Week 04: Back-Prop

&

Optimization for Deep Learning

Dr. Arghya Pal
Lecturer
Monash University

Classroom Setup

Dataset

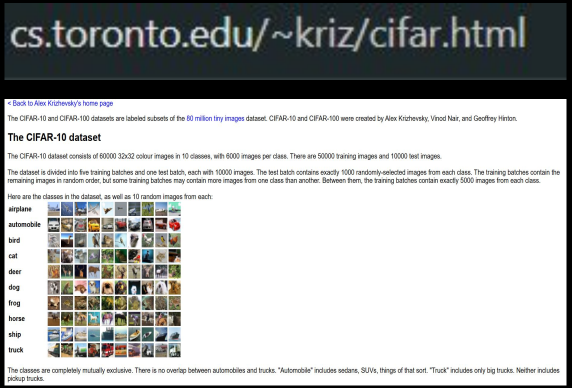
Data Calling

Data Visualization

Network Building

Optimization

Result Visualization



cs.toronto.edu/~kriz/cifar.html

< Back to Alex Krizhevsky's home page

The CIFAR-10 and CIFAR-100 datasets are labeled subsets of the 80 million tiny images dataset. CIFAR-10 and CIFAR-100 were created by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

The CIFAR-10 dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

namespace ?
allow you to use pre-loaded datasets as well as your own data

```
import torch
from torch.utils.data import Dataset, DataLoader

full_train_set =
torchvision.datasets.CIFAR10 ("./data", download=True,
transform=transform)
full_test_set = torchvision.datasets.CIFAR10 ("./data",
download=True, train=False, transform=transform)
```

Classroom Setup

Dataset

Data Calling

Data Visualization

Network Building

Optimization

Result Visualization

```
import torch
from torch.utils.data import Dataset
```

pip install datasets torchvision torch

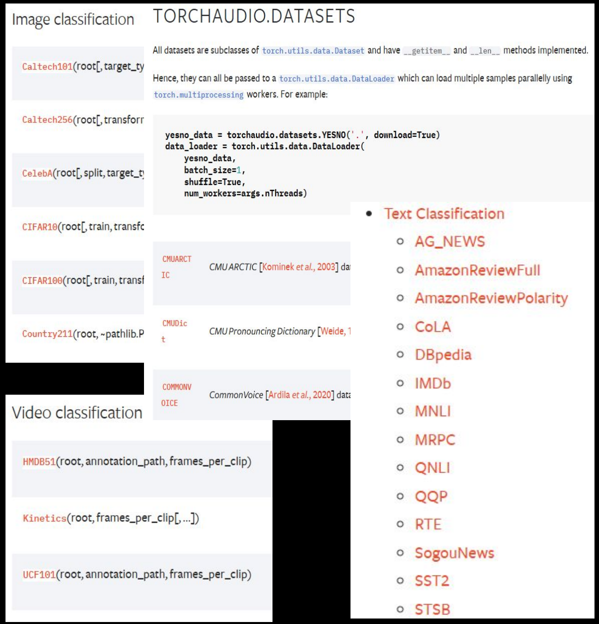


Image classification

Caltech101(root[, target_ty

Caltech256(root[, transforr

CelebA(root[, split, target_ty

CIFAR10(root[, train, transfc

CIFAR100(root[, train, transf

Country211(root, ~pathlib.P

Video classification

HMDB51(root, annotation_path, frames_per_clip)

Kinetics(root, frames_per_clip[, ...])

UCF101(root, annotation_path, frames_per_clip)

TORCHAUDIO.DATASETS

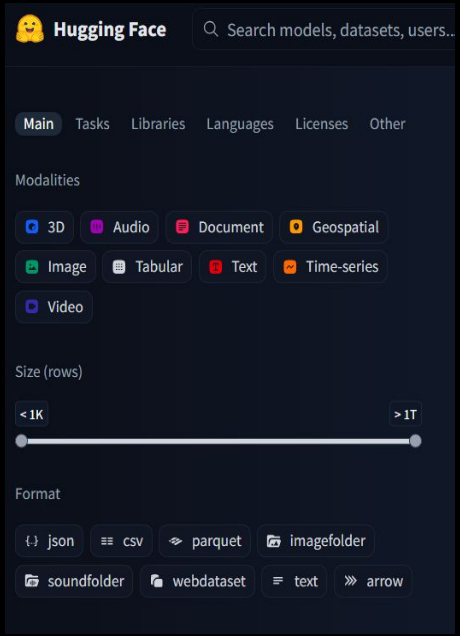
All datasets are subclasses of torch.utils.data.Dataset and have __getitem__ and __len__ methods implemented.

Hence, they can all be passed to a torch.utils.data.DataLoader which can load multiple samples parallelly using torch.multiprocessing workers. For example:

```
yesno_data = torchaudio.datasets.YESNO('.', download=True)
data_loader = torch.utils.data.DataLoader(
    yesno_data,
    batch_size=1,
    shuffle=True,
    num_workers=args.nThreads)
```

CMUARCTIC	CMU ARCTIC [Kominek et al., 2003] da
CMUDict	CMU Pronouncing Dictionary [Weide, 1
COMMONVOICE	CommonVoice [Ardila et al., 2020] data

- Text Classification
  - AG_NEWS
  - AmazonReviewFull
  - AmazonReviewPolarity
  - CoLA
  - DBpedia
  - IMDb
  - MNLI
  - MRPC
  - QNLI
  - QQP
  - RTE
  - SogouNews
  - SST2
  - STSB



Classroom Setup

Dataset

Data Calling

Data Visualization

Network Building

Optimization

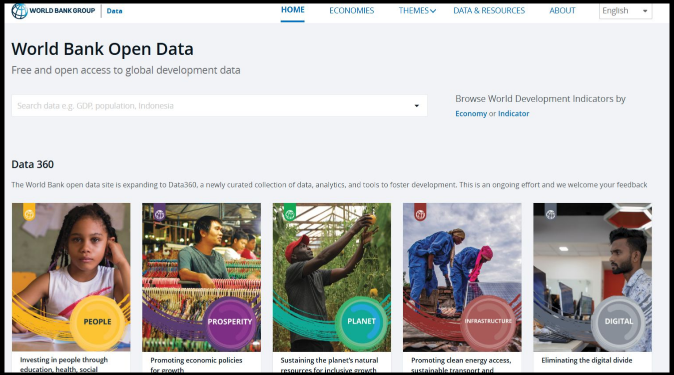
Result Visualization

```
import torch
from torch.utils.data import Dataset, DataLoader
```

```
full_train_set =
torchvision.datasets.CIFAR10("./data"
, download=True, transform=transform)
```

```
full_test_set =
torchvision.datasets.CIFAR10("./data",
download=True, train=False,
transform=transform)
```

```python
import torch
from torch.utils.data import Dataset
```



```python
class CustomDataset(Dataset):
    """Face Landmarks dataset."""

    def __init__(self, csv_file, root_dir, transform=None):
        """
        Arguments:
            csv_file (string): Path to the csv file with annotations.
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to be applied
                on a sample.
        """
        self.landmarks_frame = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform

    def __len__(self):
        return len(self.landmarks_frame)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_name = os.path.join(self.root_dir,
                                self.landmarks_frame.iloc[idx, 0])
        image = io.imread(img_name)
        landmarks = self.landmarks_frame.iloc[idx, 1:]
        landmarks = np.array([landmarks], dtype=float).reshape(-1, 2)
        sample = {'image': image, 'landmarks': landmarks}

        if self.transform:
            sample = self.transform(sample)

        return sample
your_dataset = 
CustomDataset(csv_file='data/faces/face_landmarks.csv',root_dir='data/faces/')
```

Classroom Setup

Dataset

Data Calling

Data Visualization

Network Building

Optimization

Result Visualization

```
import math
def imshow(img):
    img = img / 2 + 0.5  # unnormalize
    plt.imshow(np.transpose(img, (1, 2, 0)))
def visualize_data(images, categories,
images_per_row = 8):
    class_names = ['airplane', 'automobile',
'bird', 'cat', 'deer',
'dog', 'frog', 'horse', 'ship', 'truck']
    n_images = len(images)
    n_rows = math.ceil (float(n_images)/images_per_row)
    fig = plt.figure(figsize=(1.5*images_per_row, 1.5*n_rows))
    fig.patch.set_facecolor('white')
    for i in range(n_images):
        plt.subplot(n_rows, images_per_row, i+1)
        plt.xticks([])
        plt.yticks([])
        imshow(images[i])
            class_index = categories [i]
            plt.xlabel (class_names[class_index])
    plt.show()
```

Classroom Setup

Dataset

Data Calling

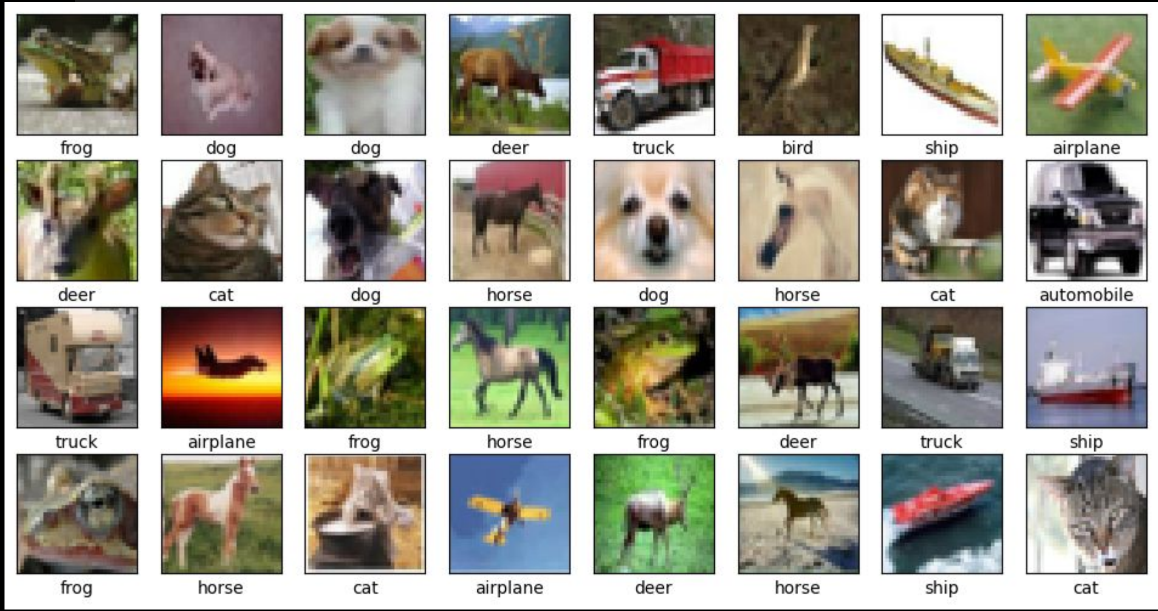
Data Visualization

Network Building

Optimization

Result Visualization

```
import math
def imshow(img):
    img = img / 2 + 0.5   # unnormalize
    plt.imshow(np.transpose(img, (1, 2, 0)))
```



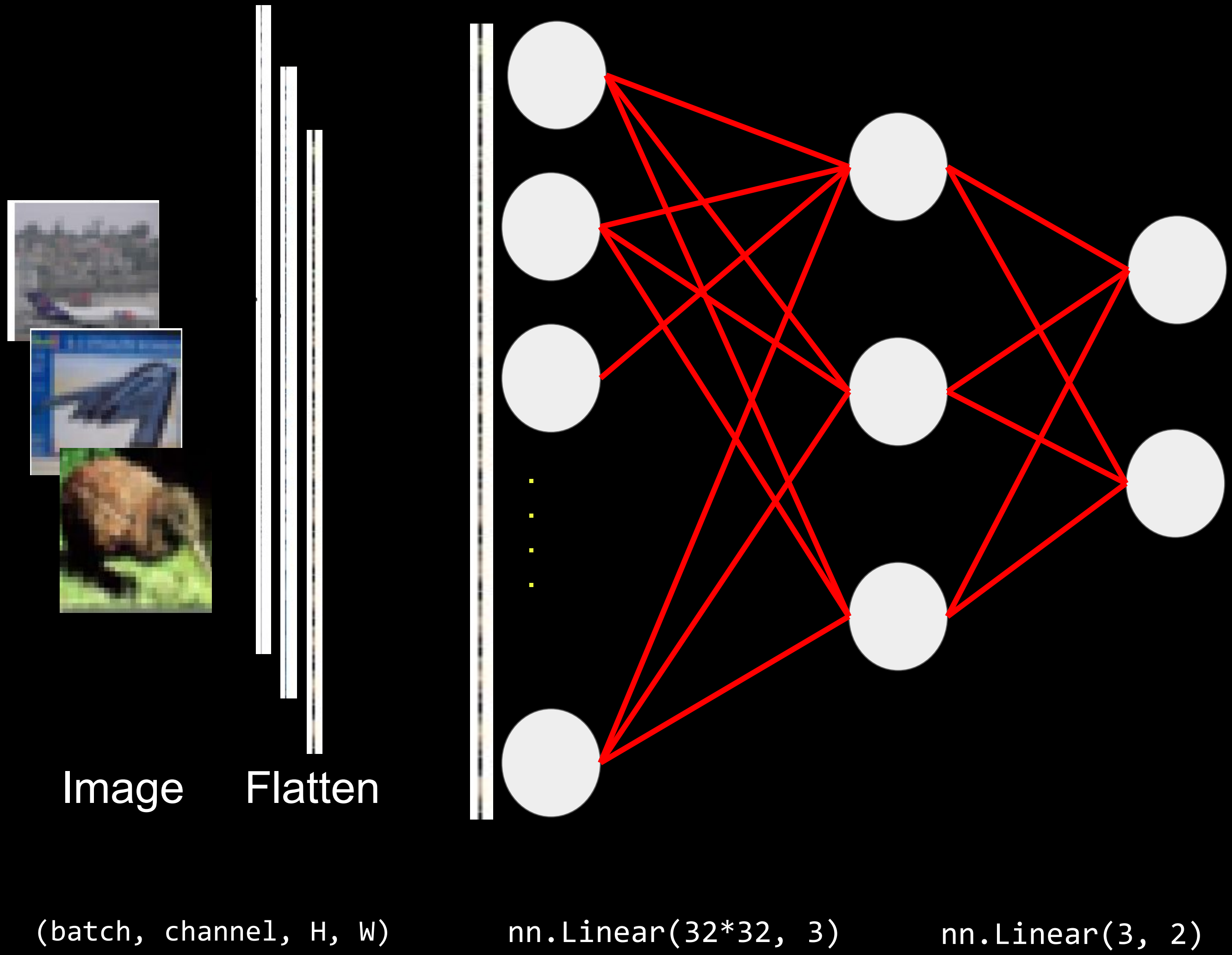

```
        class_index = categories[i]
        plt.xlabel(class_names[class_index])
    plt.show()
```

Classroom Setup

Dataset
Data Calling
Data Visualization

Network Building

Optimization
Result Visualization

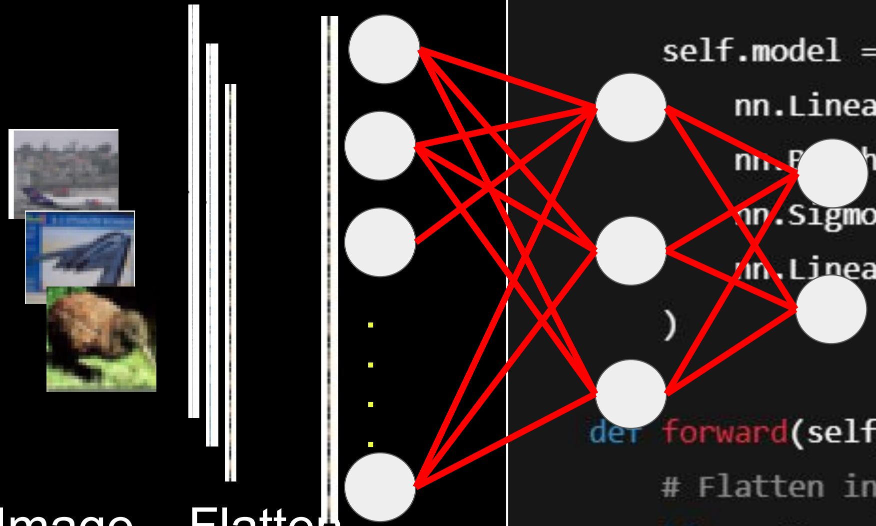


Classroom Setup

Dataset
Data Calling
Data Visualization

Network Building

Optimization
Result Visualization




```python
import torch
import torch.nn as nn

class SimpleMLP(nn.Module):
    def __init__(self):
        super(SimpleMLP, self).__init__()

        self.model = nn.Sequential(
            nn.Linear(32 * 32, 3),
            nn.BatchNorm1d(3),
            nn.Sigmoid(),
            nn.Linear(3, 2)
        )

    def forward(self, x):
        # Flatten input if it's an image
        if x.ndim > 2:
            x = x.view(x.size(0), -1)
        return self.model(x)
```

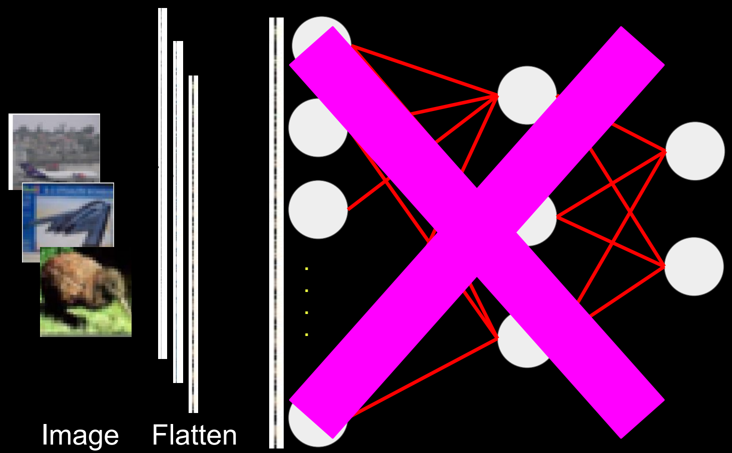
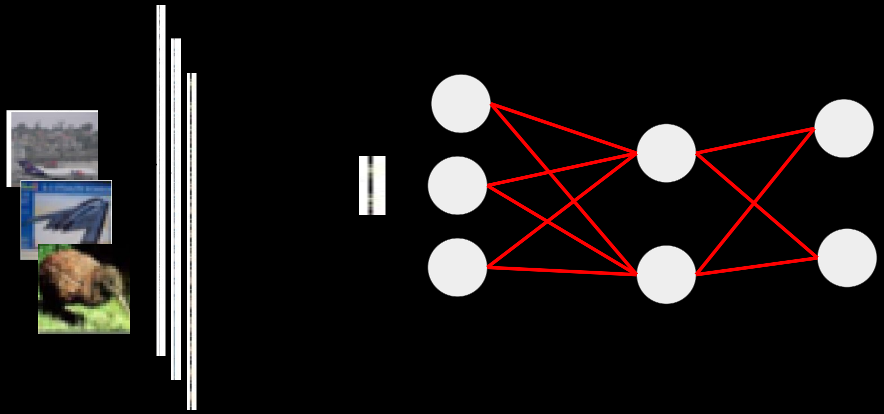
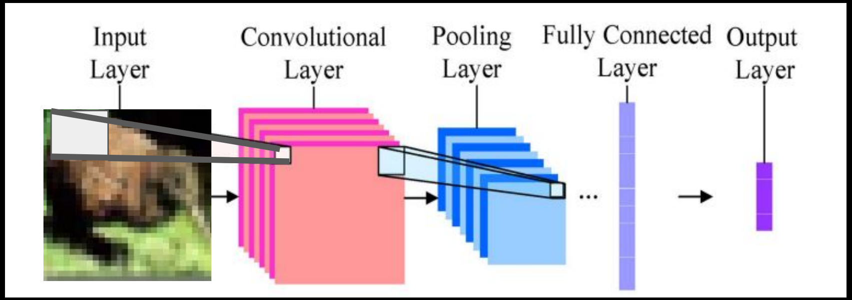


Image
Flatten


Yoshua Bengio
Geoffrey Hinton
Yann LeCun



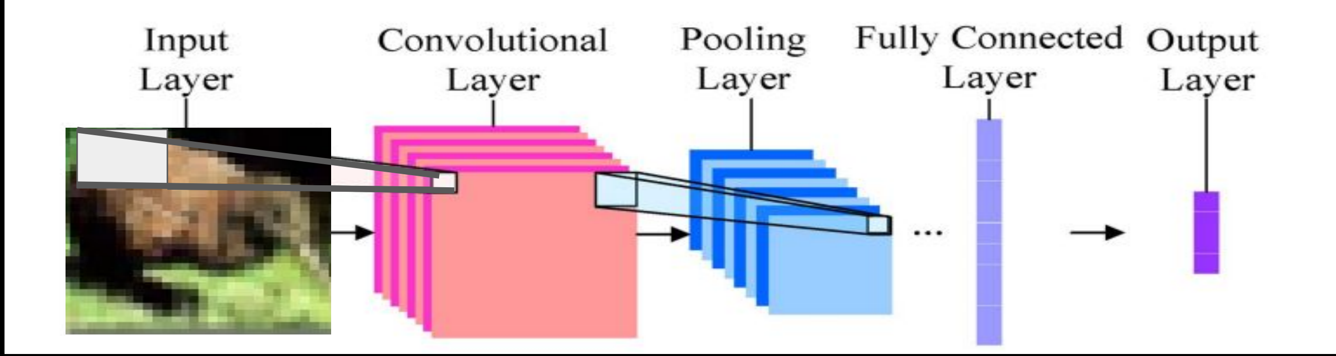

Input Layer
Convolutional Layer
Pooling Layer
Fully Connected Layer
Output Layer

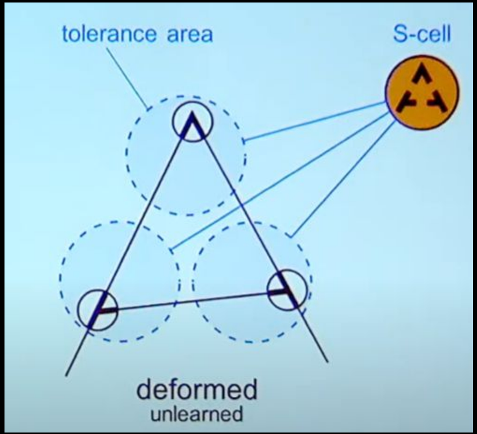
Classroom Setup

Dataset
Data Calling
Data Visualization

Network Building

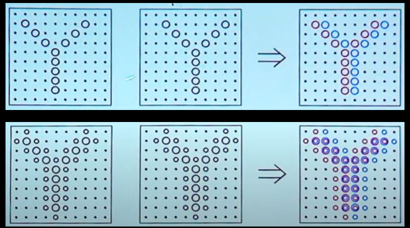
Optimization
Result Visualization



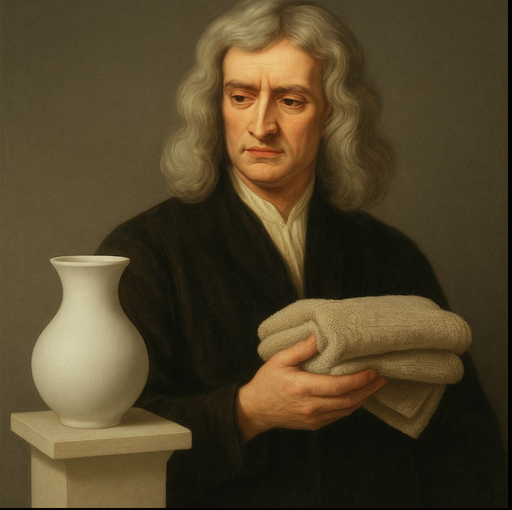


Complex Cell

A. Shift → Convolution



B. Size → Pooling



Classroom Setup

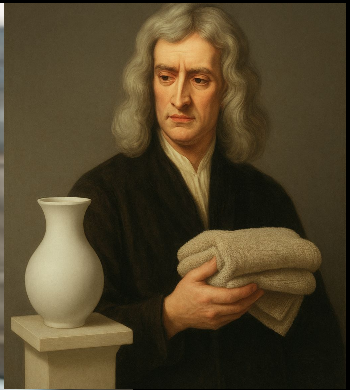
Dataset
Data Calling
Data Visualization

Network Building

Optimization

Result Visualization

- Revision of calculus
- Gradient descent and stochastic gradient descent
- Backpropagation in feed-forward neural networks
- Optimizers for deep learning.

T = 0	T = 1	T = 2	T = 3	T = 4
10	10	10	10	10

Speed Vs. Velocity Vs Acceleration

Speed: 10 m/s   Average Speed: (10+10+10+10+10)/5

Velocity: 10 m/s towards your right hand side (Speed + Direction)

Acceleration:

(T = 1)	–	(T = 0)	=	10	-	10	=	0
(T = 2)	–	(T = 1)	=	10	-	10	=	0
(T = 3)	–	(T = 2)	=	10	-	10	=	0
(T = 4)	–	(T = 4)	=	10	-	10	=	0

(Observed Quantity) / (Controlled Quantity)

- Revision of calculus
- Gradient descent and stochastic gradient descent
- Backpropagation in feed-forward neural networks
- Optimizers for deep learning.

T = 0	T = 1	T = 2	T = 3	T = 4
10	15	10	20	10

Change Vs. Rate of change

Average Speed: (10+15+10+20+10) / 5 = 13 m/s

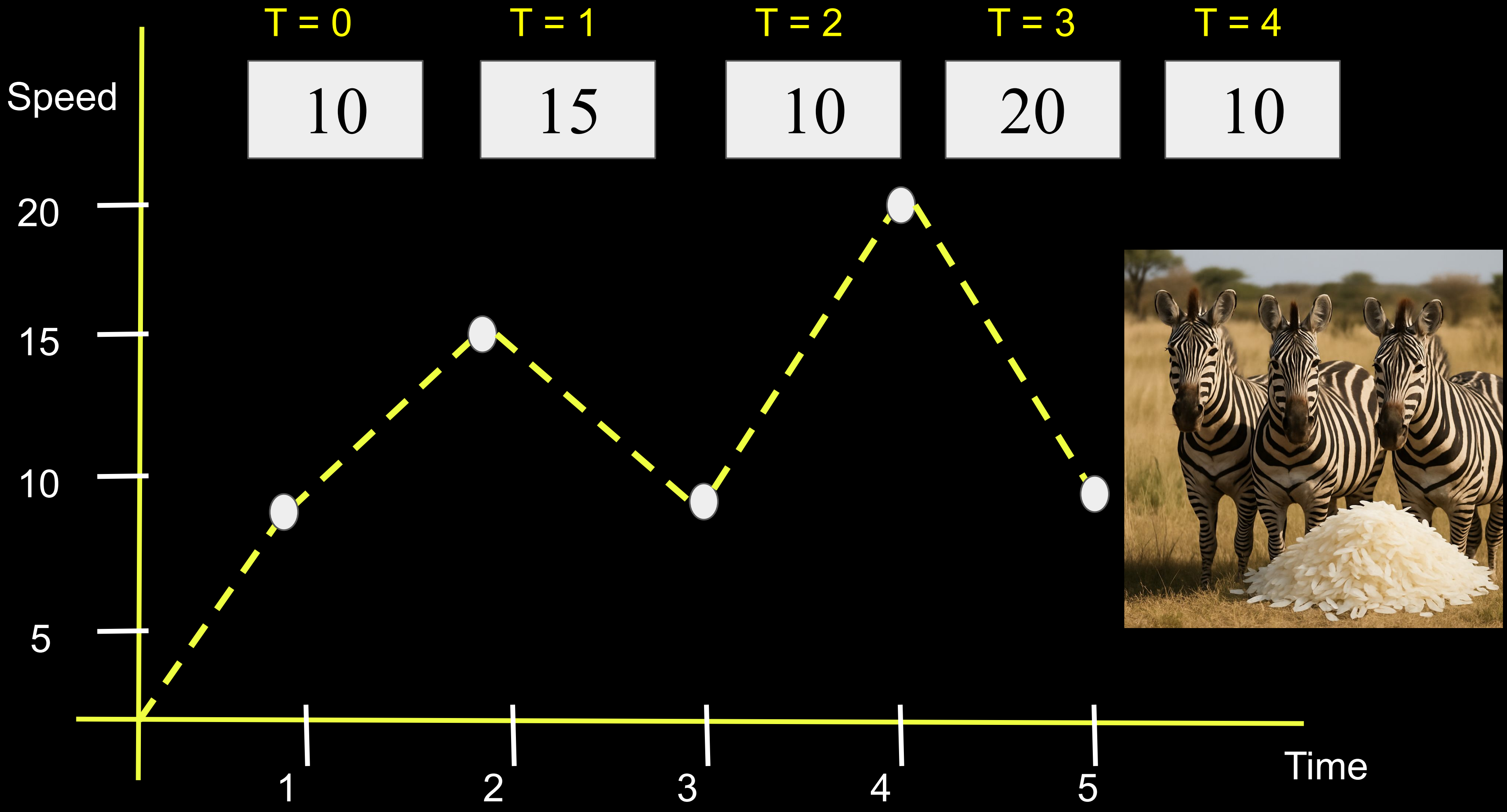
Velocity: 13 m/s towards my right hand

Acceleration

(T = 1)	–	(T = 0)	=	15	-	10	=	5
(T = 2)	–	(T = 1)	=	10	-	15	=	- 5
(T = 3)	–	(T = 2)	=	20	-	10	=	10
(T = 4)	–	(T = 4)	=	10	-	20	=	- 10

(Observed Quantity) / (Controlled Quantity)

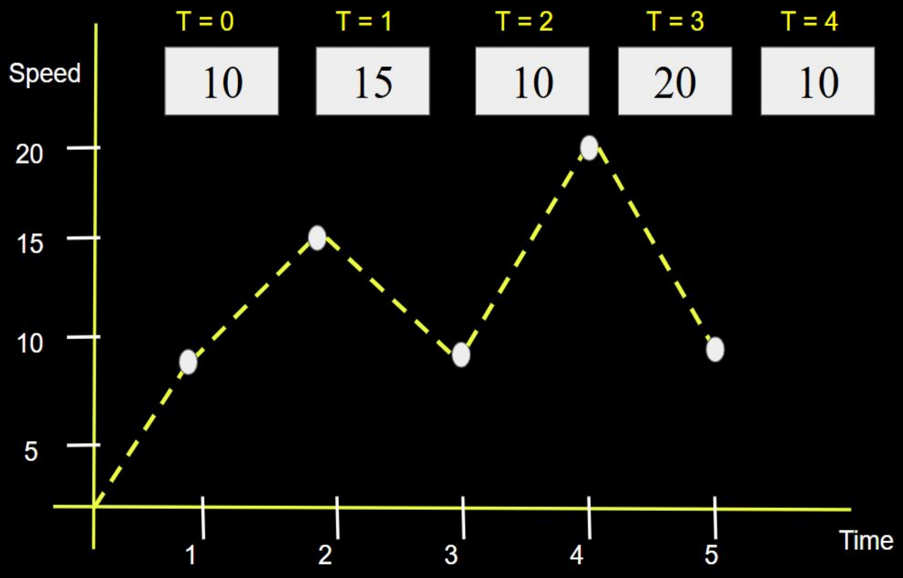
- Revision of calculus
- Gradient descent and stochastic gradient descent
- Backpropagatio n in feed-forward neural networks
- Optimizers for deep learning.



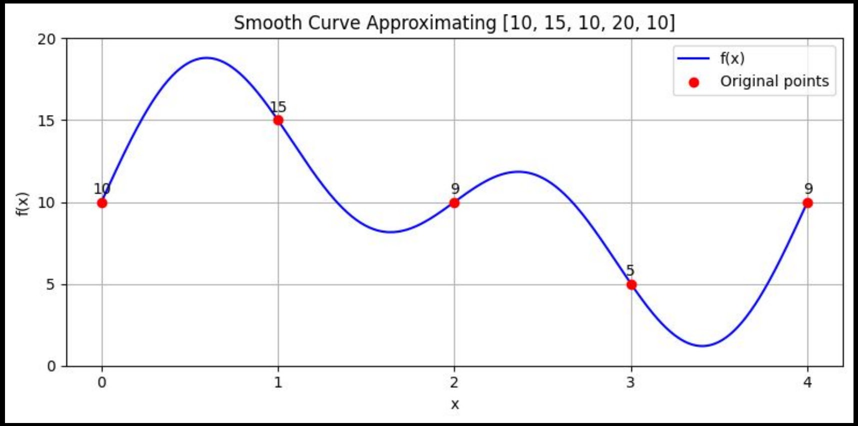
T = 0
T = 1
T = 2
T = 3
T = 4
10
15
10
20
10
Speed
20
15
10
5
1
2
3
4
5
Time

Numeric Values
Numerical Methods

Functional Values
Functional Methods



f(x)=10+5·sin(πx/2)+5·sin(π



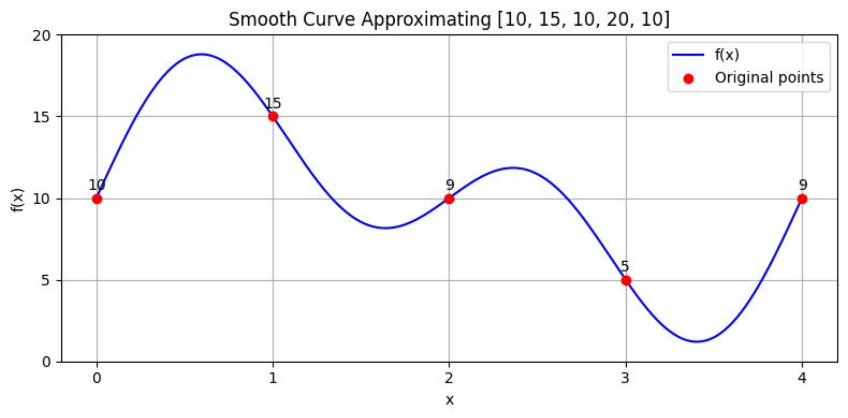
$$\frac{f(t + 1) - f(t)}{(t+1) - t}$$

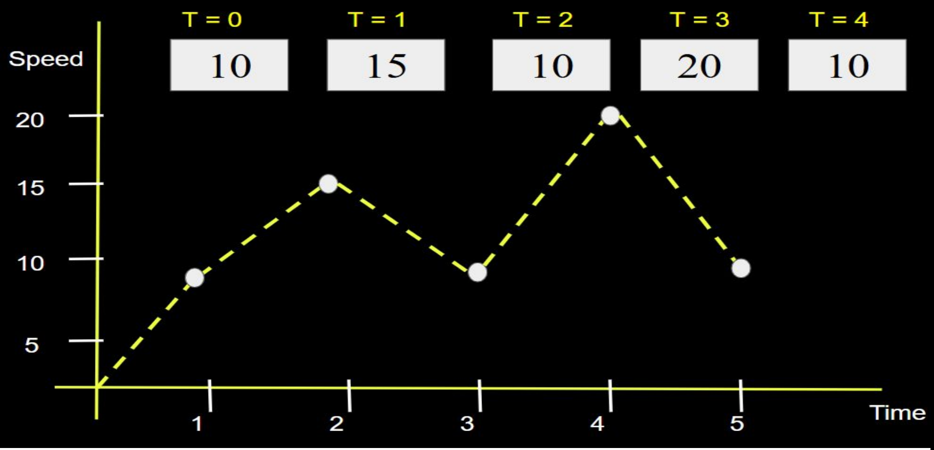
=

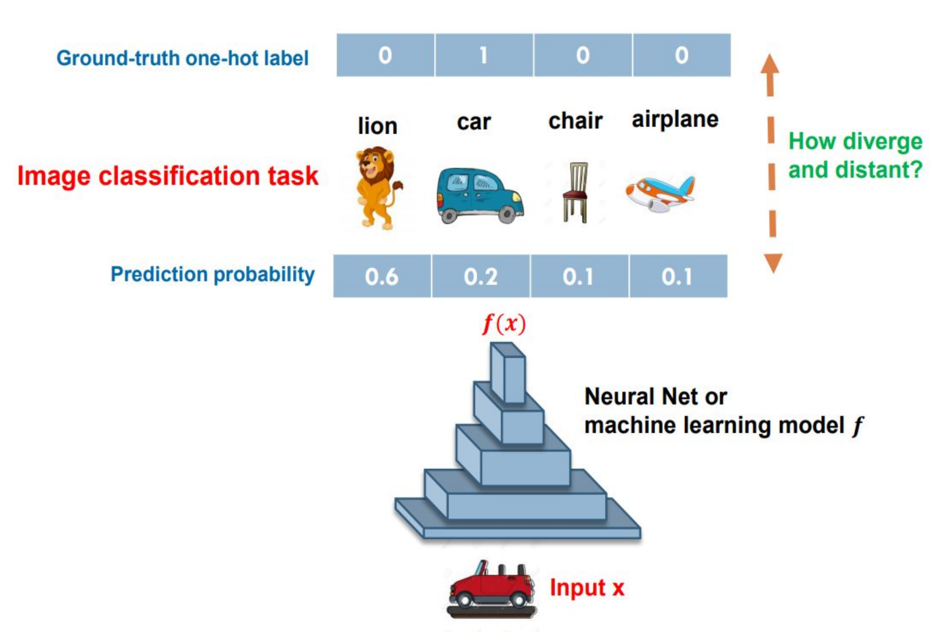
Rate of Change

Functional Values
Functional Methods

f(t)=10+5·sin(πt/2)+5·sin(π







(Observed Quantity) / (Controlled Quantity)

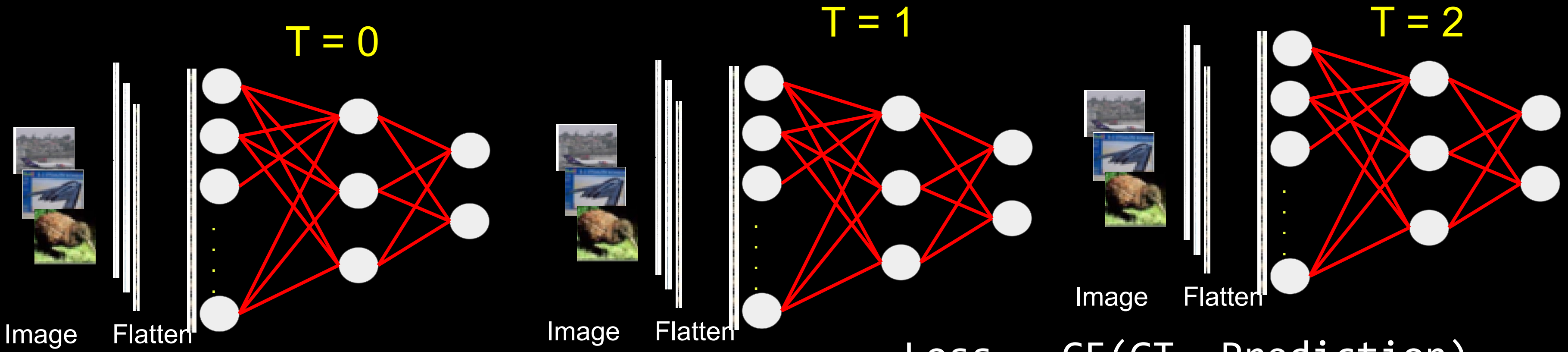
Minimize Loss / ??

Network Architecture

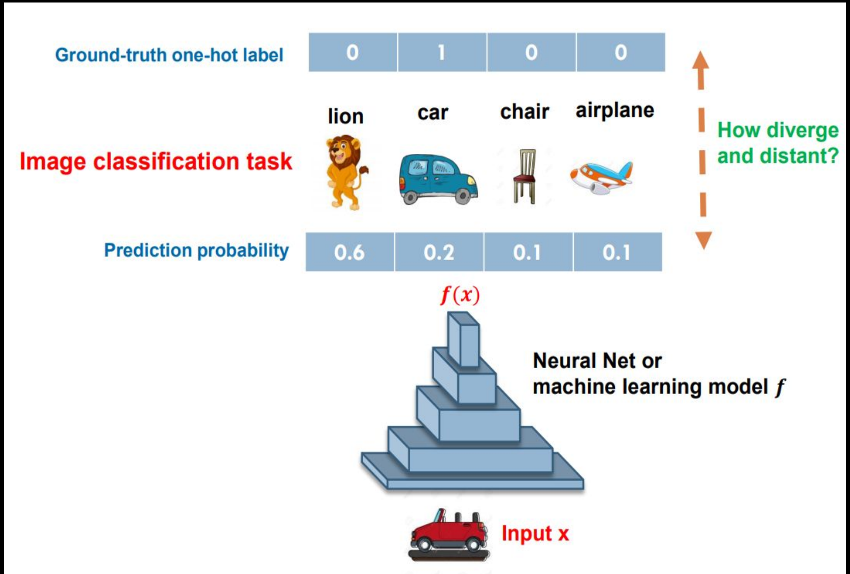
Dataset: Batch Size

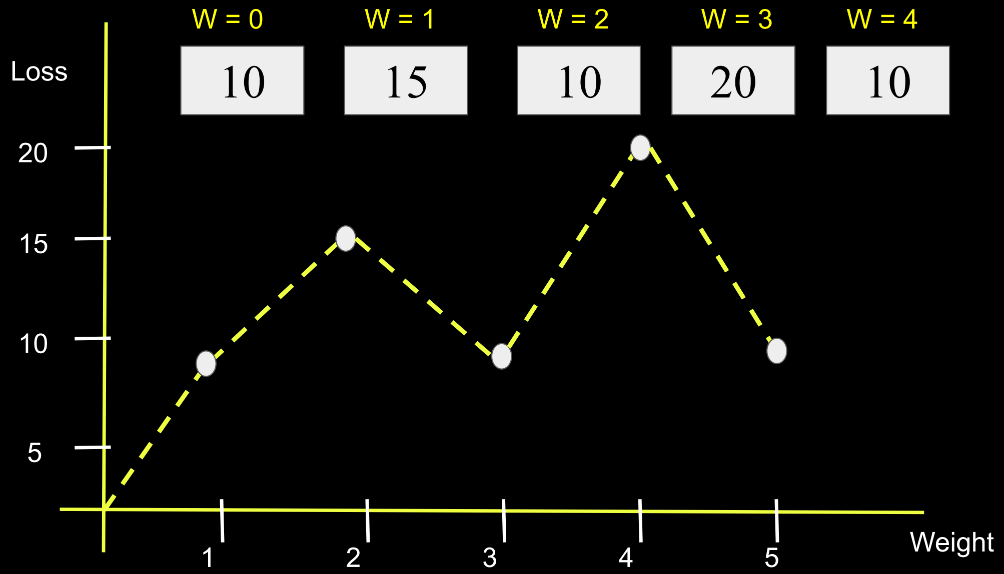
Anger

<u>Weight</u>

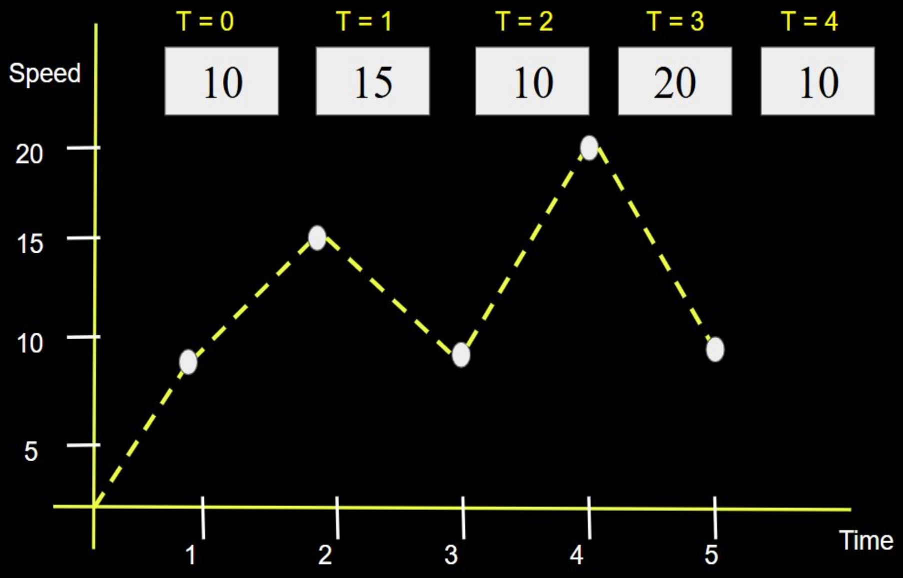


Loss = CE(GT, Prediction)



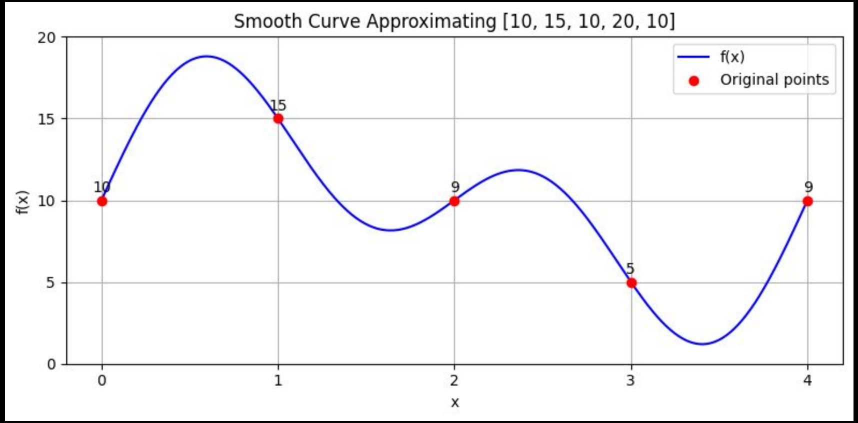


Numeric Values
Numerical Methods



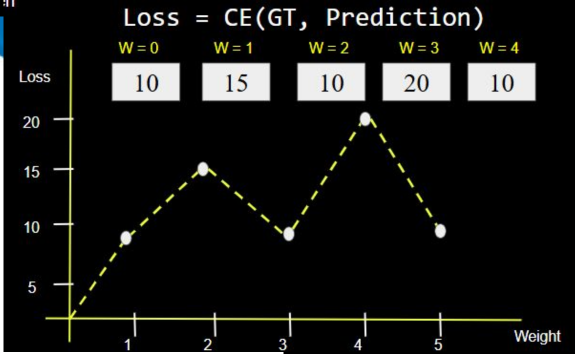
Loss Function

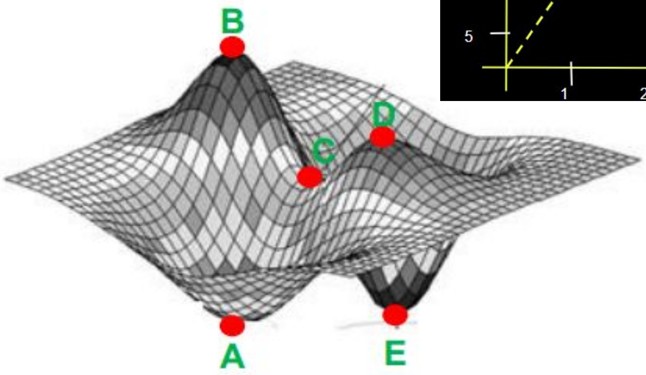
f(L)=10+5·sin(πL/2)+5·sin(π
L)

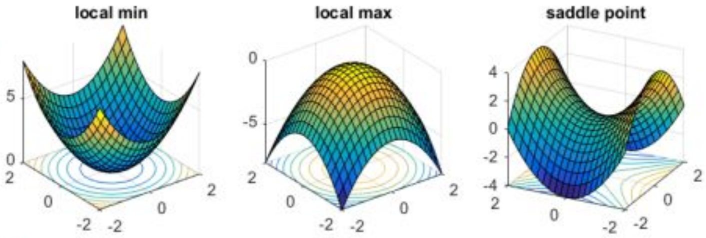


Local minima-maxima and saddle point

- Given an **objective function** $J(\theta)$ with $\theta = [\theta_1, \theta_2, \dots, \theta_P]$
  - θ is said to be a **critical point** if $\nabla J(\theta) = \mathbf{0}$ (vector $\mathbf{0}$)
- Let us denote the **set of eigenvalues** of Hessian matrix $\nabla^2 J(\theta) = H(\theta)$ by
  - $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_P$
- **Local minima**
  - $\nabla J(\theta) = \mathbf{0}$ and $\nabla^2 J(\theta) = H(\theta) \succ 0$ (positive semi-definite matrix)
  - $0 \le \lambda_1 \le \lambda_2 \le \cdots \le \lambda_P$
- **Local maxima**
  - $\nabla J(\theta) = \mathbf{0}$ and $\nabla^2 J(\theta) = H(\theta) \prec 0$ (negative semi-definite matrix)
  - $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_P \le 0$
- **Saddle point**
  - $\nabla J(\theta) = \mathbf{0}$ and $\nabla^2 J(\theta) = H(\theta) \prec 0$ (indefinite matrix)
  - $\lambda_1 \le \lambda_2 \le \cdots < 0 < \cdots \le \lambda_P$







Backprop By Hand

Goto slide 7,
Week 4

A small detour to calculus

- Calculus = **mathematics of change** (very important for deep learning)
- Properties of derivative:
  - $f'(x) = \nabla f(x) = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$
  - $(uv)' = u'v + uv'$
  - $\left(\frac{u}{v}\right)' = \frac{u'v - uv'}{v^2}$
  - $(e^u)' = u'e^u$
  - $(\log u)' = \frac{u'}{u}$
- Multi-variate function $f\colon \mathbb{R}^n \to \mathbb{R}$ with $y = f(x) = f(x_1, \dots, x_n)$.
  - Gradient/derivative: $\frac{\partial f}{\partial x}(a) = \nabla_x f(a) = [\nabla_{x_1} f(a), \nabla_{x_2} f(a), \dots, \nabla_{x_n} f(a)]$.
- Chain rule 🔗:
  - $\frac{\partial u}{\partial x} = \frac{\partial u}{\partial v} \times \frac{\partial v}{\partial x}$

Example

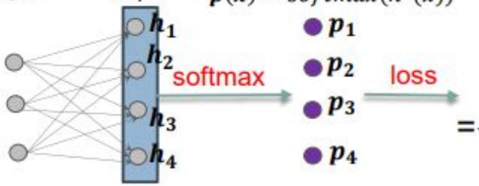
- $y = f(x) = f(x_1, x_2, x_3) = (x_1^2 + x_2^2, x_2^2 + x_3^2 x_2)$
  - $f\colon \mathbb{R}^3 \to \mathbb{R}^2$
  - $f_1(x) = f_1(x_1, x_2, x_3) = x_1^2 + x_2^2$
  - $f_2(x) = f_2(x_1, x_2, x_3) = x_2^2 + x_3^2 x_2$
  - $\frac{\partial y}{\partial x} = \nabla f \in \mathbb{R}^{2\times 3}$

$$\frac{\partial y}{\partial x} = \nabla_x f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 2x_1 & 2x_2 & 0 \\ 0 & 2x_2 + x_3^2 & 2x_2 x_3 \end{bmatrix}$$

Example

Output layer

$x = [x_1\ x_2\ x_2] \in \mathbb{R}^{1\times 3}$
$y = 2$



$$l = loss = -\log p_2$$
$$= -\log \frac{e^{h_2}}{e^{h_1}+e^{h_2}+e^{h_3}+e^{h_4}}$$

$$p_1 = \frac{e^{h_1}}{e^{h_1}+e^{h_2}+e^{h_3}+e^{h_4}}$$
$$p_2 = \frac{e^{h_2}}{e^{h_1}+e^{h_2}+e^{h_3}+e^{h_4}}$$
$$p_3 = \frac{e^{h_3}}{e^{h_1}+e^{h_2}+e^{h_3}+e^{h_4}}$$
$$p_4 = \frac{e^{h_4}}{e^{h_1}+e^{h_2}+e^{h_3}+e^{h_4}}$$

Compute $\frac{\partial l}{\partial \boldsymbol{h}}$?

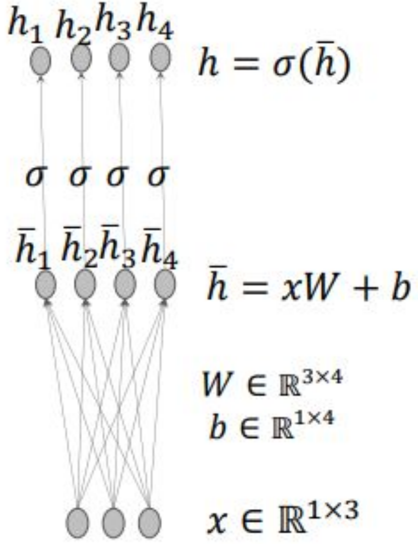
- $l = -\log \frac{e^{h_2}}{e^{h_1}+e^{h_2}+e^{h_3}+e^{h_4}} = \log(\overbrace{e^{h_1}+e^{h_2}+e^{h_3}+e^{h_4}}^{u}) - h_2$
- $\frac{\partial l}{\partial h_1} = \frac{\nabla_{h_1} u}{u} = \frac{e^{h_1}}{e^{h_1}+e^{h_2}+e^{h_3}+e^{h_4}} = p_1$
- $\frac{\partial l}{\partial h_2} = \frac{\nabla_{h_2} u}{u} - 1 = \frac{e^{h_2}}{e^{h_1}+e^{h_2}+e^{h_3}+e^{h_4}} - 1 = p_2 - 1$
- $\frac{\partial l}{\partial h_3} = \frac{\nabla_{h_3} u}{u} = \frac{e^{h_3}}{e^{h_1}+e^{h_2}+e^{h_3}+e^{h_4}} = p_3$
- $\frac{\partial l}{\partial h_4} = \frac{\nabla_{h_4} u}{u} = \frac{e^{h_4}}{e^{h_1}+e^{h_2}+e^{h_3}+e^{h_4}} = p_4$
- $\frac{\partial l}{\partial \boldsymbol{h}} = [p_1, p_2 - 1, p_3, p_4] = [p_1, p_2, p_3, p_4] - [0,1,0,0] = \boldsymbol{p} - \boldsymbol{1}_2 = \boldsymbol{p} - \boldsymbol{1}_y$

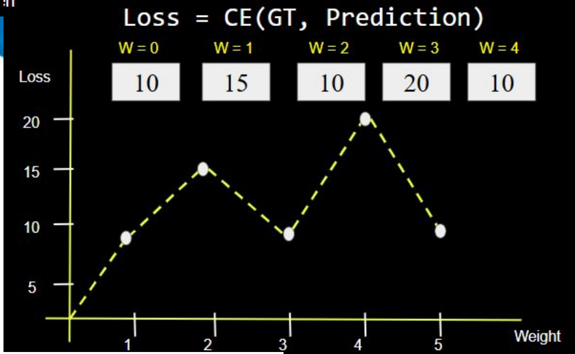
Example
Intermediate layer

- $\bar{h} = xW + b$ and $h = \sigma(\bar{h})$
  - $h = \sigma(xW + b)$
  - σ is the **activation function**
- $\frac{\partial h}{\partial x} = \frac{\partial h}{\partial \bar{h}} \times \frac{\partial \bar{h}}{\partial x} = diag\left(\sigma'\left(\bar{h}\right)\right) W^T \in \mathbb{R}^{4\times 3}$

- $$\frac{\partial h}{\partial \bar{h}} = \begin{bmatrix} \frac{\partial h_1}{\partial \bar{h}_1} & \frac{\partial h_1}{\partial \bar{h}_2} & \frac{\partial h_1}{\partial \bar{h}_3} & \frac{\partial h_1}{\partial \bar{h}_4} \\ \frac{\partial h_2}{\partial \bar{h}_1} & \frac{\partial h_2}{\partial \bar{h}_2} & \frac{\partial h_2}{\partial \bar{h}_3} & \frac{\partial h_2}{\partial \bar{h}_4} \\ \frac{\partial h_3}{\partial \bar{h}_1} & \frac{\partial h_3}{\partial \bar{h}_2} & \frac{\partial h_3}{\partial \bar{h}_3} & \frac{\partial h_3}{\partial \bar{h}_4} \\ \frac{\partial h_4}{\partial \bar{h}_1} & \frac{\partial h_4}{\partial \bar{h}_2} & \frac{\partial h_4}{\partial \bar{h}_3} & \frac{\partial h_4}{\partial \bar{h}_4} \end{bmatrix} = \begin{bmatrix} \sigma'(\bar{h}_1) & 0 & 0 & 0 \\ 0 & \sigma'(\bar{h}_2) & 0 & 0 \\ 0 & 0 & \sigma'(\bar{h}_3) & 0 \\ 0 & 0 & 0 & \sigma'(\bar{h}_4) \end{bmatrix} = diag(\sigma'(\bar{h}))$$

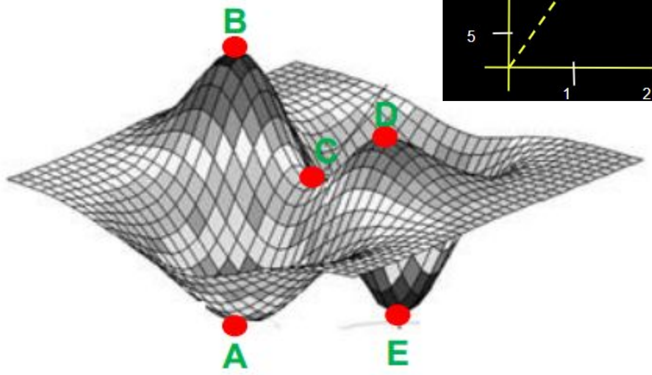
- $\frac{\partial \bar{h}}{\partial x} = W^T$

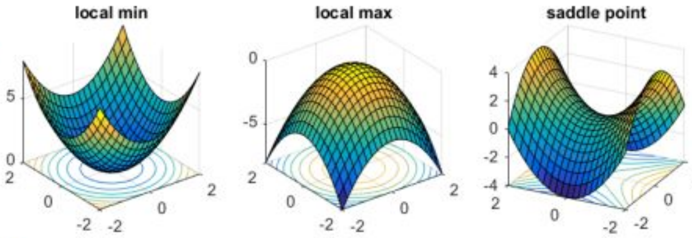




Local minima-maxima and saddle point

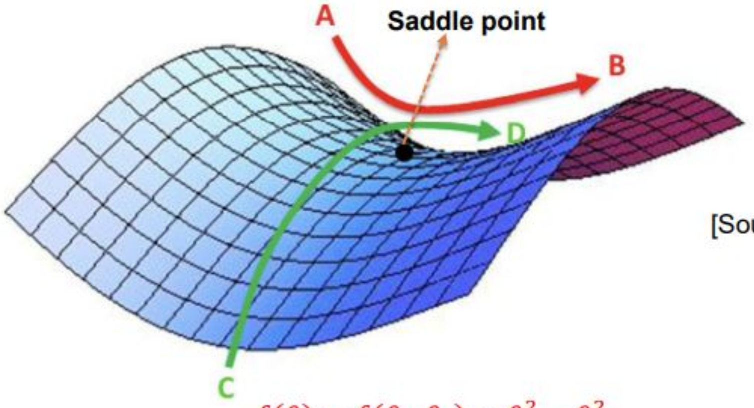
- Given an **objective function** $J(\theta)$ with $\theta = [\theta_1, \theta_2, \ldots, \theta_P]$
  - θ is said to be a **critical point** if $\nabla J(\theta) = \mathbf{0}$ (vector $\mathbf{0}$)
- Let us denote the **set of eigenvalues** of Hessian matrix $\nabla^2 J(\theta) = H(\theta)$ by
  - $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_P$
- **Local minima**
  - $\nabla J(\theta) = \mathbf{0}$ and $\nabla^2 J(\theta) = H(\theta) \succ 0$ (positive semi-definite matrix)
  - $0 \leq \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_P$
- **Local maxima**
  - $\nabla J(\theta) = \mathbf{0}$ and $\nabla^2 J(\theta) = H(\theta) \prec 0$ (negative semi-definite matrix)
  - $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_P \leq 0$
- **Saddle point**
  - $\nabla J(\theta) = \mathbf{0}$ and $\nabla^2 J(\theta) = H(\theta) \prec 0$ (indefinite matrix)
  - $\lambda_1 \leq \lambda_2 \leq \cdots < 0 < \cdots \leq \lambda_P$





MONASH University

More on saddle point



[Source: Internet]

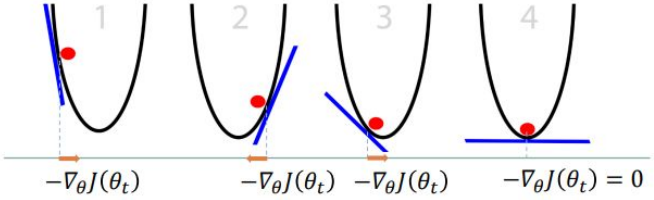
$$f(\theta) = f(\theta_1, \theta_2) = \theta_1^2 - \theta_2^2$$

Gradient $g = \begin{bmatrix} \frac{\partial f}{\partial \theta_1} \\ \frac{\partial f}{\partial \theta_2} \end{bmatrix} = \begin{bmatrix} 2\theta_1 \\ -2\theta_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ → a **critical point** $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$.

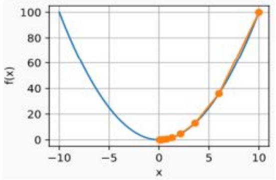
Hessian matrix is $H = \begin{bmatrix} \frac{\partial^2 f}{\partial \theta_1^2} & \frac{\partial^2 f}{\partial \theta_1 \partial \theta_2} \\ \frac{\partial^2 f}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 f}{\partial \theta_2^2} \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix}$

Two eigenvalues $\lambda_1 = -2 < 0 < 2 = \lambda_2$ → $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$ is a **saddle point**.

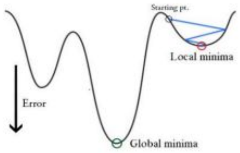
Gradient descend



- We need to solve
  - $\min_\theta J(\theta)$
- Follow to **the opposite side** of the current gradient
  - $\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta_t)$ where $\boldsymbol{\eta} > \mathbf{0}$ is the **learning rate**.
- Guarantee to converge to a **global minima** if $J(.)$ is **convex**.
- Get stuck in a **local minima** or **saddle points** if $J(.)$ is non-convex.

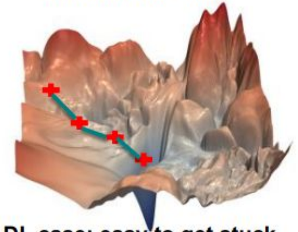


Convex case



(Source: www.cs.ubc.ca)

Non-convex case



DL case: easy to get stuck in saddle points

Gradient descend

Algorithm

- **Input**: objective function $\boldsymbol{J}(\boldsymbol{\theta})$
- **Output**: optimal solution $\boldsymbol{\theta}^*$

1. Initialize parameters θ_0 randomly $\sim N(0, \sigma^2)$.
2. for t=1 to T
3. Compute gradients $\nabla_\theta J(\theta_t) = \frac{\partial J}{\partial \theta}(\theta_t)$
4. Update $\theta_{t+1} = \theta_t - \eta_t \nabla_\theta J(\theta_t)$
5. Return $\theta^* = \theta_{T+1}$