

Dive into Deep Learning in 1 Day

1 Basics · **2 Convnets** · **3 Computation** · **4 Sequences**

ODSC 2019

Alex Smola

<http://courses.d2l.ai/odsc2019/>

Outline

- Installation
- Linear Algebra and Deep Numpy
- Autograd
- Softmax Classification
- Optimization
- Multilayer Perceptron

A close-up photograph showing the installation of a copper pipe fitting. Two copper pipes are being joined at a right angle. One pipe is secured with a large, adjustable wrench, while the other is held by a smaller wrench. A brass compression fitting is being slid onto the pipe, and a crimping tool is being used to secure it. The background shows other copper pipes and fittings.

Installation

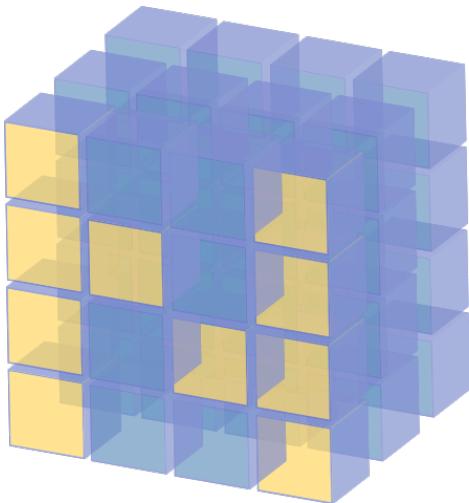
Tools

- Python
- Jupyter
- Reveal (for notebook slides)

```
conda create --name mxnet
conda activate mxnet
conda install python==3.7 pip jupyter rise
pip install d2l mxnet-cu100 -pre
pip install pip install git+https://github.com/d2l-ai/
d2l-en@numpy2
git clone https://github.com/d2l-ai/1day-notebooks.git
```

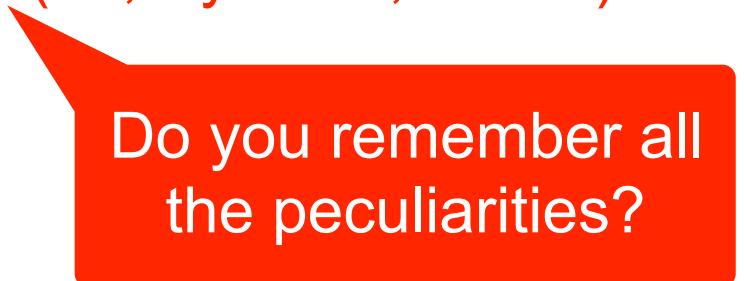
mxnet.np

Deep NumPy



Typical Deep Learning Workflow

- Get data and load it (e.g. in Pandas)
 - Preprocess it in NumPy
 - Convert to framework specific format
- Train model
 - Build model in specific format (TF, PyTorch, Gluon)
 - Optimize
- Convert predictions into NumPy
 - Plot and serve



Do you remember all
the peculiarities?

Matrix Multiplication in NumPy

```
import numpy as np  
from time import time
```

```
# testing NumPy  
n = 1024  
x = np.random.normal(0, 1, (n, n))  
y = np.random.normal(0, 1, (n, n))
```

Random normal matrix

```
# single invocation  
def single():  
    z = np.dot(x,y)  
timer(single)  
  
def multiple():  
    for i in range(1000):  
        z = np.dot(x,y)  
timer(multiple, True)
```

Matrix-matrix multiplication

0.065011 seconds
53.658205 seconds
4.002153 Gigaflops

Numbers from old 2015 MacBook Pro
(runs on GPU, too)



Matrix Multiplication in PyTorch

```
import numpy as np  
from time import time
```

```
# testing NumPy  
n = 1024  
x = np.random.normal(0, 1, (n, n))  
y = np.random.normal(0, 1, (n, n))
```

```
# single invocation  
def single():  
    z = np.dot(x,y)  
timer(single)  
  
def multiple():  
    for i in range(1000):  
        z = np.dot(x,y)  
timer(multiple, True)
```

0.065011 seconds
53.658205 seconds
4.002153 Gigaflops

Random
normal matrix

```
import torch
```

```
x = torch.randn(n,n)  
y = torch.randn(n,n)
```

```
# single invocation
```

```
def single():  
    z = torch.mm(x,y)  
timer(single)
```

```
def multiple():  
    for i in range(1000):  
        z = torch.mm(x,y)  
timer(multiple, True)
```

0.029826 seconds

22.214673 seconds

9.666960 Gigaflops

Matrix-matrix
multiplication



Matrix Multiplication in TensorFlow

```
import numpy as np  
from time import time
```

```
# testing NumPy  
n = 1024  
x = np.random.normal(0, 1, (n, n))  
y = np.random.normal(0, 1, (n, n))
```

```
# single invocation  
def single():  
    z = np.dot(x,y)  
timer(single)  
  
def multiple():  
    for i in range(1000):  
        z = np.dot(x,y)  
timer(multiple, True)
```

0.065011 seconds
53.658205 seconds
4.002153 Gigaflops

Random
normal matrix

```
import tensorflow
```

```
x = tf.random.normal([n, n])  
y = tf.random.normal([n, n])
```

```
# single invocation
```

```
def single():  
    z = tf.matmul(x,y)  
timer(single)
```

```
def multiple():  
    for i in range(1000):  
        z = tf.matmul(x,y)  
timer(multiple, True)
```

0.028445 seconds
19.343846 seconds
11.101638 Gigaflops

Matrix-matrix
multiplication



Matrix Multiplication in ...

```
import numpy as np
from time import time
```

```
# testing NumPy
n = 1024
x = np.random.normal(0, 1, (n, n))
y = np.random.normal(0, 1, (n, n))
```

```
# single invocation
def single():
    z = np.dot(x,y)
timer(single)

def multiple():
    for i in range(1000):
        z = np.dot(x,y)
timer(multiple, True)
```

0.065011 seconds
53.658205 seconds
4.002153 Gigaflops

- MXNet Gluon
- Caffe2
- CNTK
- Chainer
- Paddle
- Keras

Confusing



Deep NumPy

```
import numpy as np
from time import time
```

```
# testing NumPy
n = 1024
x = np.random.normal(0, 1, (n, n))
y = np.random.normal(0, 1, (n, n))
```

```
# single invocation
def single():
    z = np.dot(x,y)
timer(single)

def multiple():
    for i in range(1000):
        z = np.dot(x,y)
timer(multiple, True)
```

0.065011 seconds
53.658205 seconds
4.002153 Gigaflops

```
from mxnet import np as np
```

```
# testing Deep NumPy
x = np.random.normal(0, 1, (n, n))
y = np.random.normal(0, 1, (n, n))
```

```
# single invocation
def single():
    z = np.dot(x,y)
timer(single)

def multiple():
    for i in range(1000):
        z = np.dot(x,y)
timer(multiple)
```

```
def multiple_touch():
    for i in range(1000):
        z = np.dot(x,y)
        z.wait_to_read()
timer(multiple_touch, True)
```

0.000663 seconds
0.008336 seconds
18.248058 seconds
11.768286 Gigaflops

Barrier

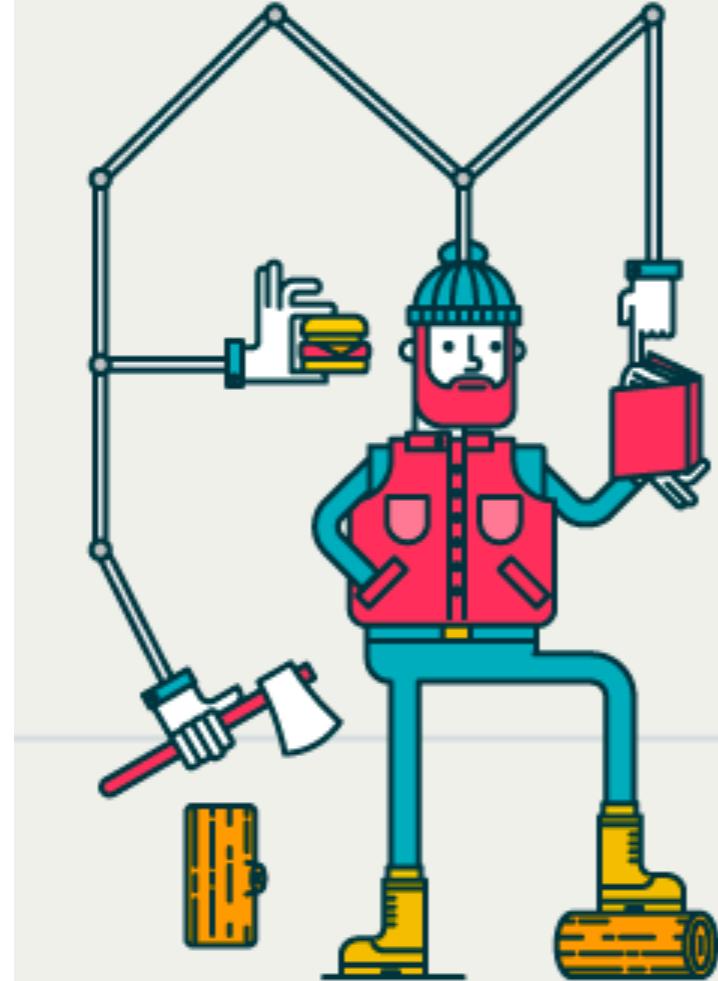


Frontend for Deep NumPy (on MXNet)

- **np (numpy drop-in replacement)**
 - 100% NumPy compatible* (arguments for device context)
 - Asynchronous evaluation
- **autograd**
- **npx (numpy extensions)**
 - Devices
 - Convolutions and other DL operators
- **nn (neural networks)**
 - Layers, optimizers, loss functions ... (MXNet Gluon)

NumPy notebook

Automatic Differentiation



Automatic Differentiation

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

Computing derivatives by hand is hard

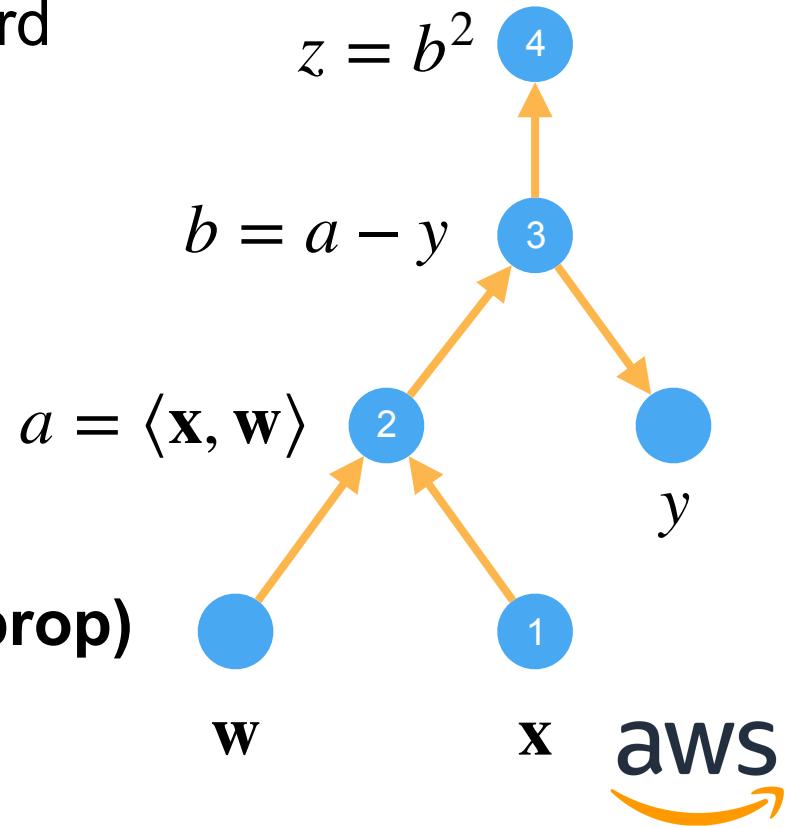
Computers can automate this

Compute graph

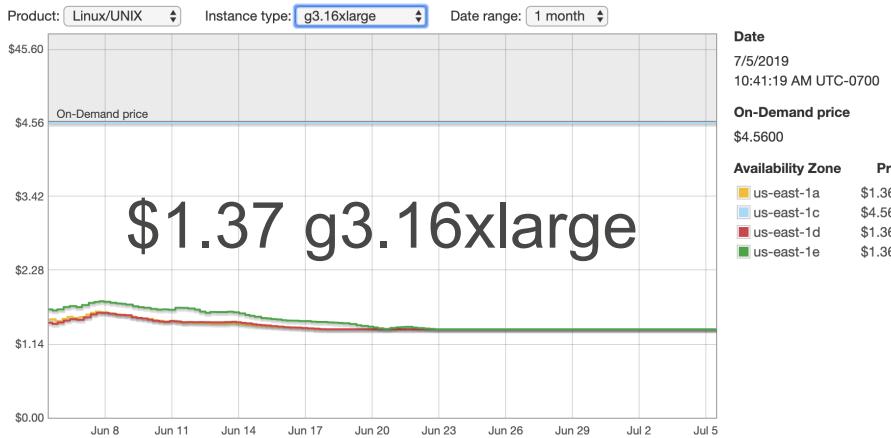
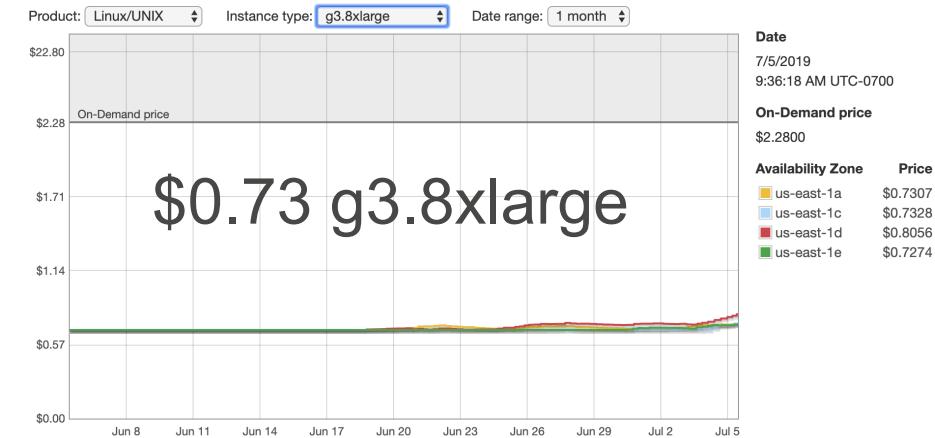
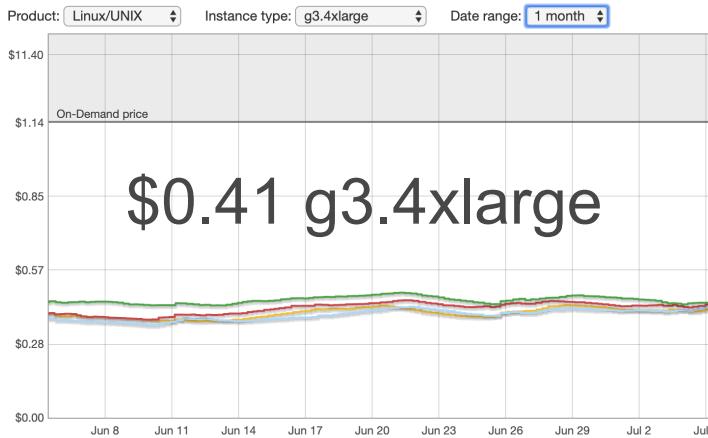
- Build explicitly
(TensorFlow, MXNet Symbol)
- Build implicitly by tracing
(Chainer, PyTorch, DeepNumPy)

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



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

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

Linear Model

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

- Linear model

$$\hat{y} = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

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

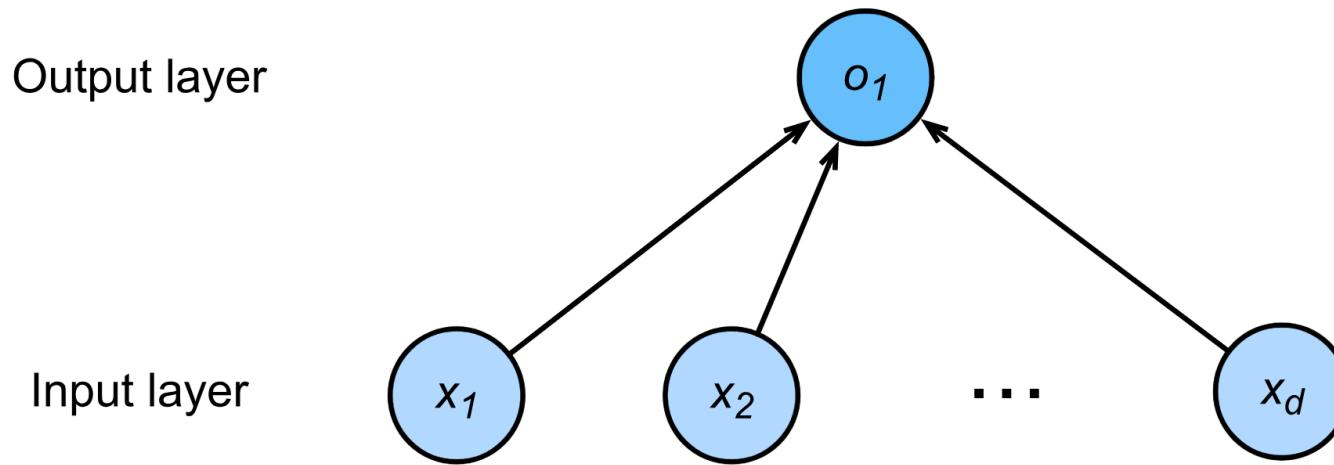
- Vectorized version

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

- 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

$$\mathbf{X} = [\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_n]^T \quad \mathbf{y} = [y_0, y_1, \dots, y_n]^T$$

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

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

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

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

- 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

Logistic Regression

Regression vs. Classification

Regression estimates a continuous value

Classification predicts a discrete category

MNIST: classify hand-written digits (10 classes)

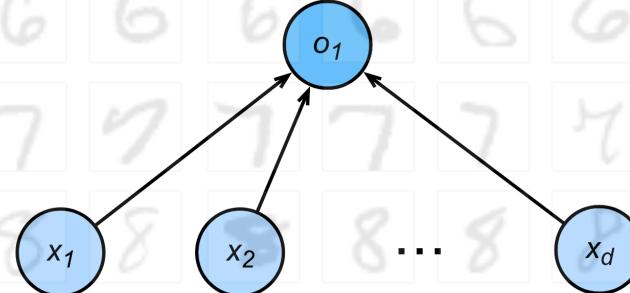
ImageNet: classify nature objects (1000 classes)



From Regression to Multi-class Classification

Regression

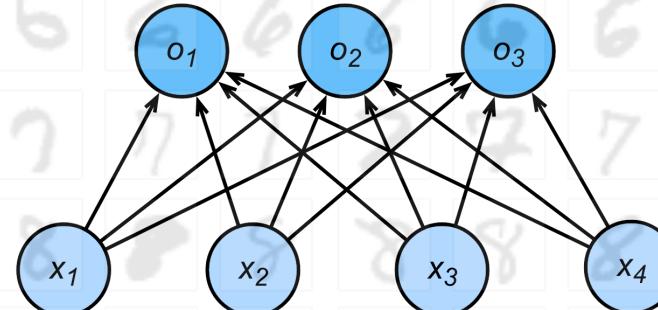
- Single continuous output
- Natural scale in \mathbb{R}
- Loss given e.g. in terms of difference $y - f(x)$



numpy.d2l.ai

Classification

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



aws

From Regression to Multi-class Classification

Square Loss

- One hot encoding per class

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

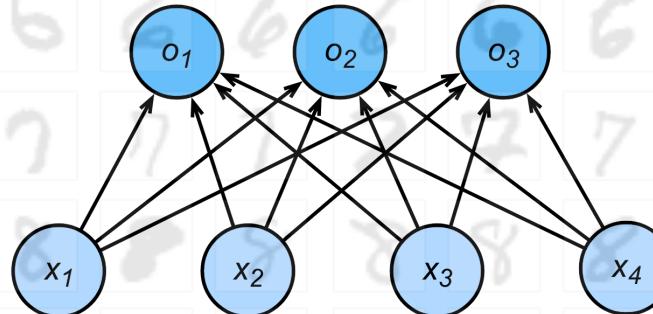
$$y_i = \begin{cases} 1 & \text{if } i = y \\ 0 & \text{otherwise} \end{cases}$$

- 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)

$$p(y|o) = \text{softmax}(o)$$

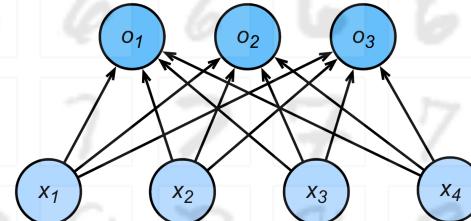
$$= \frac{\exp(o_y)}{\sum_i \exp(o_i)}$$

- Negative log-likelihood

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

Classification

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



Softmax and Cross-Entropy Loss

- Negative log-likelihood (for given label y)

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

- Cross-Entropy Loss (for probability distribution y)

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

- Gradient

Difference between true
and estimated probability

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



Logistic Regression Notebook

Stochastic Gradient Descent

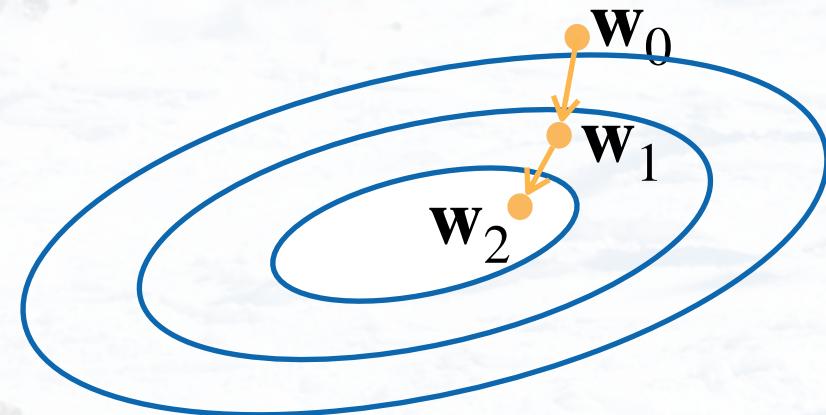


Gradient Descent

Choose a starting point \mathbf{w}_0

Repeat to update weight

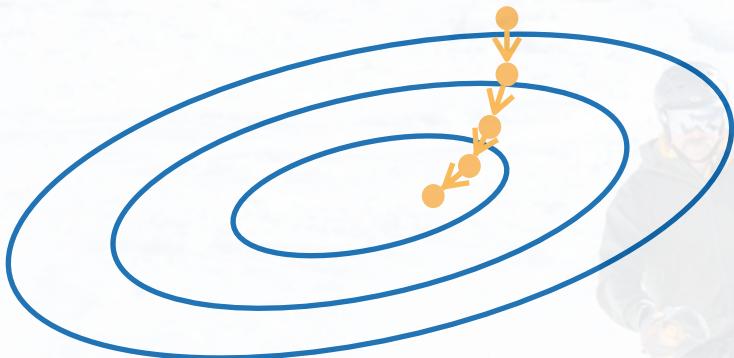
$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \partial_w l(w_{t-1})$$



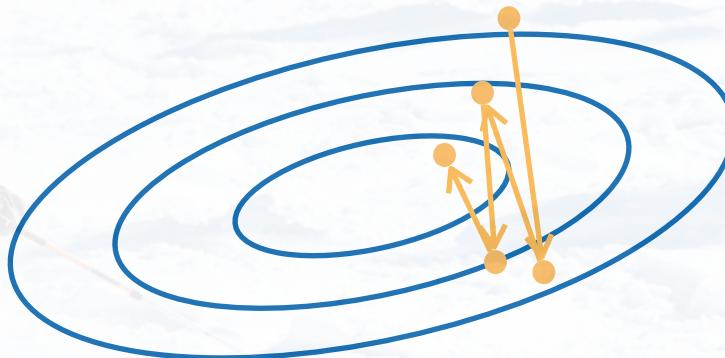
- Gradient direction indicates increase in value
- Learning rate adjusts step length

Goldilocks Learning Rate

Too small



Too big



Stochastic Gradient Descent (SGD)

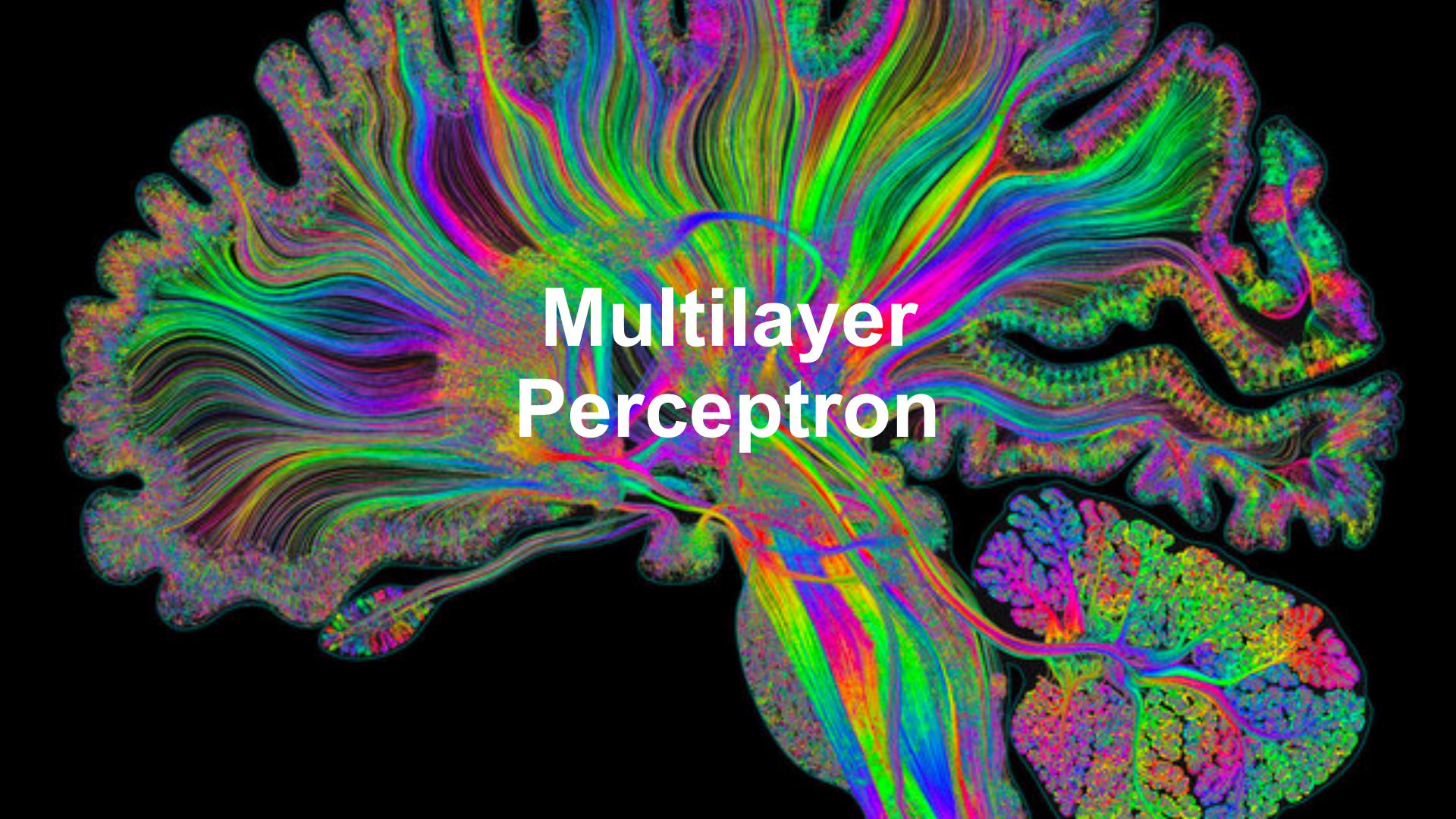
- Computing the gradient over all data is too slow
- Redundancy in data (e.g. many similar digits)



- Single observation is not efficient on GPU
- Sample b examples i_1, \dots, i_b to approximate loss/gradient

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

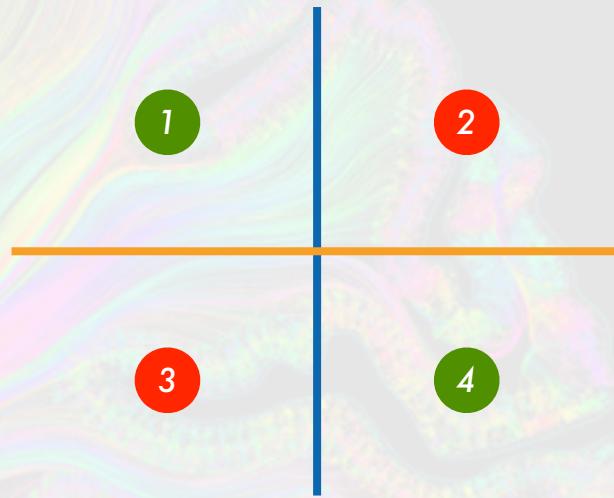
b is the mini-batch size (chosen for GPU efficiency)



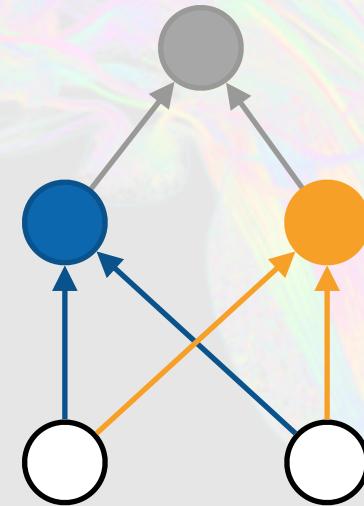
Multilayer Perceptron

Learning XOR

	1	2	3	4
blue	+	-	+	-
orange	+	+	-	-
product	+	-	-	+

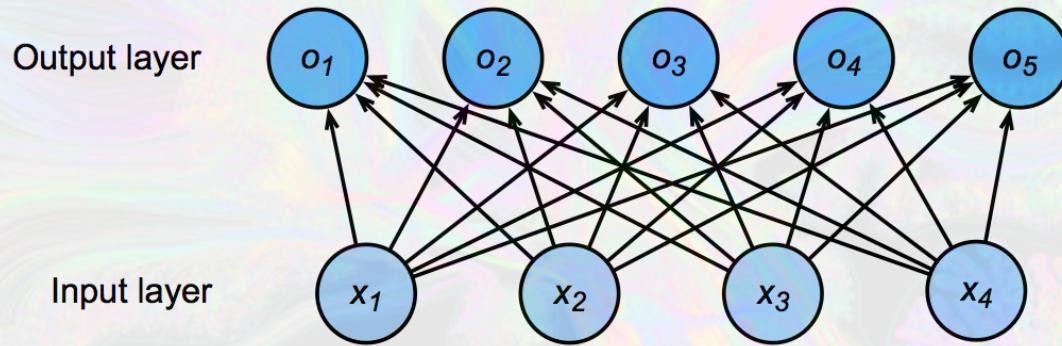


$$w_{11}x_1 + w_{12}x_2 + b_1$$

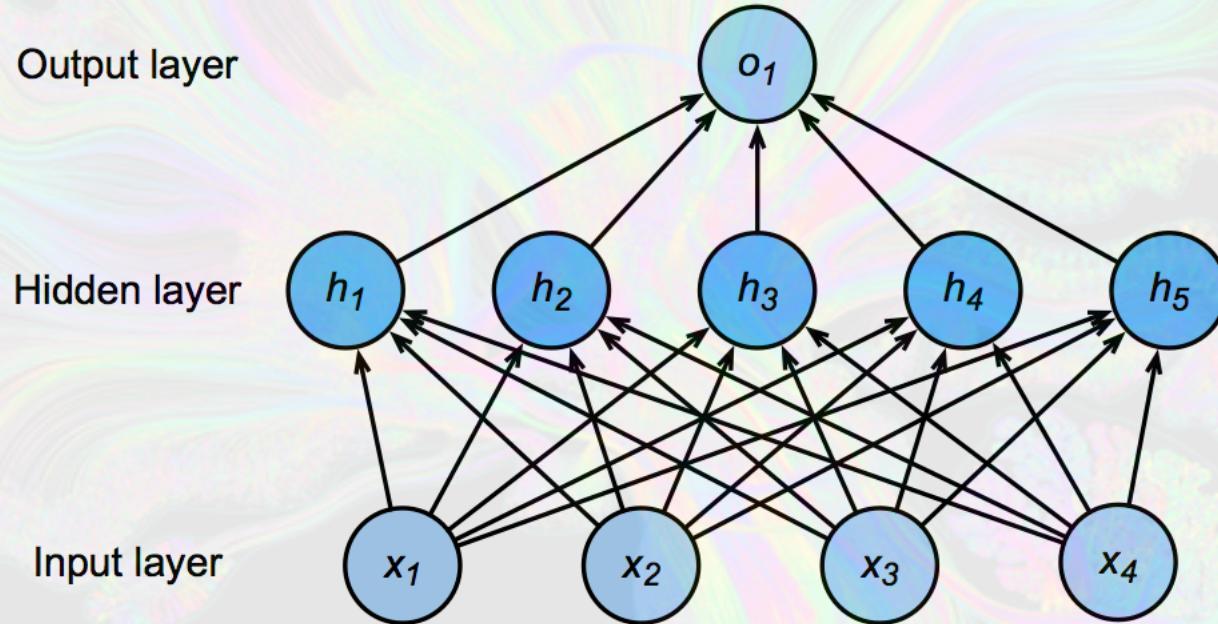


$$w_{21}x_1 + w_{22}x_2 + b_2$$

Single Hidden Layer



Single Hidden Layer



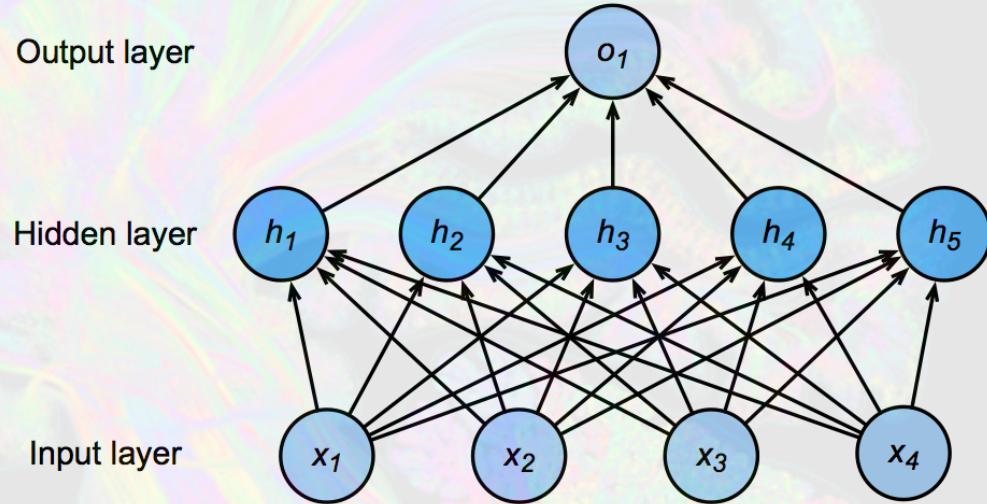
Single Hidden Layer

- 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



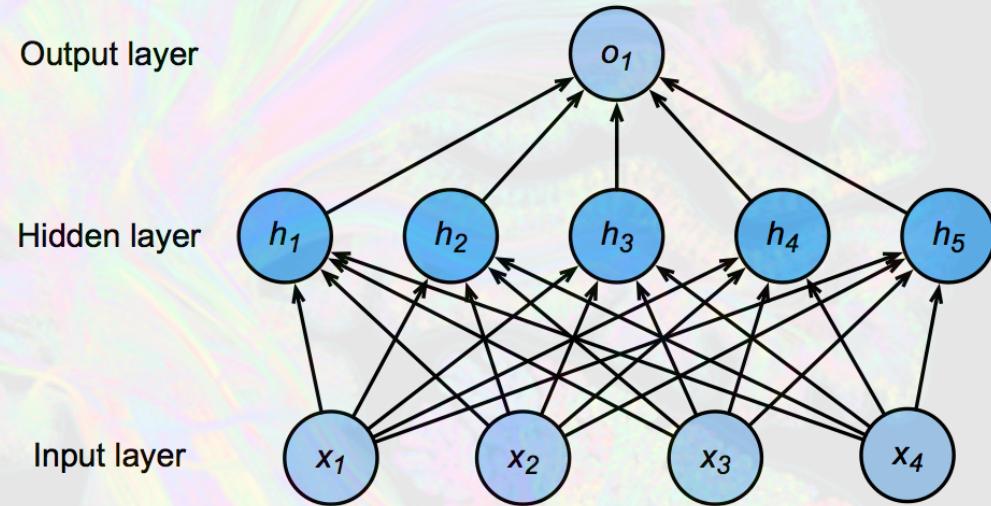
Single Hidden Layer

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



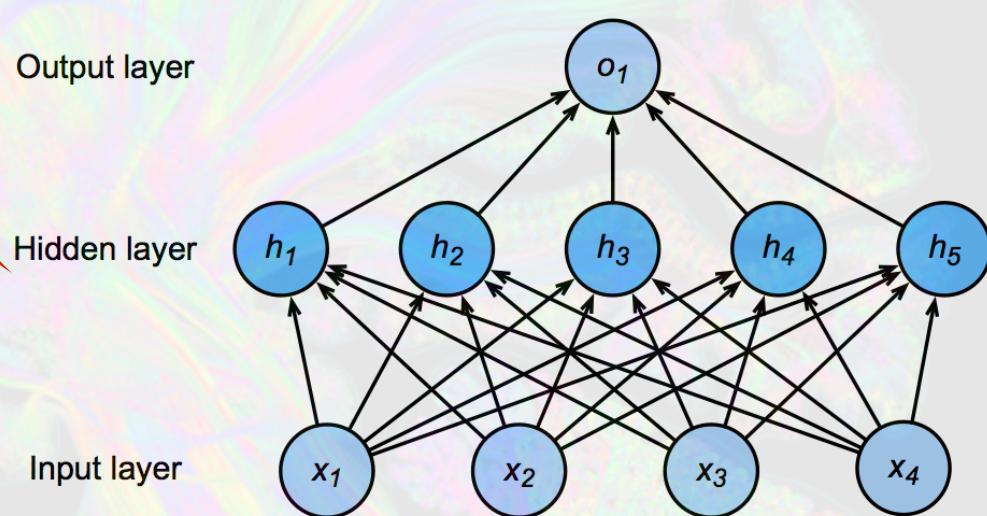
Single Hidden Layer

Why do we need an a
nonlinear activation?

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

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

$$\text{hence } o = \mathbf{w}_2^T \mathbf{W}_1 \mathbf{x} + b'$$

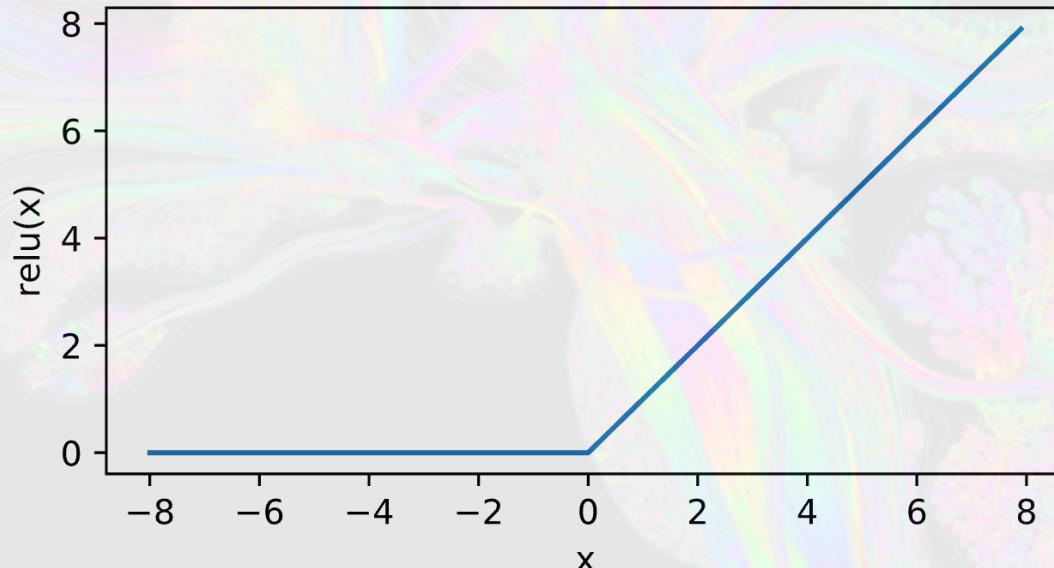


Linear ...

ReLU Activation

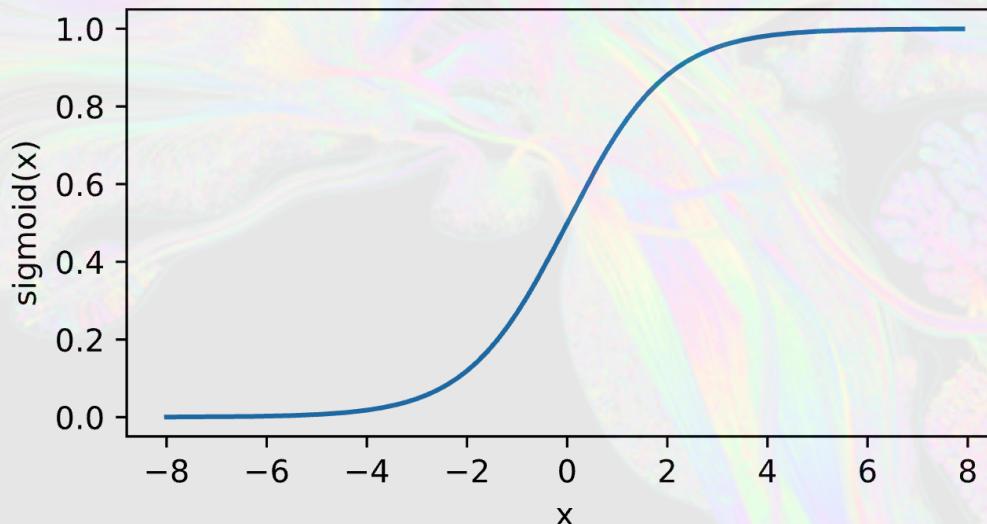
ReLU: rectified linear unit

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



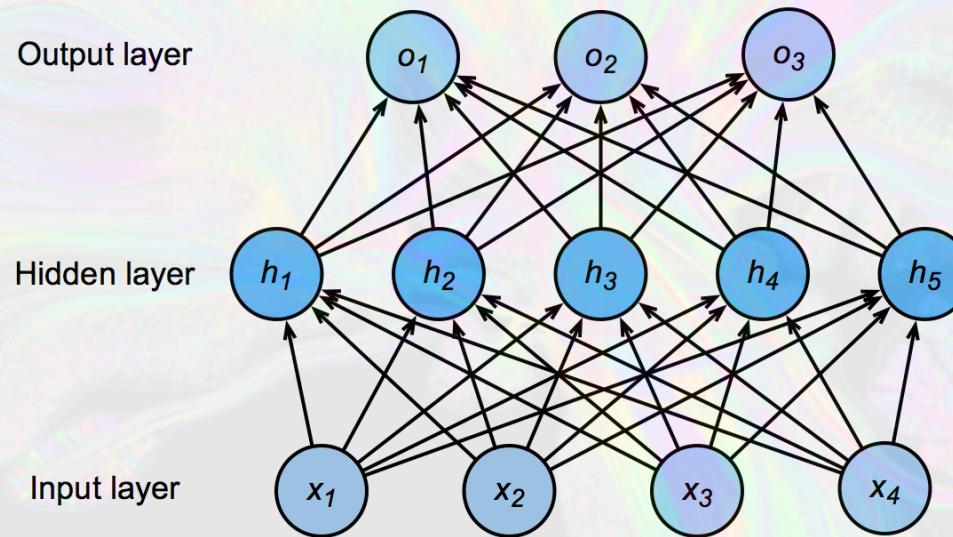
Sigmoid Activation

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

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



Multiple Hidden Layers

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

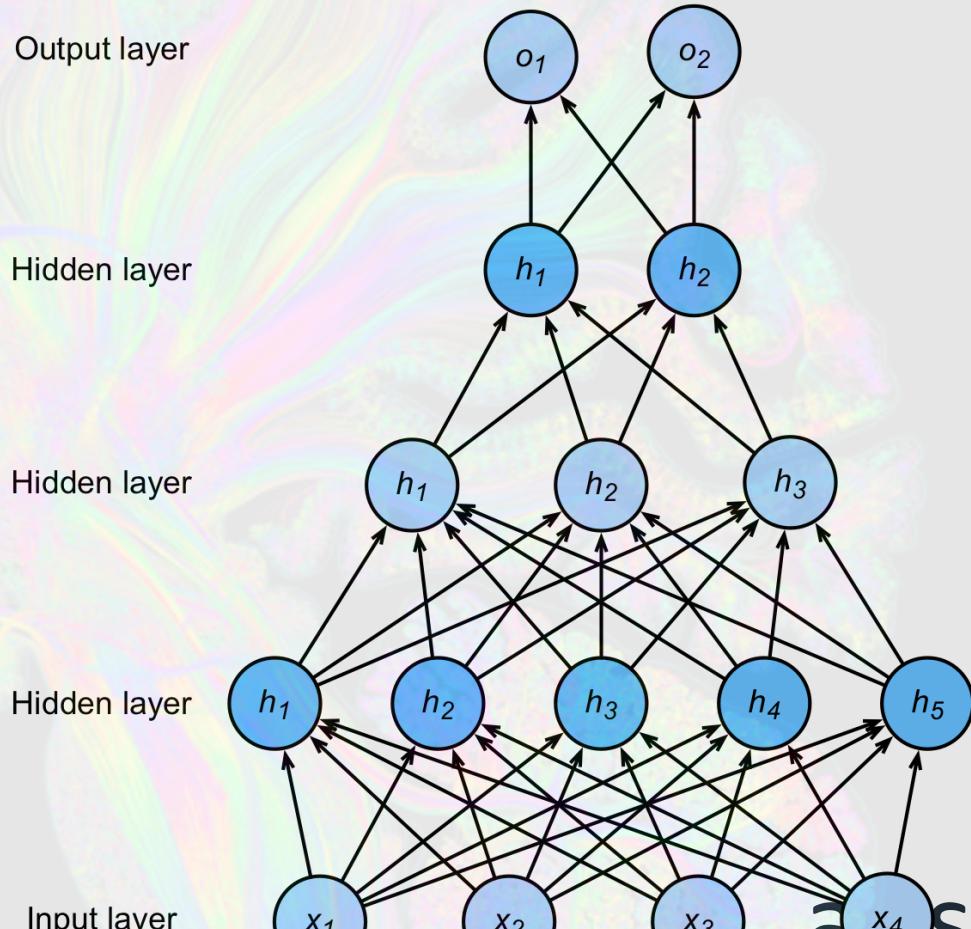
$$\mathbf{h}_2 = \sigma(\mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2)$$

$$\mathbf{h}_3 = \sigma(\mathbf{W}_3 \mathbf{h}_2 + \mathbf{b}_3)$$

$$\mathbf{o} = \mathbf{W}_4 \mathbf{h}_3 + \mathbf{b}_4$$

Hyper-parameters

- # of hidden layers
- Hidden size for each layer



MLP Notebook

Summary

- Installation
- Linear Algebra and Deep Numpy
- Autograd
- Softmax Classification
- Optimization
- Multilayer Perceptron