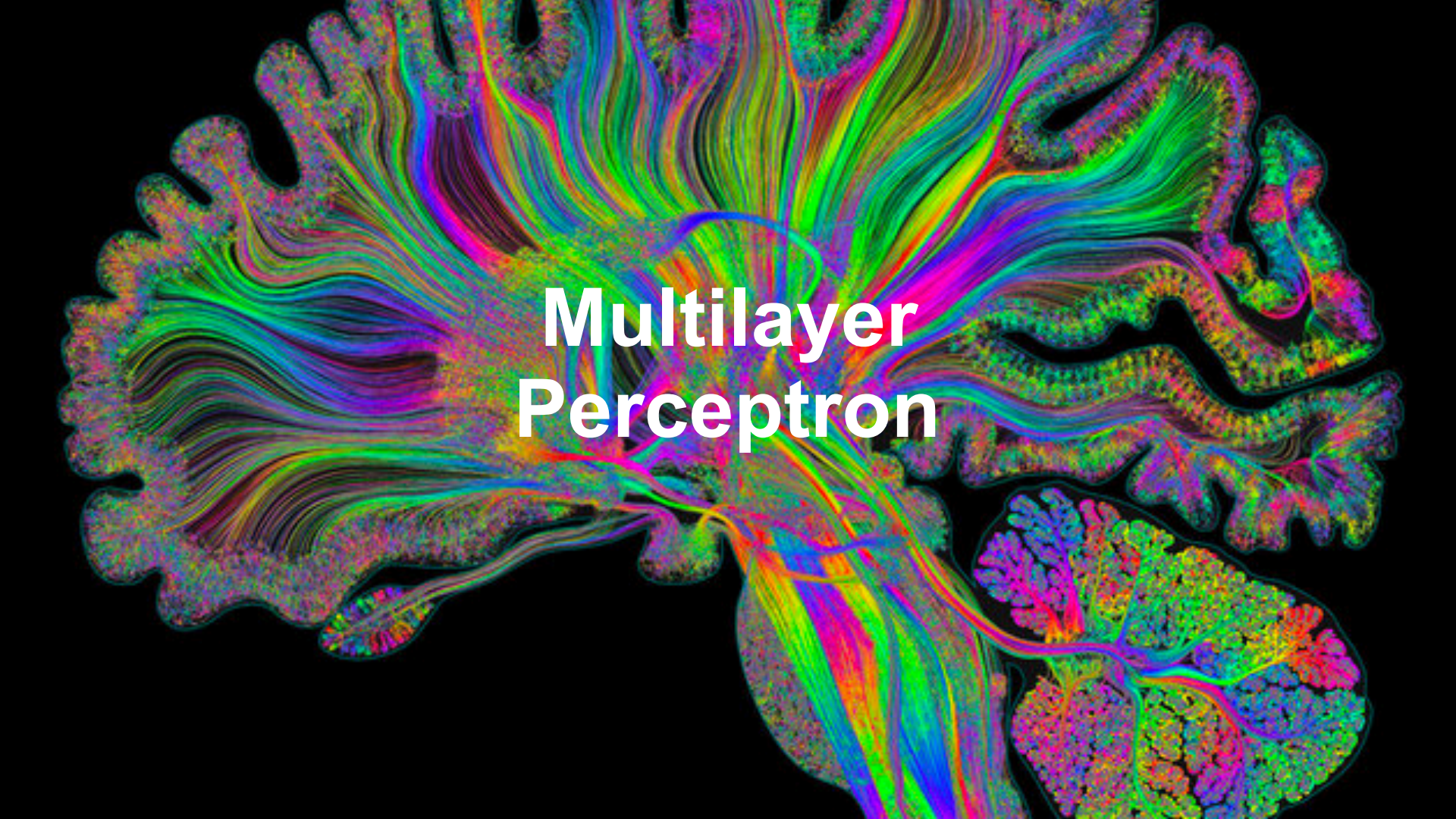
$$l(y, o) = \log \sum_i \exp(o_i) - y^\top o$$

- **Gradient**

Difference between true and estimated probability

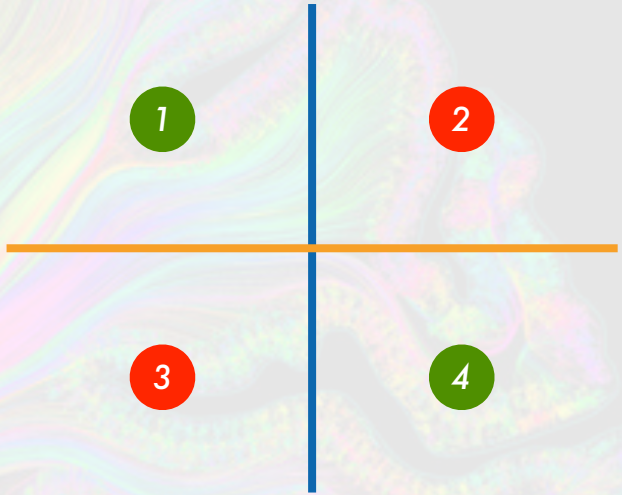
$$\partial_o l(y, o) = \frac{\exp(o)}{\sum_i \exp(o_i)} - y$$

Logistic Regression Notebook

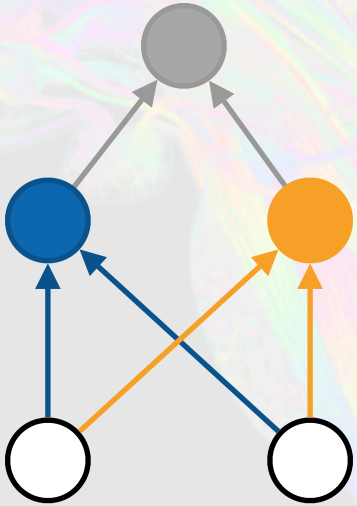

Multilayer
Perceptron

Learning XOR

	1	2	3	4
	+	-	+	-
	+	+	-	-
product	+	-	-	+

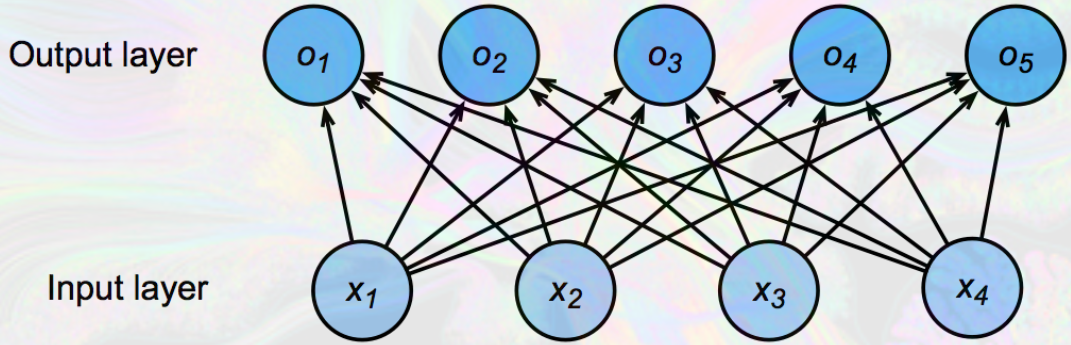


$$w_{11}x_1 + w_{12}x_2 + b_1$$

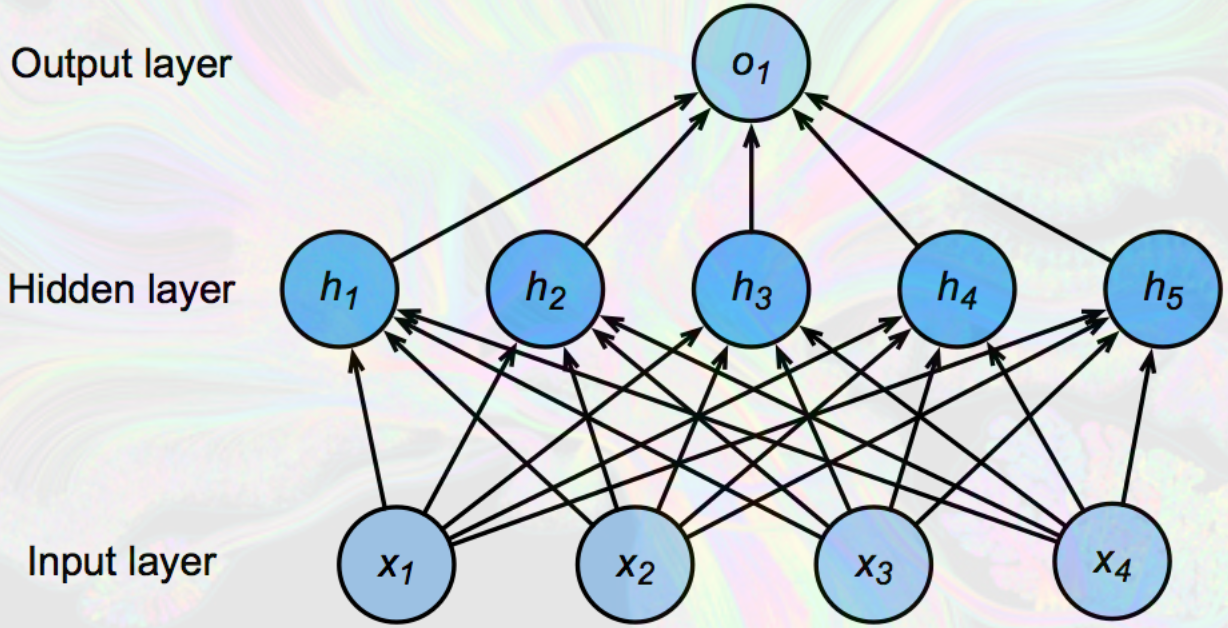


$$w_{21}x_1 + w_{22}x_2 + b_2$$

Single Hidden Layer



Single Hidden Layer



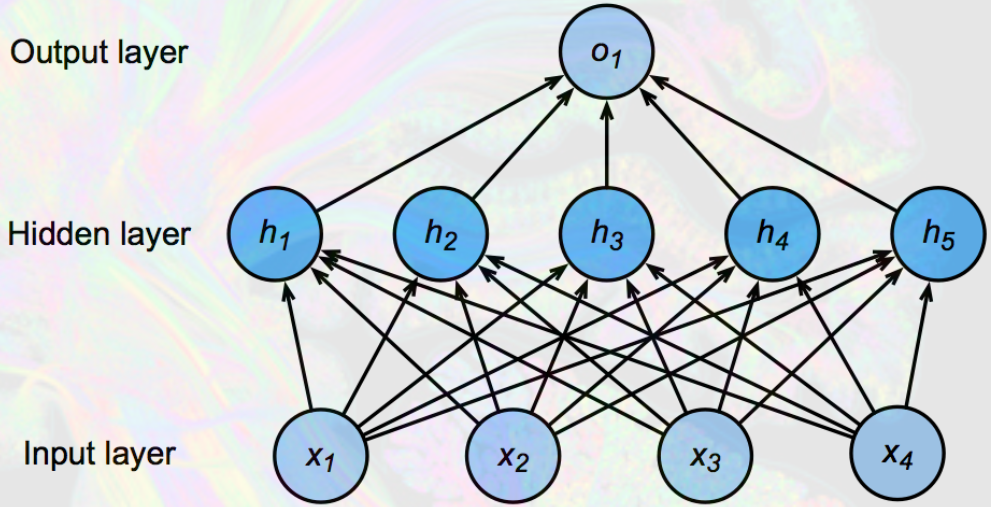
Single Hidden Layer

- Input $\mathbf{x} \in \mathbb{R}^n$
- Hidden $\mathbf{W}_1 \in \mathbb{R}^{m \times n}, \mathbf{b}_1 \in \mathbb{R}^m$
- Output $\mathbf{w}_2 \in \mathbb{R}^m, b_2 \in \mathbb{R}$

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{w}_2^T \mathbf{h} + b_2$$

σ is an element-wise activation function



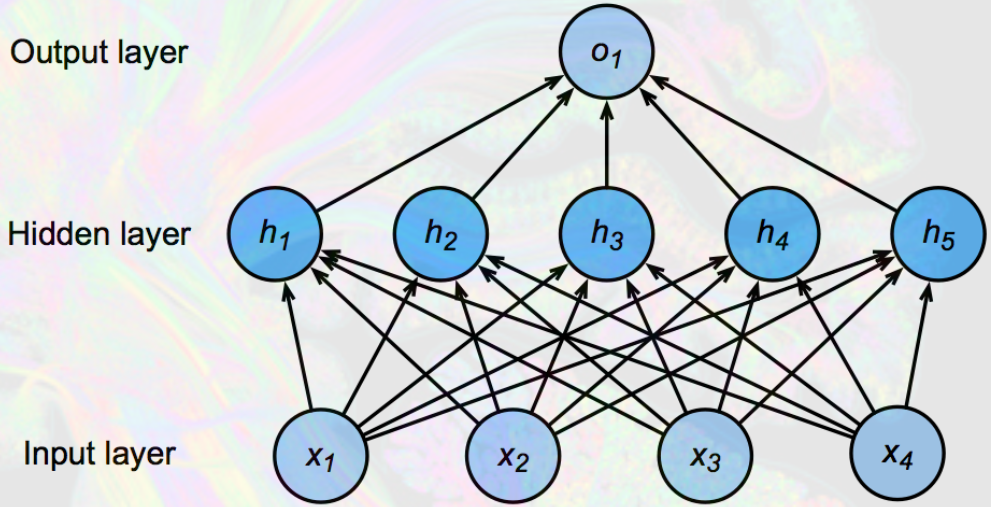
Single Hidden Layer

Why do we need an a nonlinear activation?

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{w}_2^T\mathbf{h} + b_2$$

σ is an element-wise activation function



Single Hidden Layer

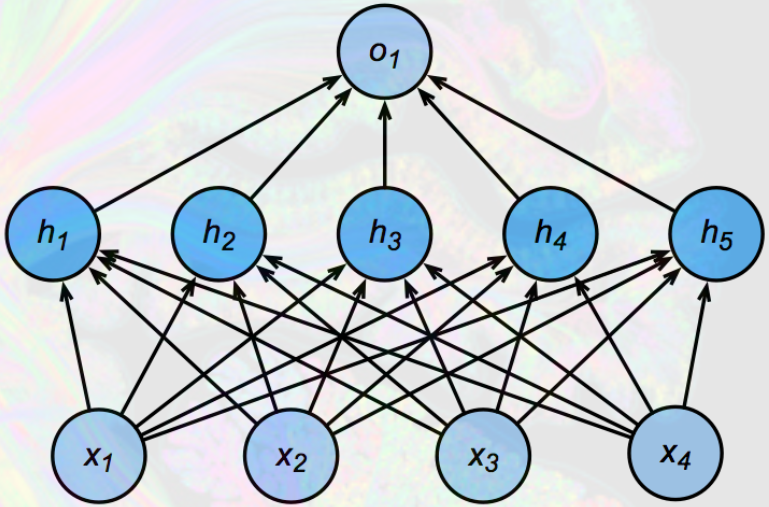
Why do we need an a nonlinear activation?

$$\mathbf{h} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$$

$$\mathbf{o} = \mathbf{w}_2^T\mathbf{h} + b_2$$

hence $o = \mathbf{w}_2^\top\mathbf{W}_1\mathbf{x} + b'$

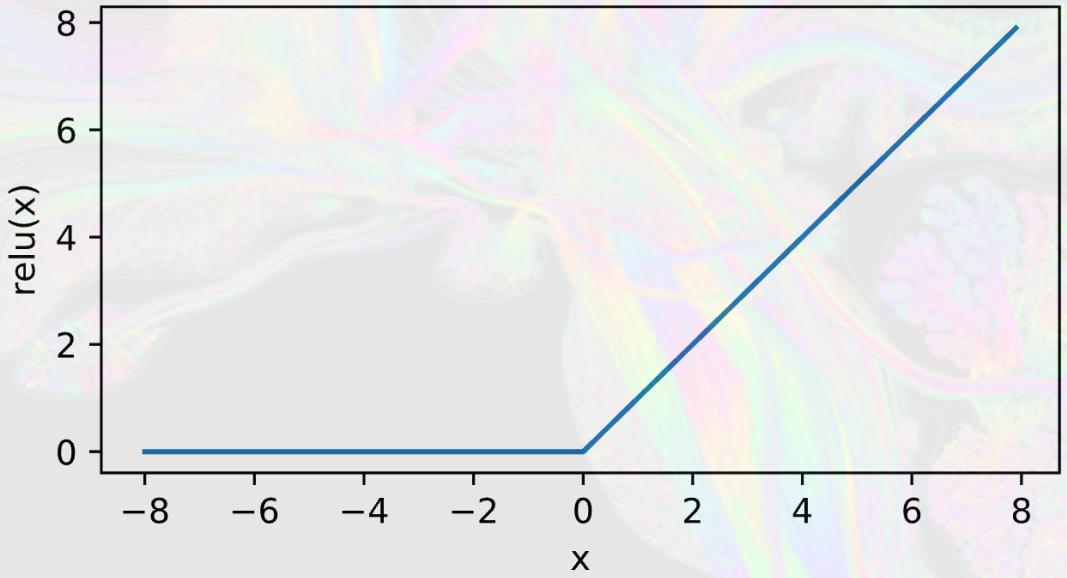
Linear …



ReLU Activation

ReLU: rectified linear unit

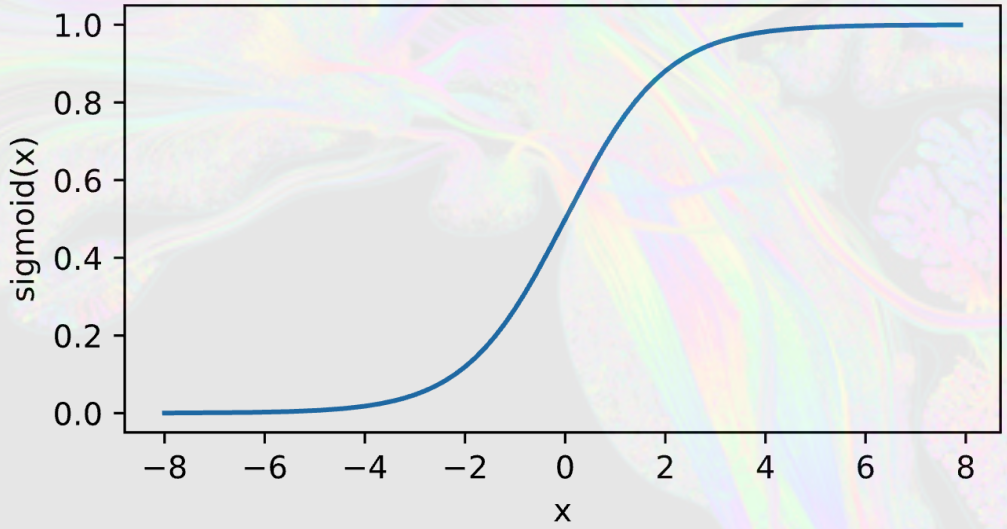
$$\mathrm{ReLU}(x) = \max(x, 0)$$



Sigmoid Activation

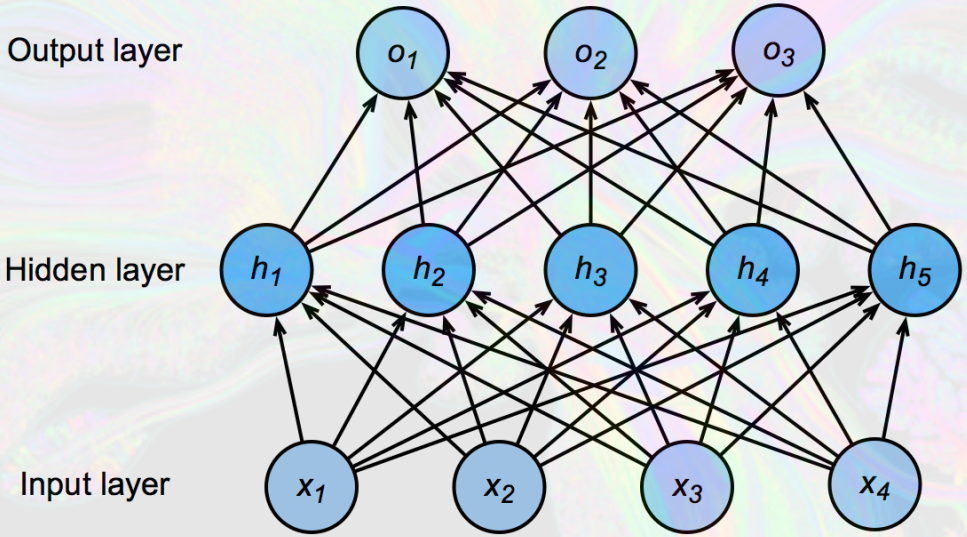
Map input into (0, 1), a soft version of $\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$



Multiclass Classification

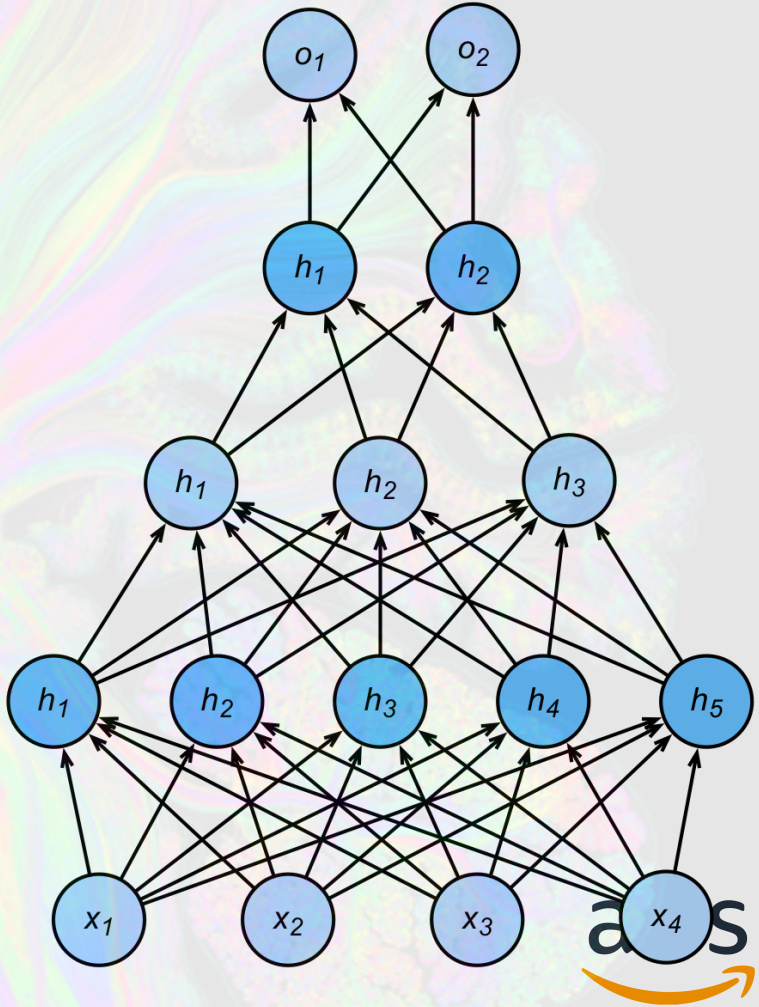
$$y_1, y_2, \ldots, y_k = \text{softmax}(o_1, o_2, \ldots, o_k)$$



Multiple Hidden Layers

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$$
$$\mathbf{h}_3 = \sigma(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$$
$$\mathbf{o} = \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4$$

Hyper-parameters
- # of hidden layers
- Hidden size for each layer



MLP Notebook

Summary

- **Introduction to Deep Learning**
- **Installation** (Cloud, Laptop, GitHub)
- **Linear Algebra and Deep Numpy**
- **Autograd**
- **Linear Regression**
- **Optimization**
- **Softmax Classification**
- **Multilayer Perceptron (train MNIST)**