The Neural Network Zoo

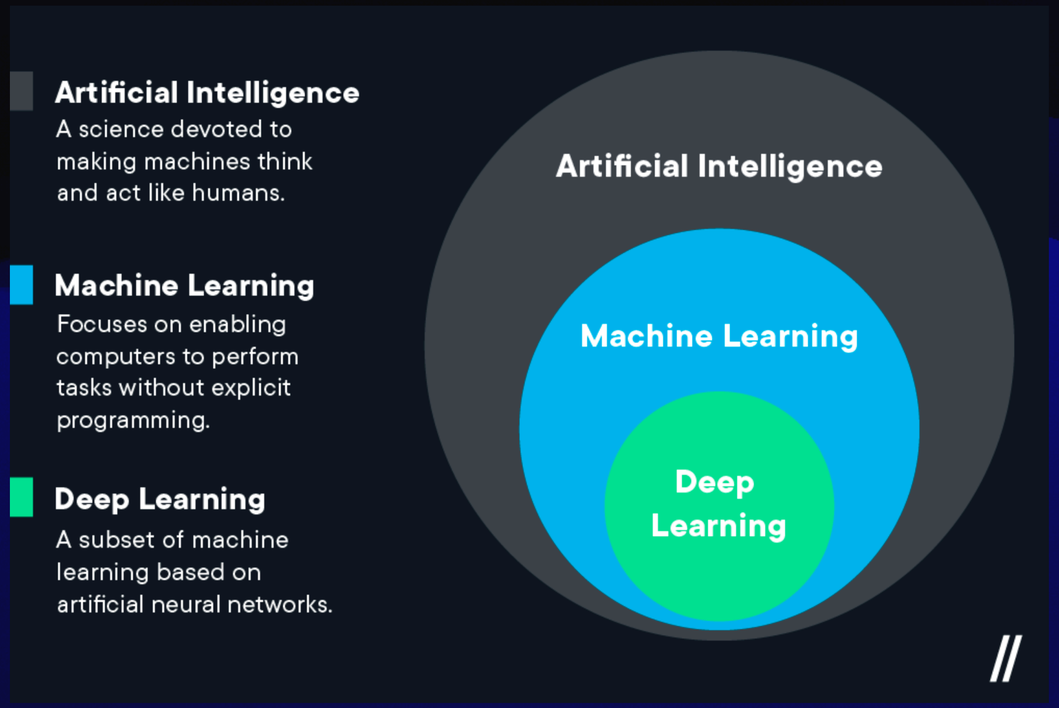
A Survey of Neural Network Architectures
PICScIE/PRC Workshop Series

Gage DeZoort | 3/25/24

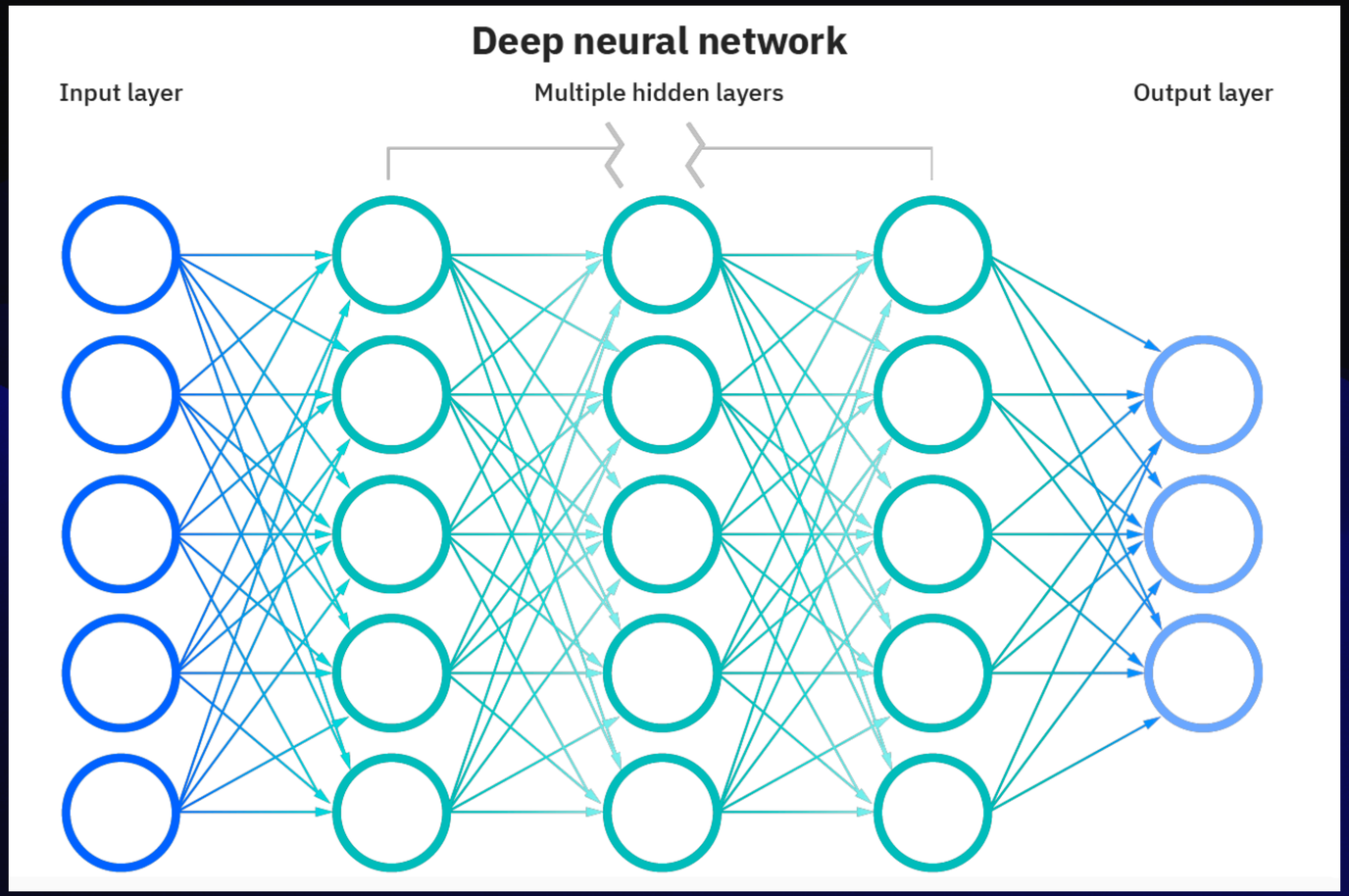
1. Intro to DL Concepts
2. The Neural Network Zoo
    A. Deep Neural Networks (DNNs)
    B. Convolutional Neural Networks (CNNs)
    C. Recurrent Neural Networks (RNNs)
    D. Graph Neural Networks (GNNs)
    E. Generative Adversarial Network (GANs)
    F. Autoencoders (AEs)
    G. Variational Autoencoders (VAEs)

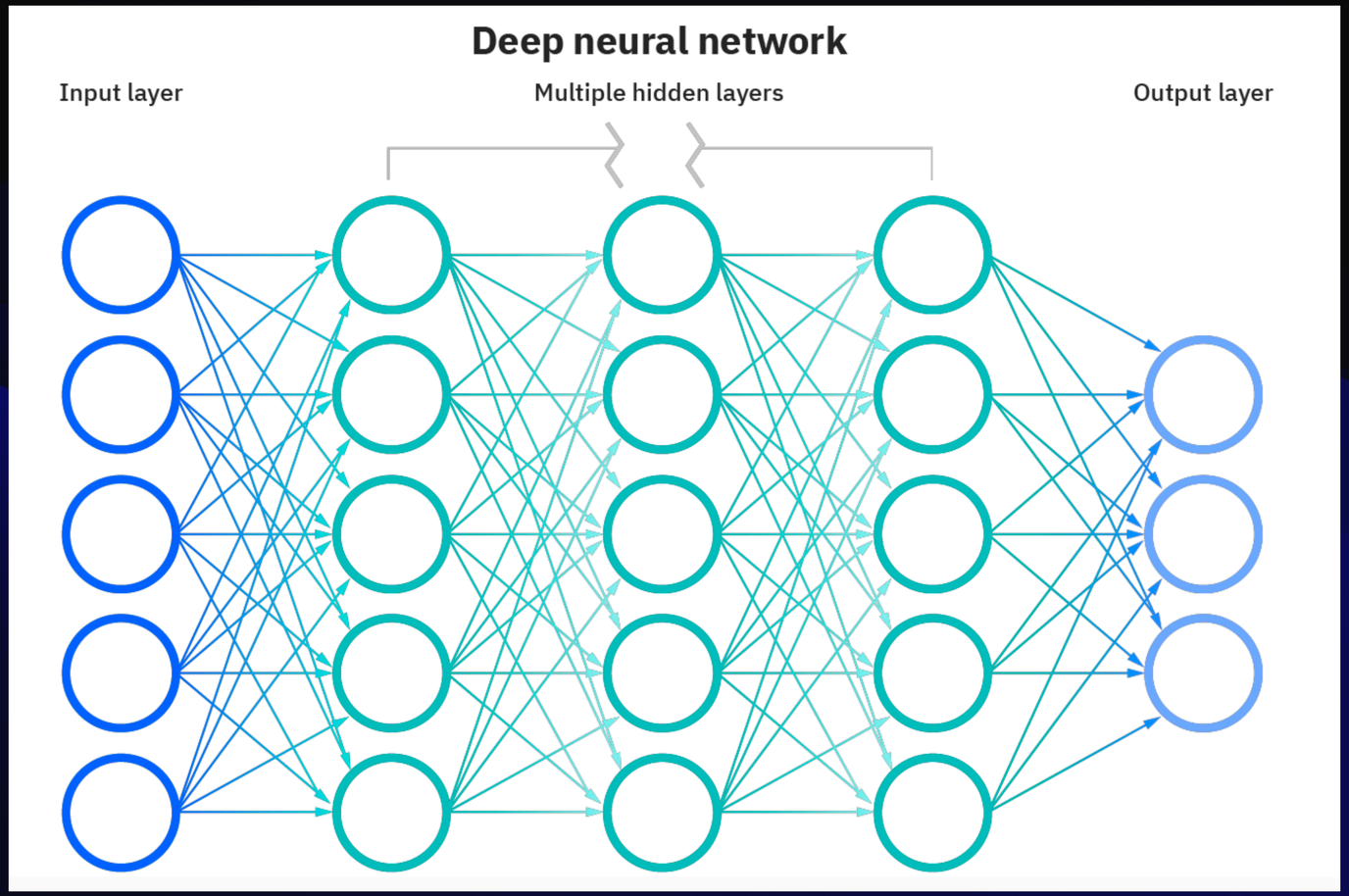
Deep Learning

- Deep Learning (DL) algorithms leverage *depth* to sequentially construct complicated concepts from simpler concepts
- Importance of depth:
  - Mathematical view → composition of many simple functions leads to rich expressivity
  - Computational view → sequential computer program with communicative intermediate states



Feedforward Neural Networks
Fully-Connected Neural Networks (FCNNs)
Deep Neural Networks (DNNs)
Multi-layer Perceptrons (MLPs)


Deep neural network
Input layer
Multiple hidden layers
Output layer



Inputs: images, vector features, words, sentences, etc

E.g. a flattened greyscale image

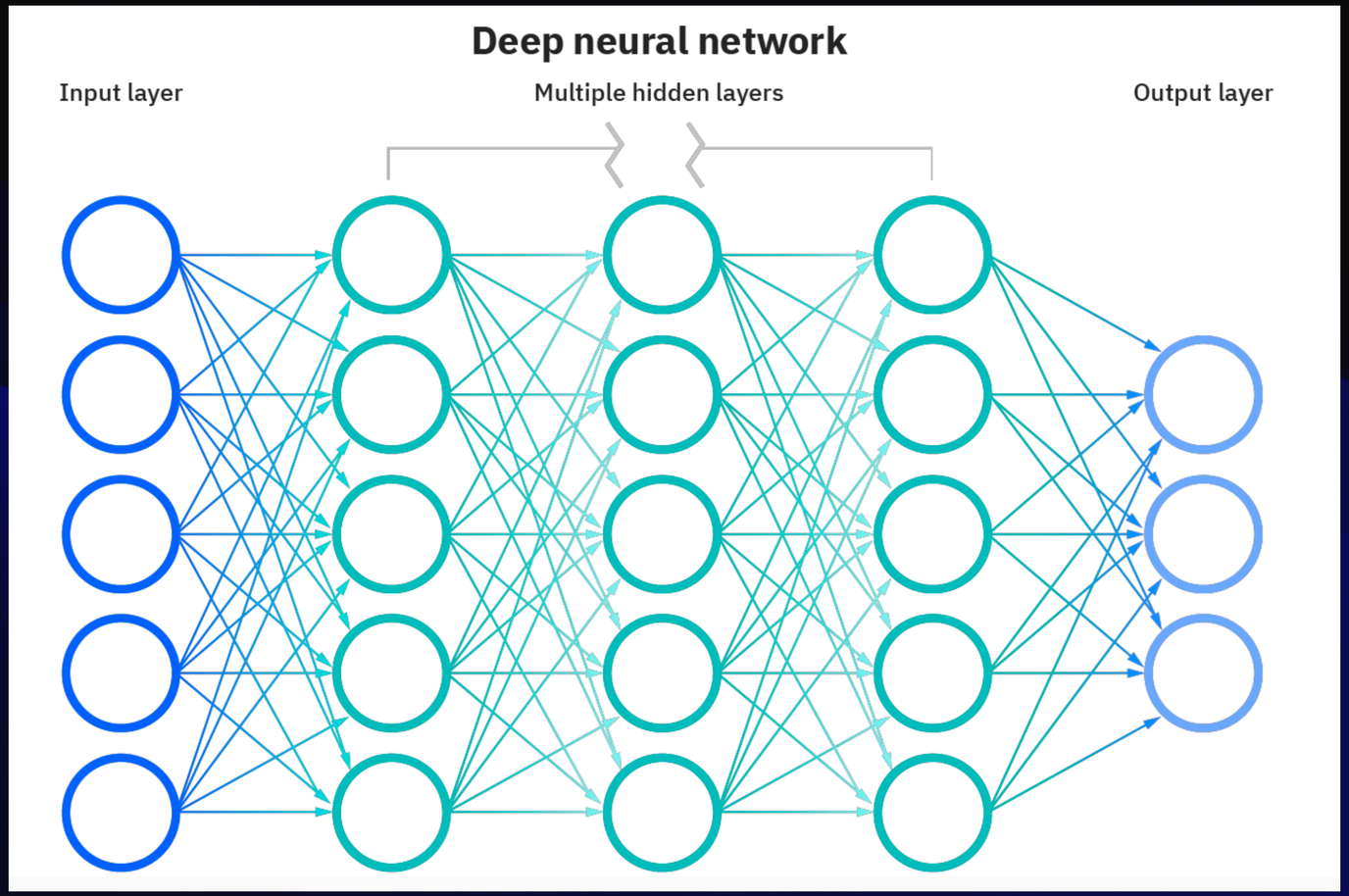
Pixel 0

Pixel 1

Pixel 2

.
.
.

Pixel N



Inputs: images, vector features, words, sentences, etc

E.g. a flattened greyscale image

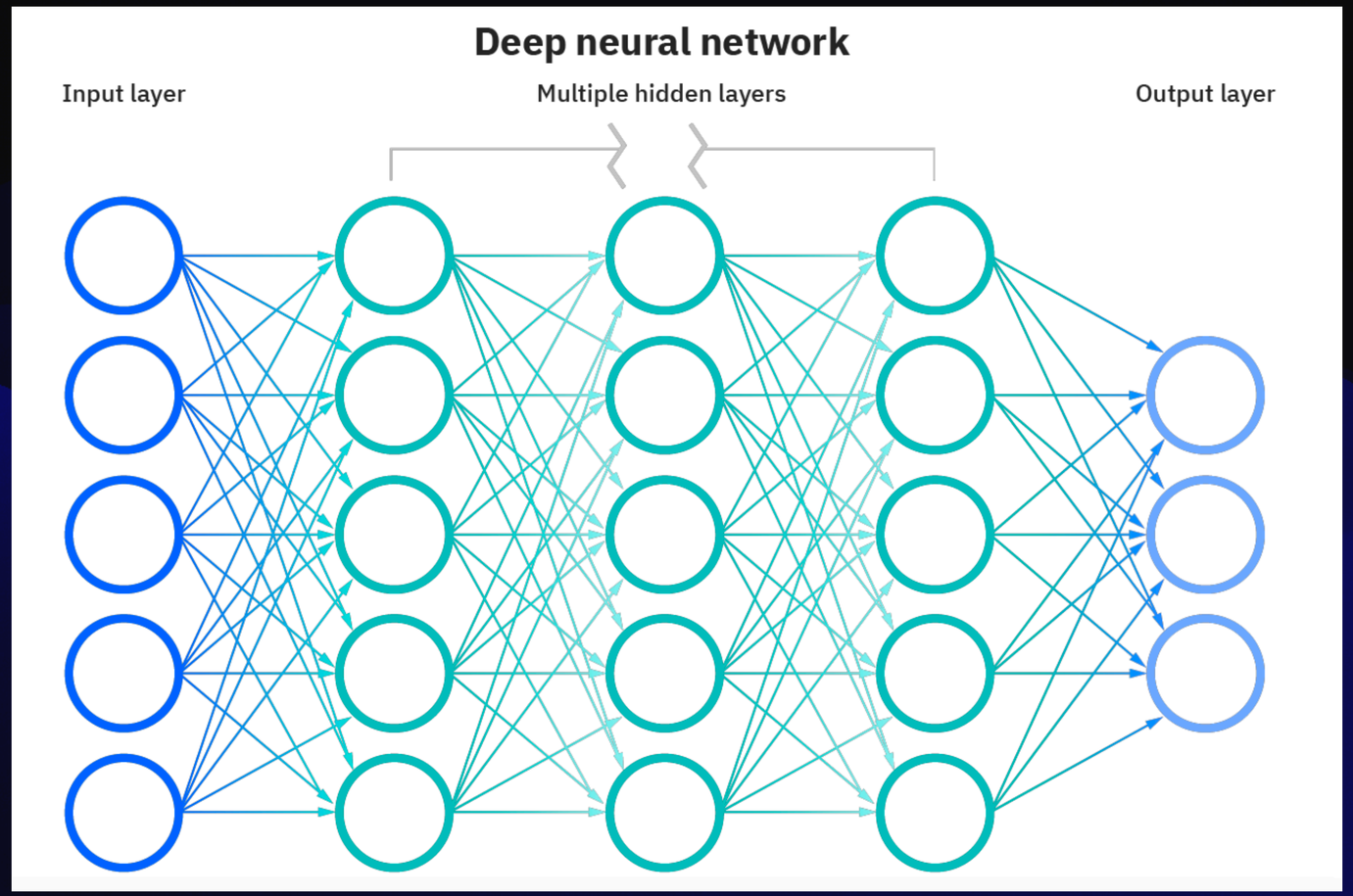
Pixel 0

Pixel 1

Pixel 2

.
.
.

Pixel N



Inputs: images, vector features, words, sentences, etc

Hidden States: learnable parameters are used to transform the input sequentially

E.g. a flattened greyscale image

3 “hidden layers”
Updates of the form
x’ = Wx + b

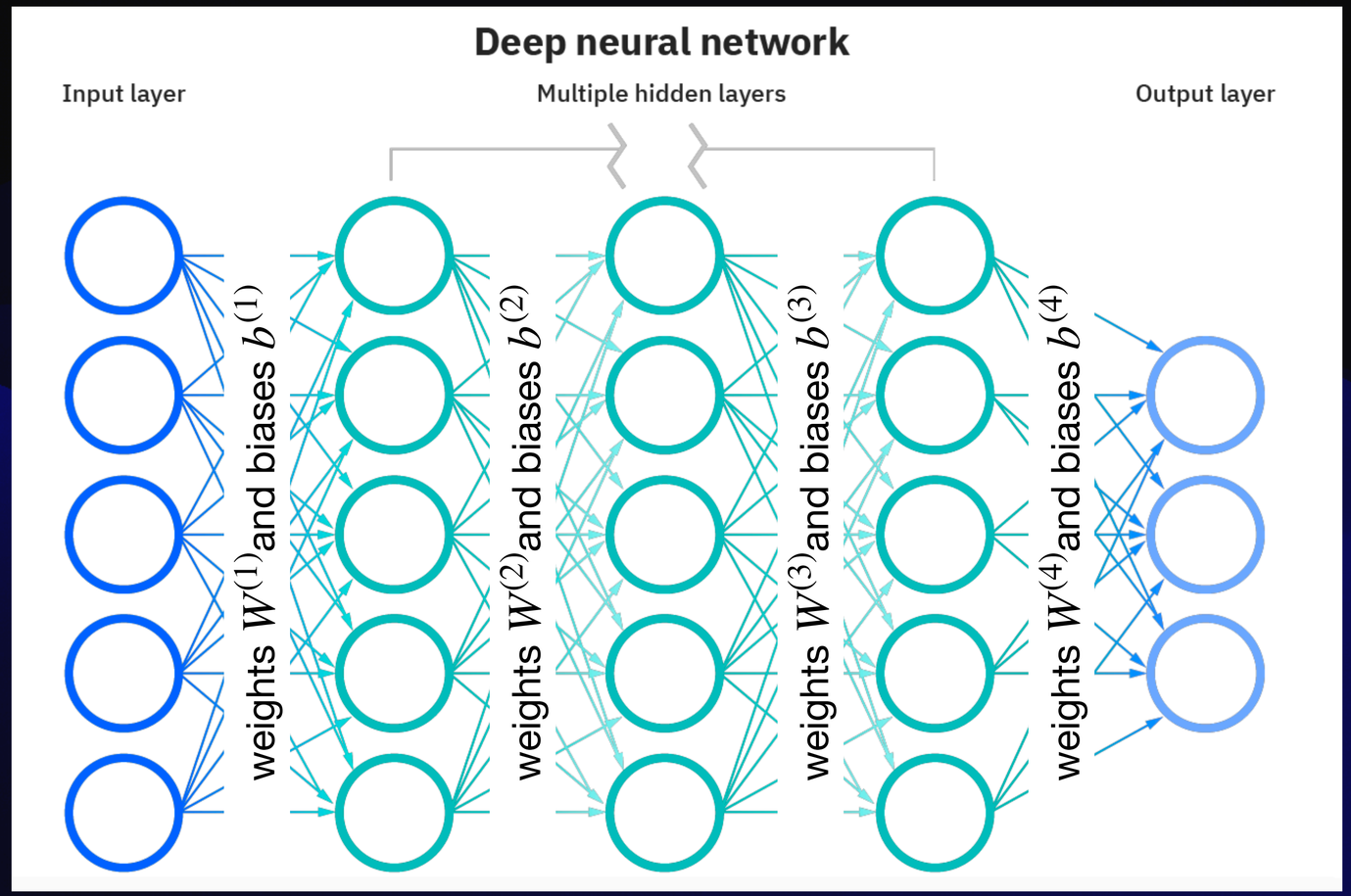
Pixel 0

Pixel 1

Pixel 2

.
.
.

Pixel N



Inputs: images, vector features, words, sentences, etc

Hidden States: learnable parameters are used to transform the input sequentially

E.g. a flattened greyscale image

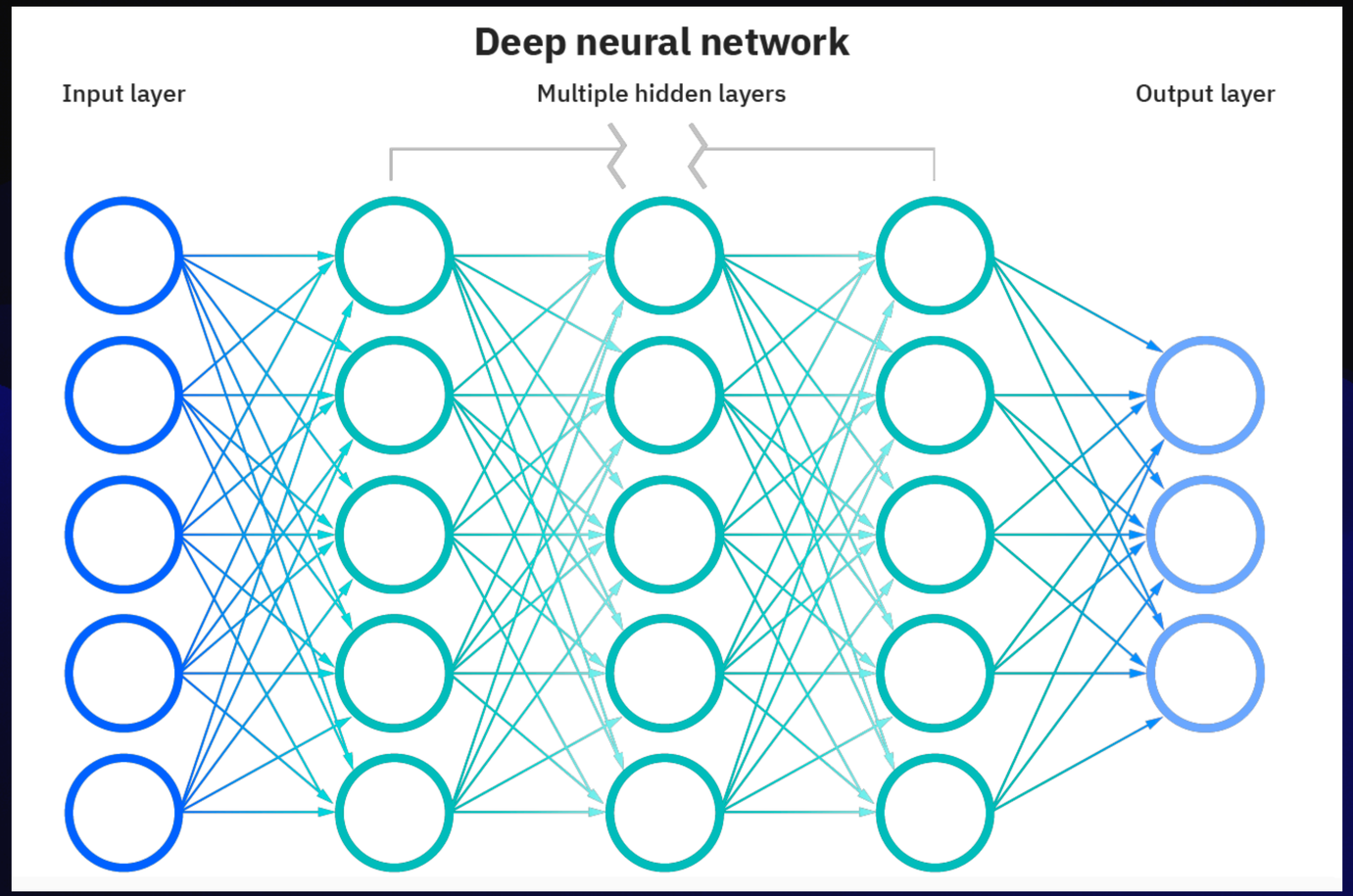
Pixel 0

Pixel 1

Pixel 2

.
.
.

Pixel N



Probability of Boat

Probability of Car

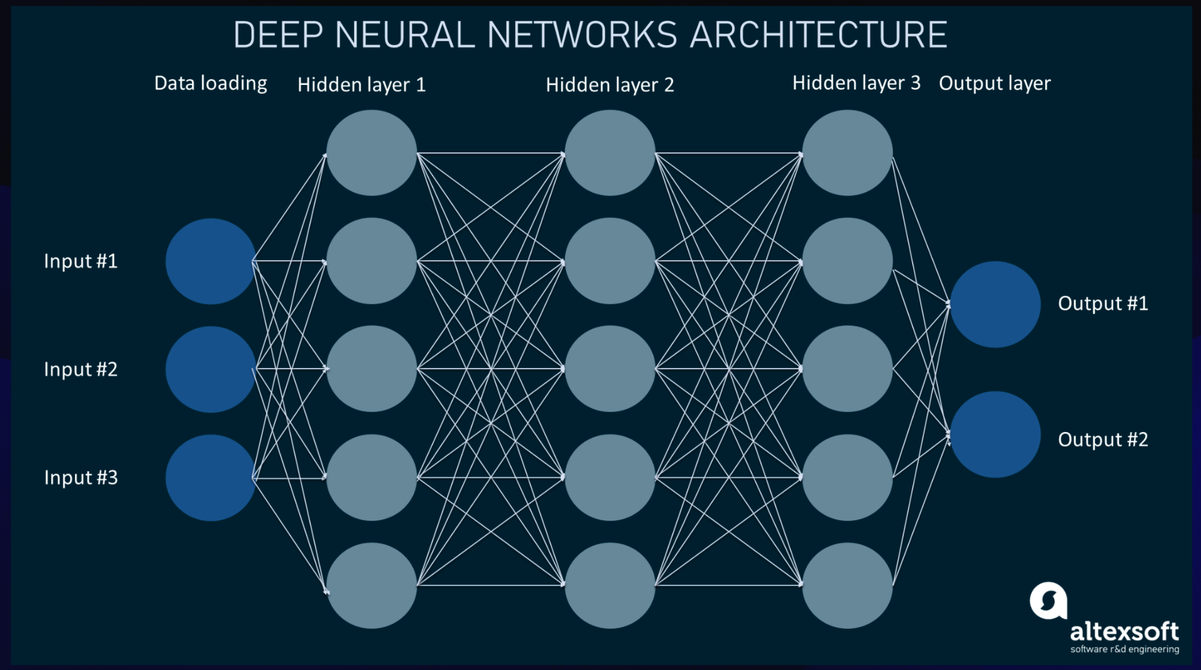
Probability of Airplane

Inputs: images, vector features, words, sentences, etc

Hidden States: learnable parameters are used to transform the input sequentially

Output: model's predictions

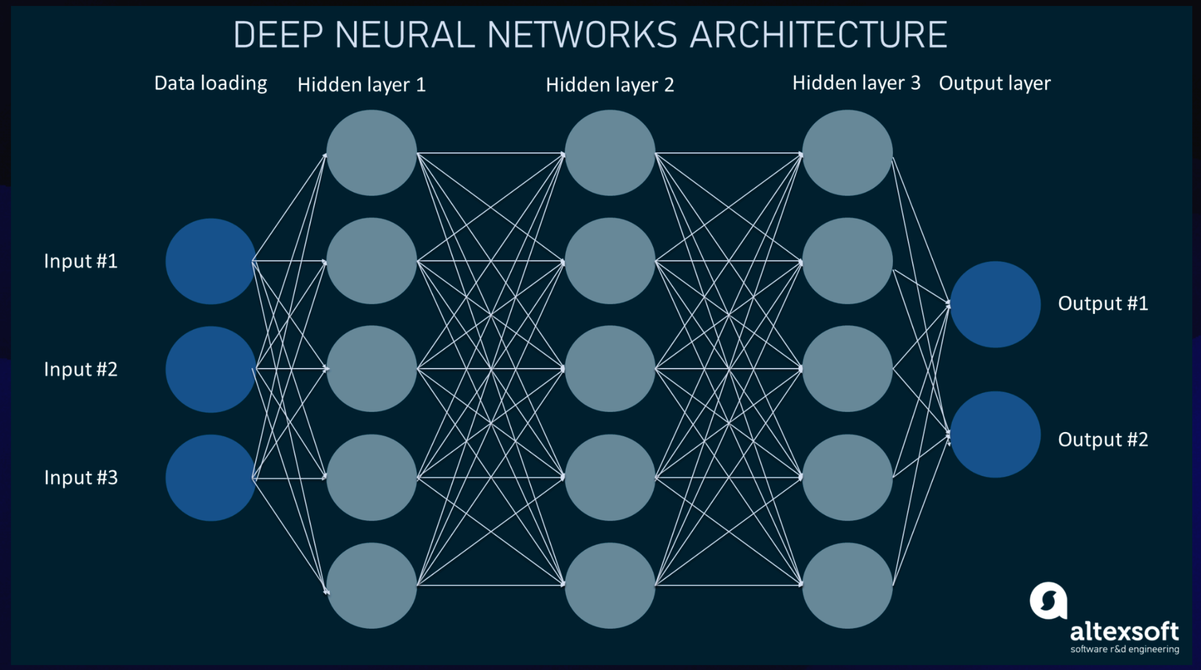
Make a prediction



Objective / Loss Function describes how well the prediction matches a target

Update the model

Make a prediction



Update the model

- Forward pass (inference):

$$z_i^{(\ell+1)} = \sum_{i=1}^{n_\ell} W_{ij}^{(\ell+1)} \sigma(z_j^{(\ell)}) + b_i^{(\ell+1)}$$

- Backward pass (training):

$$W_{ij}^{(\ell+1)} = W_{ij}^{(\ell)} - \gamma \frac{\partial L}{\partial W_{ij}} \Big|_{W_{ij}^{(\ell)}}$$

$$b_i^{(\ell+1)} = b_i^{(\ell)} - \gamma \frac{\partial L}{\partial b_i} \Big|_{b_i^{(\ell)}}$$

(gradient descent)

```python
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

$$z_i^{(\ell+1)} = \sum_{i=1}^{n_\ell} W_{ij}^{(\ell+1)} \sigma(z_j^{(\ell)}) + b_i^{(\ell+1)}$$

```python
def train_one_epoch(epoch_index, tb_writer):
    running_loss = 0.
    last_loss = 0.

    # Here, we use enumerate(training_loader) instead of
    # iter(training_loader) so that we can track the batch
    # index and do some intra-epoch reporting
    for i, data in enumerate(training_loader):
        # Every data instance is an input + label pair
        inputs, labels = data

        # Zero your gradients for every batch!
        optimizer.zero_grad()

        # Make predictions for this batch
        outputs = model(inputs)

        # Compute the loss and its gradients
        loss = loss_fn(outputs, labels)
        loss.backward()

        # Adjust learning weights
        optimizer.step()

        # Gather data and report
        running_loss += loss.item()
        if i % 1000 == 999:
            last_loss = running_loss / 1000 # loss per batch
            print('  batch {} loss: {}'.format(i + 1, last_loss))
            tb_x = epoch_index * len(training_loader) + i + 1
            tb_writer.add_scalar('Loss/train', last_loss, tb_x)
            running_loss = 0.

    return last_loss
```

```python
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

$$W_{ij}^{(\ell+1)} = W_{ij}^{(\ell)} - \gamma \frac{\partial L}{\partial W_{ij}}\bigg|_{W_{ij}^{(\ell)}} \qquad b_i^{(\ell+1)} = b_i^{(\ell)} - \gamma \frac{\partial L}{\partial b_i}\bigg|_{b_i^{(\ell)}}$$

```python
def train_one_epoch(epoch_index, tb_writer):
    running_loss = 0.
    last_loss = 0.

    # Here, we use enumerate(training_loader) instead of
    # iter(training_loader) so that we can track the batch
    # index and do some intra-epoch reporting
    for i, data in enumerate(training_loader):
        # Every data instance is an input + label pair
        inputs, labels = data

        # Zero your gradients for every batch!
        optimizer.zero_grad()

        # Make predictions for this batch
        outputs = model(inputs)

        # Compute the loss and its gradients
        loss = loss_fn(outputs, labels)
        loss.backward()

        # Adjust learning weights
        optimizer.step()

        # Gather data and report
        running_loss += loss.item()
        if i % 1000 == 999:
            last_loss = running_loss / 1000 # loss per batch
            print('  batch {} loss: {}'.format(i + 1, last_loss))
            tb_x = epoch_index * len(training_loader) + i + 1
            tb_writer.add_scalar('Loss/train', last_loss, tb_x)
            running_loss = 0.

    return last_loss
```

DNN Notebook

neural-network-zoo_DNNs.ipynb

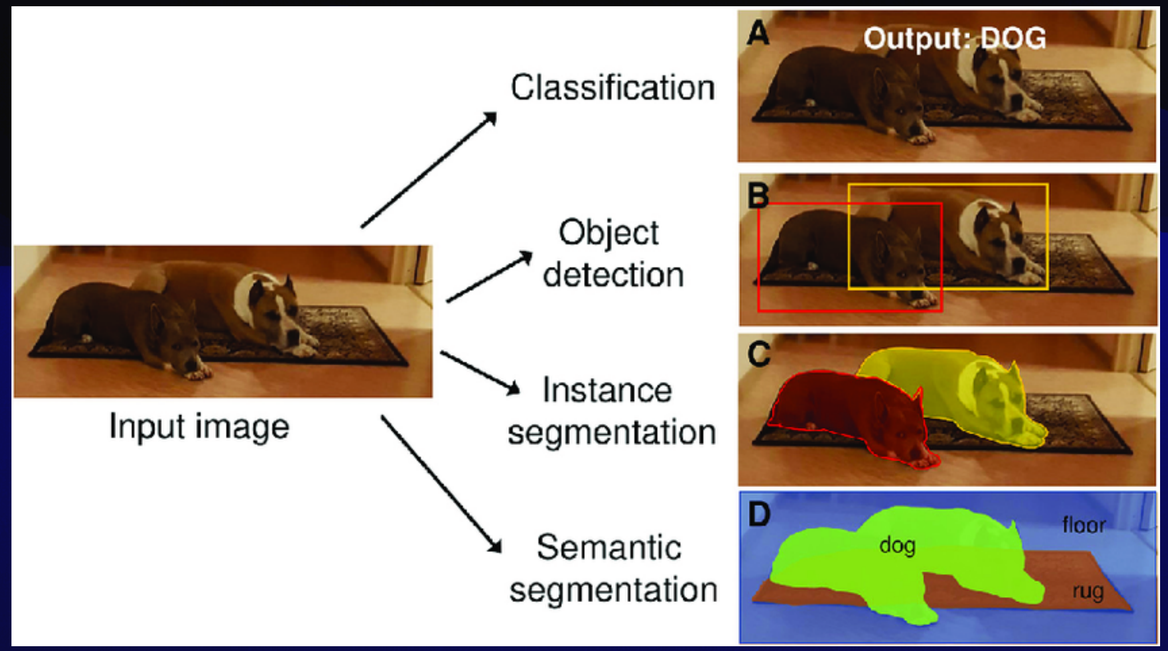
https://tinyurl.com/2fx8smzk

https://github.com/GageDeZoort/neural-network-zoo/tree/main

Convolutional Neural Networks

Computer Vision

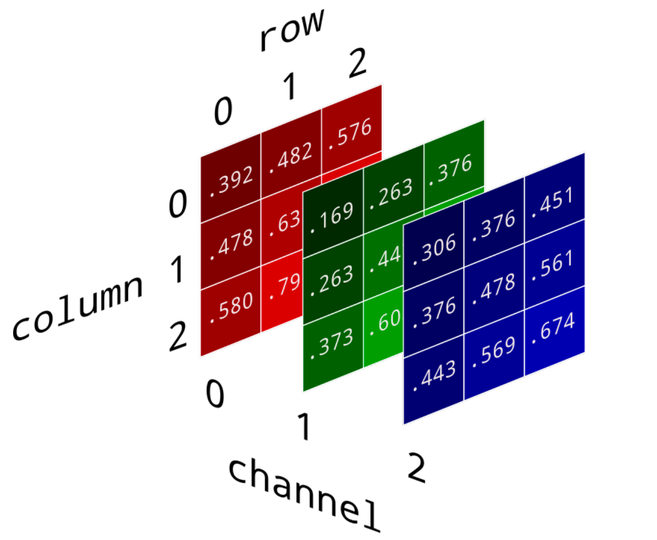
- **Classification**: predict one image label
- **Object detection**: predict bounding boxes for objects in image
- **Instance segmentation**: predict pixel boundaries for objects in images
- **Semantic segmentation**: predict class labels per pixel of objects in image



10.1556/1647.2022.00104

Image Classification
Problem Description

- Image → data on a grid
  - For square images, inputs are $I \in \mathbb{R}^{n_{\text{pixels}} \times n_{\text{pixels}} \times n_{\text{features}}}$
  - Example: 3x3 RGB frog images
- CNN maps image to probability predictions
  - Example: CNN(image) = P(frog)

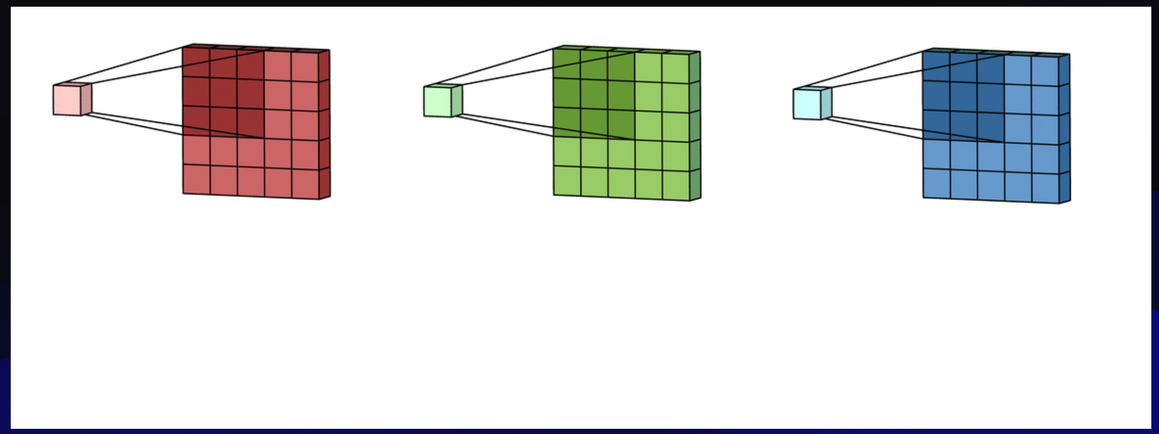


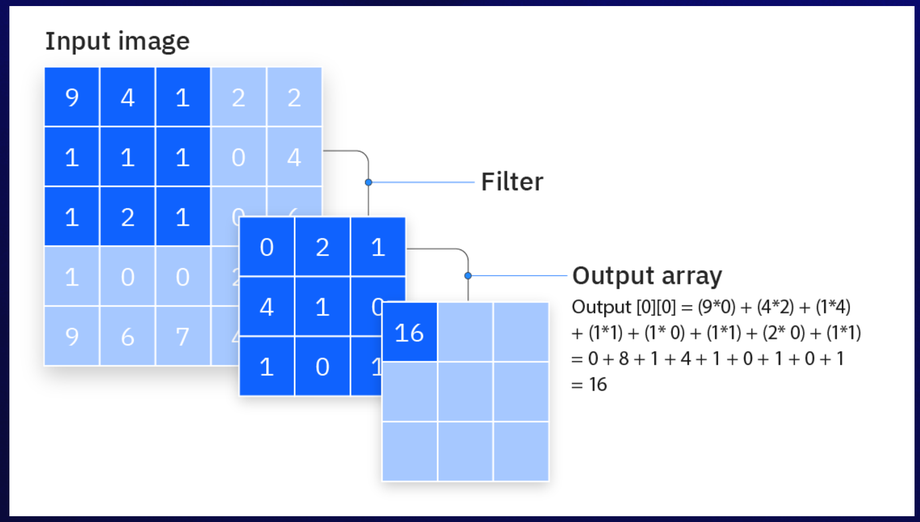
https://ai.plainenglish.io/complete-guide-image-classification-on-cifar-10-chapter-i-multiclass-logistic-regression-525daee0775b

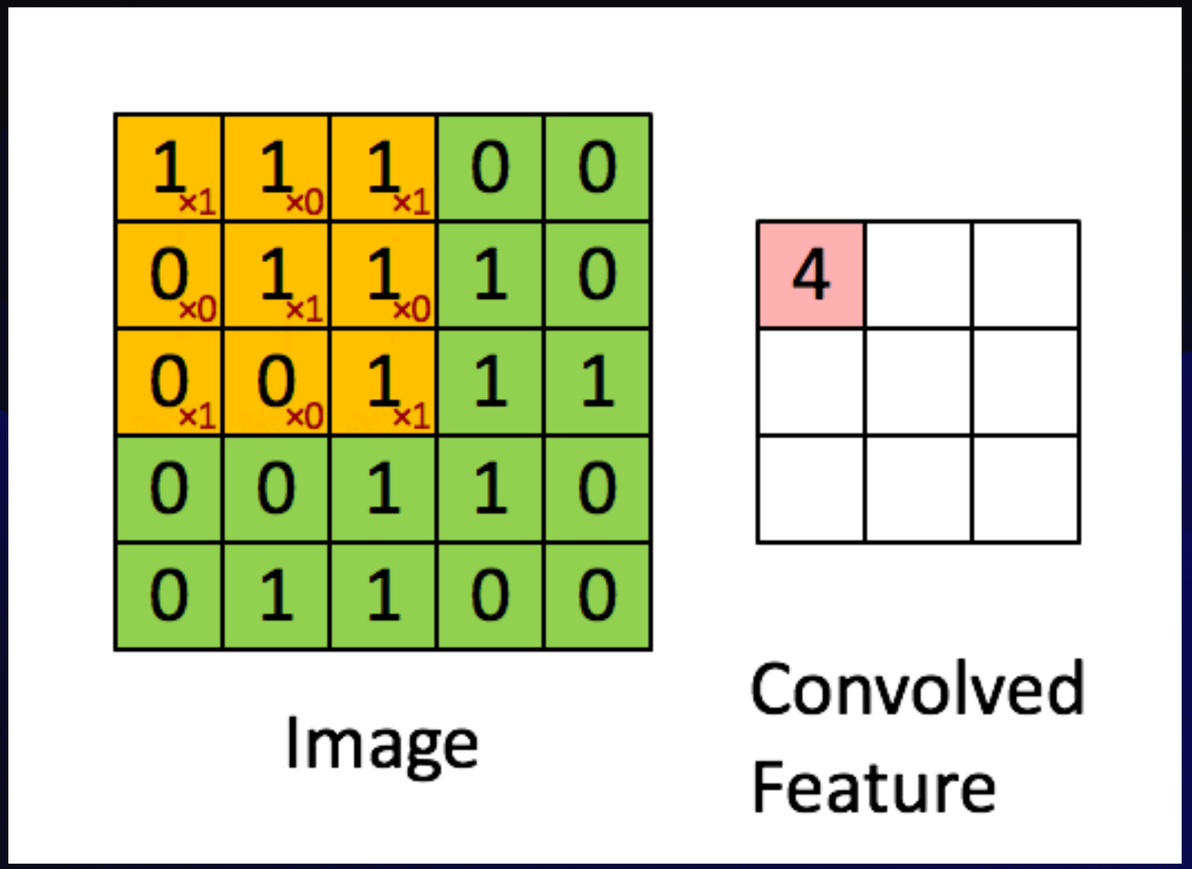
Convolutional Neural Networks

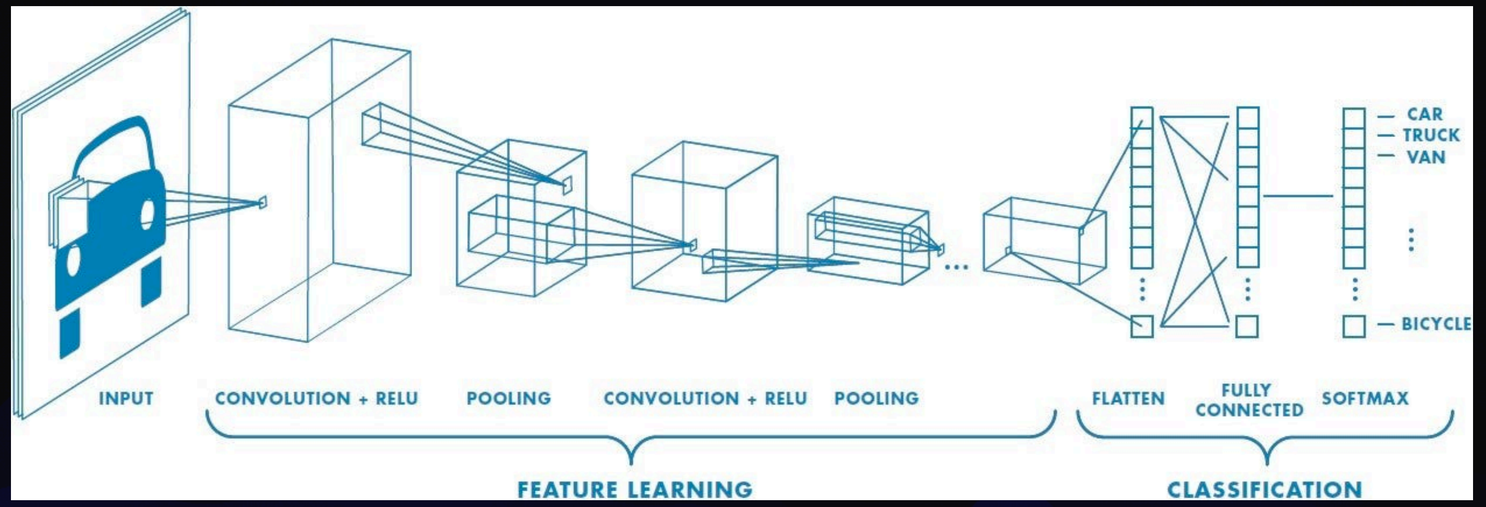
Convolutional Layers

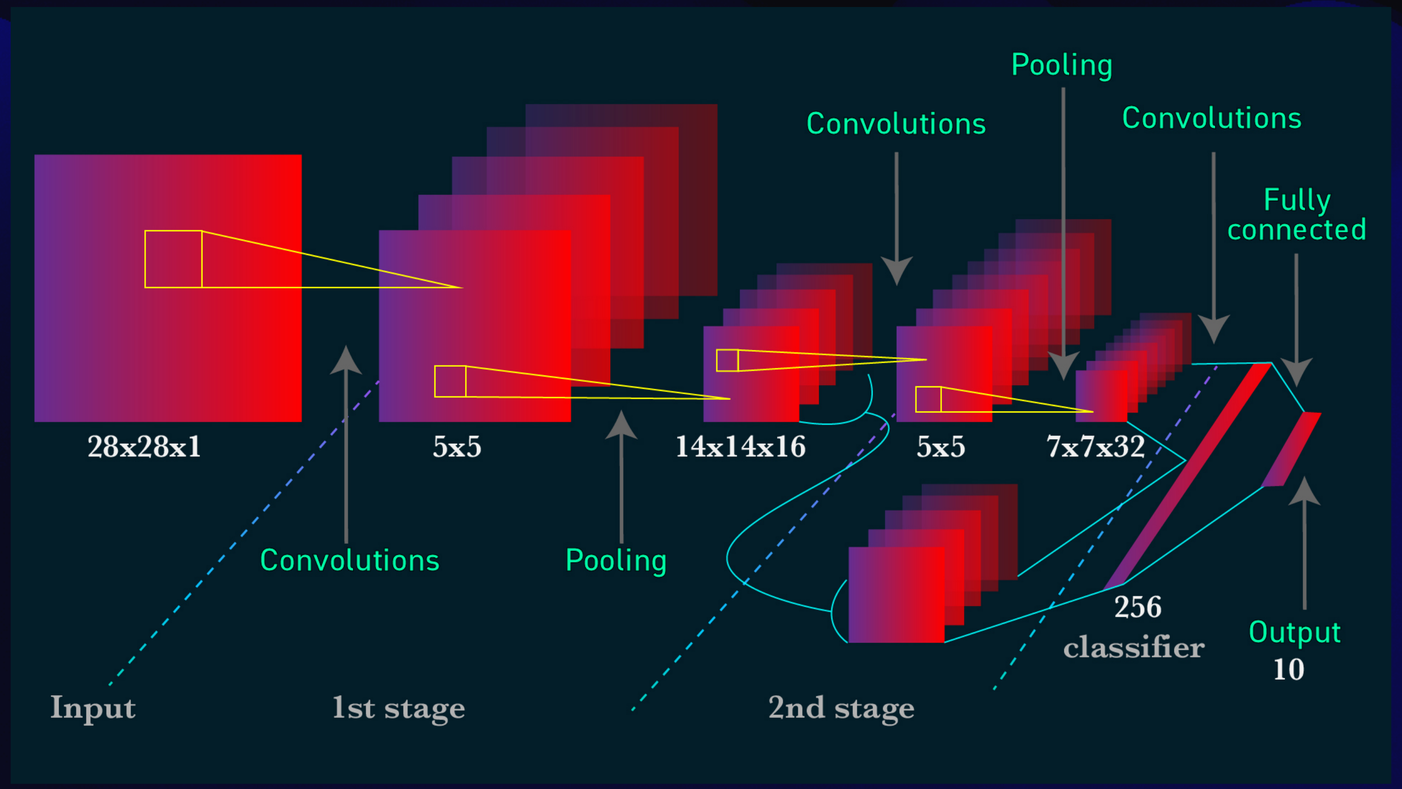
- Convolutional layers act like fully-connected layers in DNNs, but have several key differences:
  - The weights in the convolutional layer (*filter*) are applied to small patches of the image
  - Parameter sharing - the *same weights* are applied equally to every region of the input image!
  - Tuning the convolutional weights allows the network to look for different features - *feature extraction*!






1×1 1×0 1×1 0 0
0×0 1×1 1×0 1 0
0×1 0×0 1×1 1 1
0 0 1 1 0
0 1 1 0 0
Image
4
Convolved Feature


CAR
TRUCK
VAN
BICYCLE
INPUT
CONVOLUTION + RELU
POOLING
CONVOLUTION + RELU
POOLING
FLATTEN
FULLY CONNECTED
SOFTMAX
FEATURE LEARNING
CLASSIFICATION


Pooling
Convolutions
Convolutions
Fully connected
28x28x1
5x5
14x14x16
5x5
7x7x32
Convolutions
Pooling
256 classifier
Output 10
Input
1st stage
2nd stage

CNN Notebook

neural-network-zoo_CNNs.ipynb

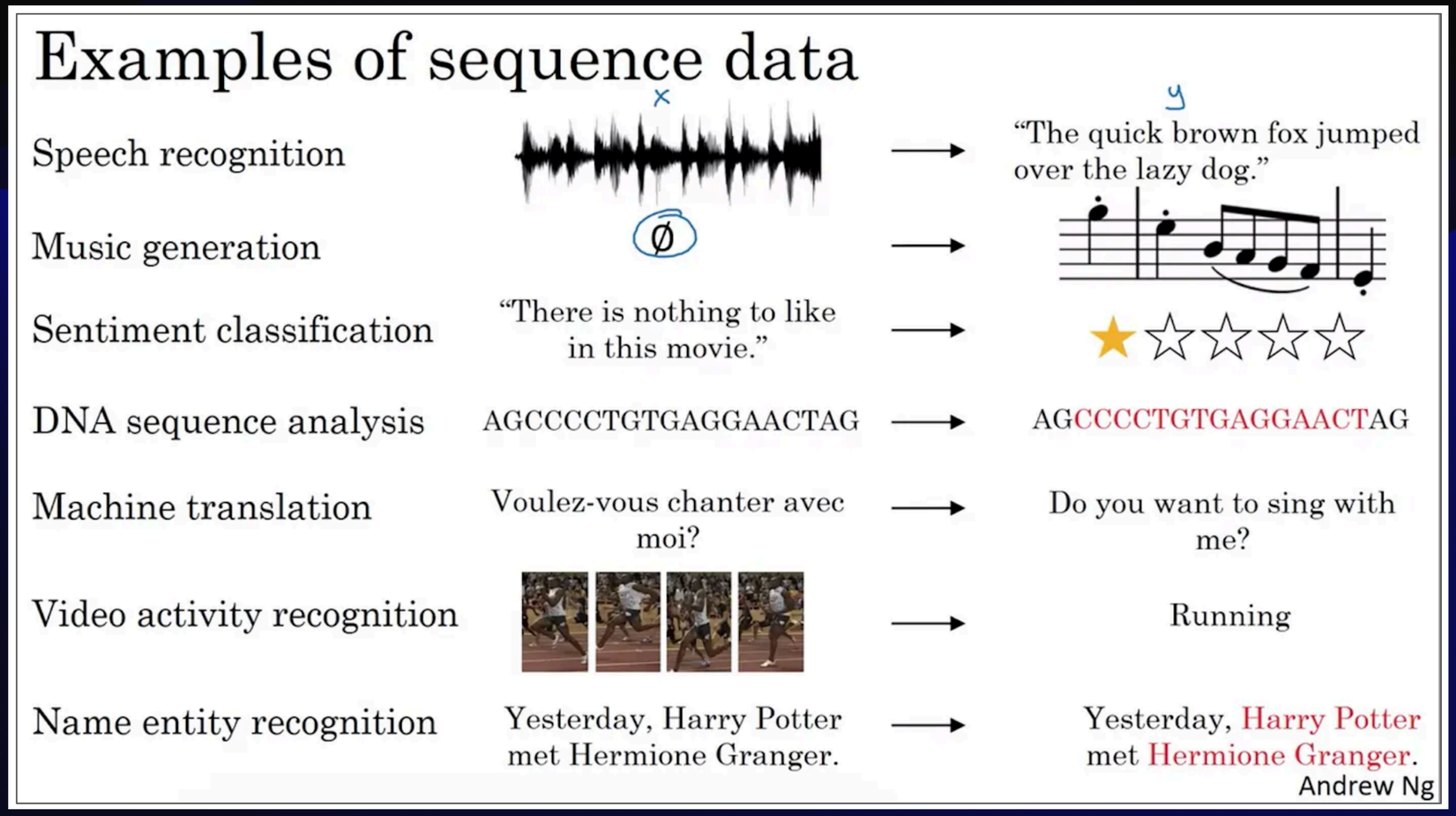
https://tinyurl.com/2fx8smzk

https://github.com/GageDeZoort/neural-network-zoo/tree/main

Recurrent Neural Networks

Recurrent Neural Networks

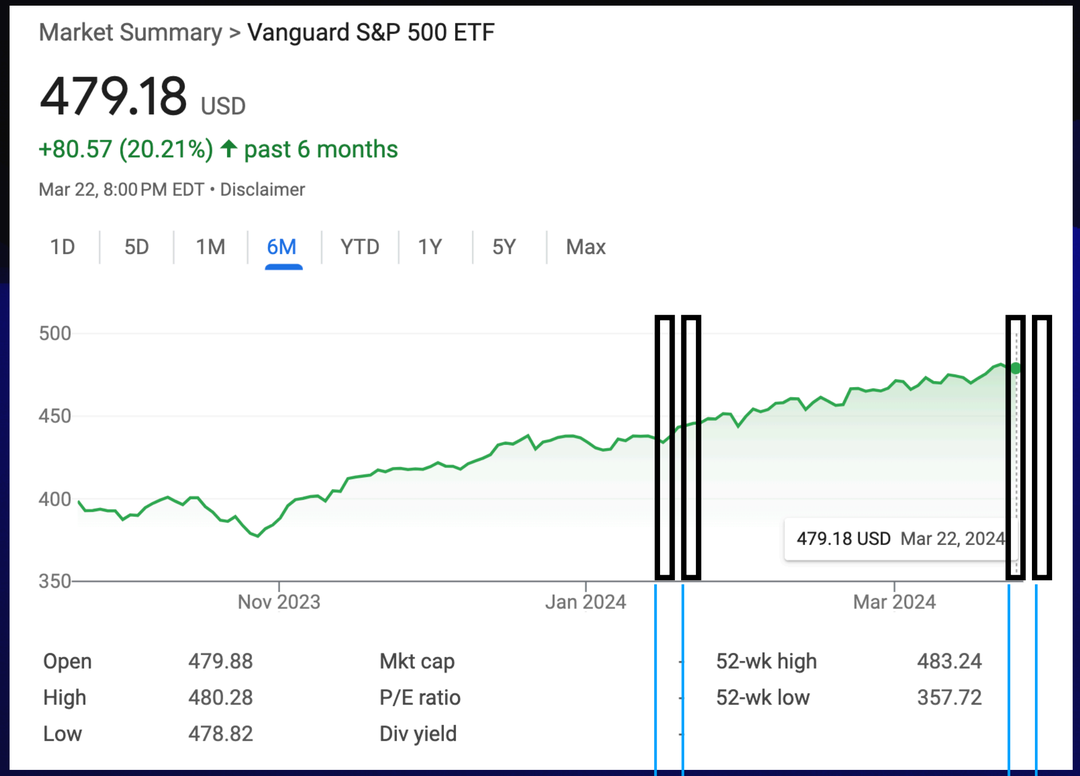
Predictions on Sequential Data



https://aiml.com/what-does-sequential-data-mean-which-models-are-best-suited-for-handling-sequential-data/

Recurrent Neural Networks
Basic Idea

- Time-indexed inputs: $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(T)}$
  - word 1, word 2, word 3,…
  - stock price day 1, stock price day 2,…
- Parameter sharing: apply the *same function* (with learnable weights) at every time *t*



Function *f* Function *f*

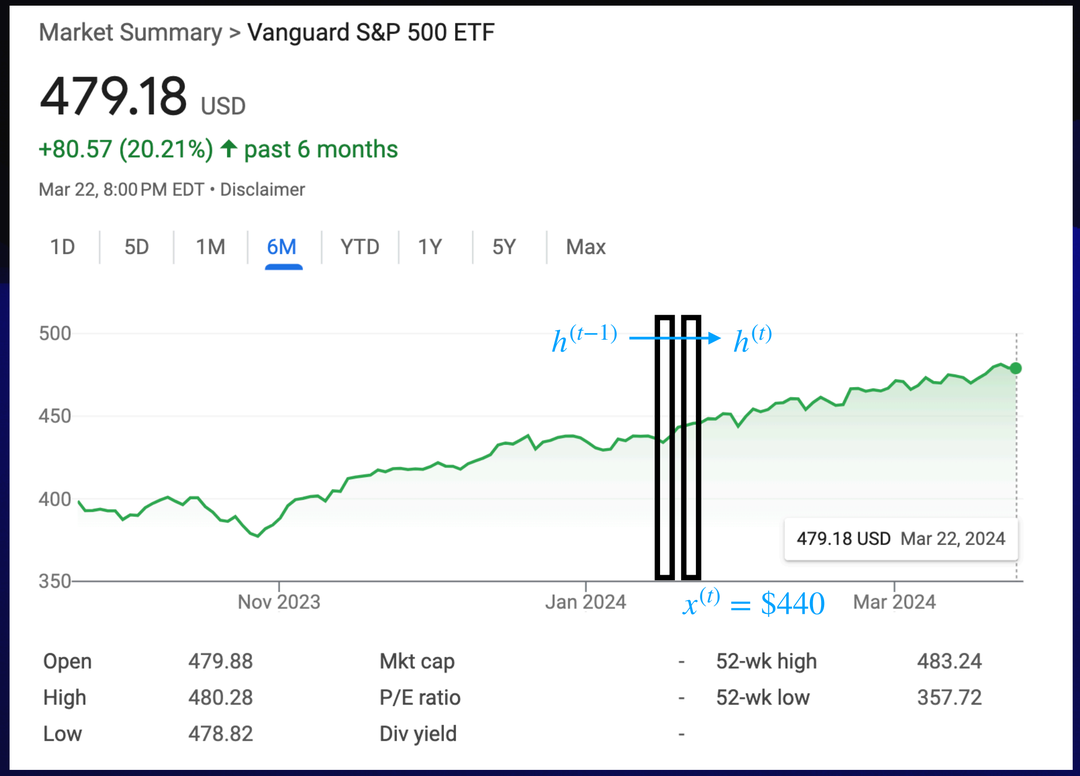
Recurrent Neural Networks

Basic Idea

Given a set of learnable parameters θ, the hidden units in many RNNs are calculated via

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \theta)$$

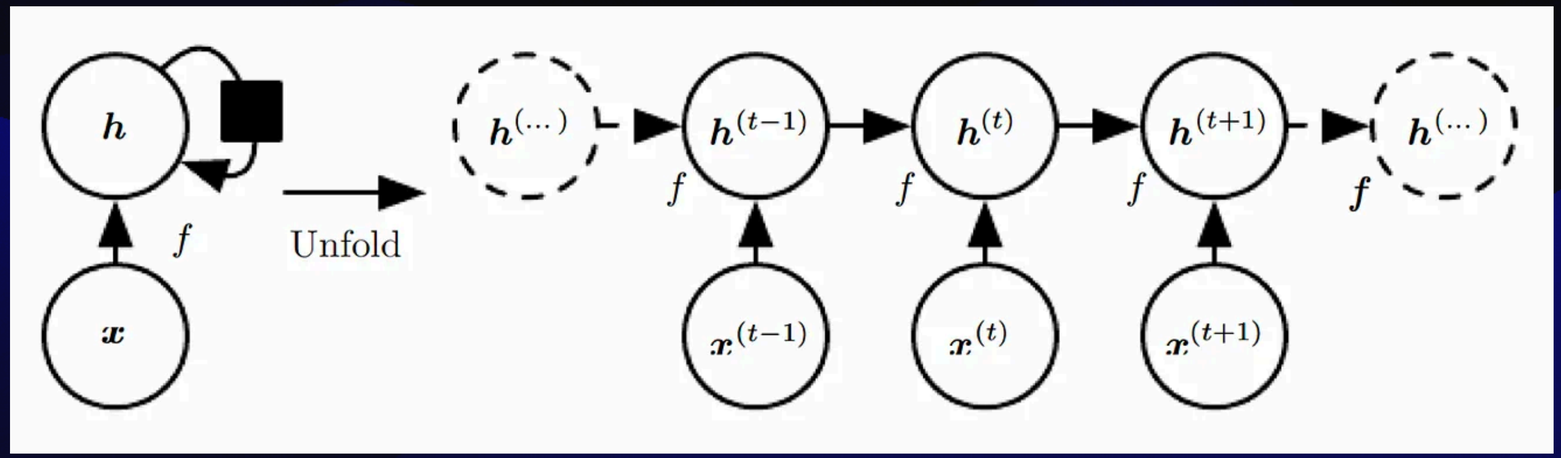
"Hidden Representations" *h(t)* leveraged to make predictions



Recurrent Neural Networks

Basic Idea

Image from *Deep Learning* by Goodfellow et al.



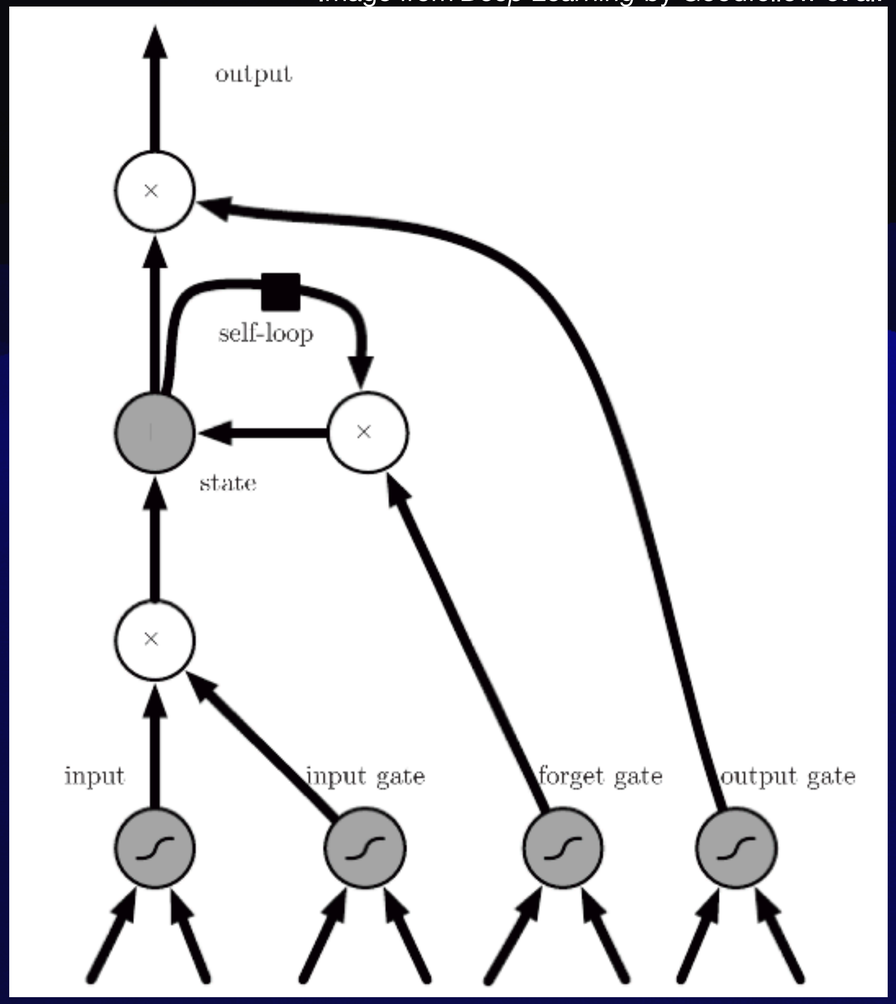
RNN with no outputs; information is processed sequentially, taking into account both $\mathbf{x}^{(t)}$ and $h^{(t-1)}$ but applying the *same function* $f(\cdot\,;\theta)$ *at each timestep*

Long Short-Term Memory (LSTM)

An Upgraded RNN Module

- RNNs are finicky to train; they often suffer from exploding/vanishing gradients
- This has motivated the development of more advanced RNNs like LSTMs
- The LSTM is a recurrent “cell” that is applied to all timesteps equally

Image from *Deep Learning* by Goodfellow et al.



RNN Notebook

neural-network-zoo_RNNs.ipynb

https://tinyurl.com/2fx8smzk

https://github.com/GageDeZoort/neural-network-zoo/tree/main

Graph Neural Networks

Graph Neural Networks

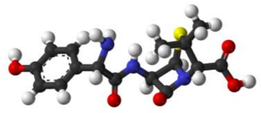
Intro to Graphs

- Graphs: vertices (nodes) connected by edges (relationships)
- Examples:
  - Social networks of people connected by friendships
  - Molecules of atoms connected by nodes
  - Planets connected by mutual gravitational interactions

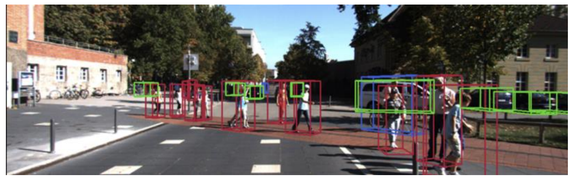
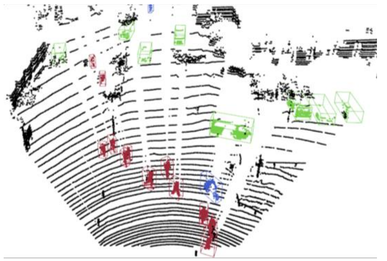
Graph Neural Networks

Graph Learning Tasks

DRUG DISCOVERY

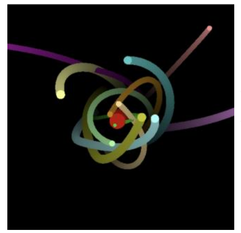
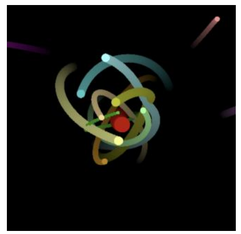
$GNN($  $) \rightarrow$ molecule property

INSTANCE SEGMENTATION

$GNN($  $) \rightarrow$ 

2003.01251.pdf (arxiv.org)

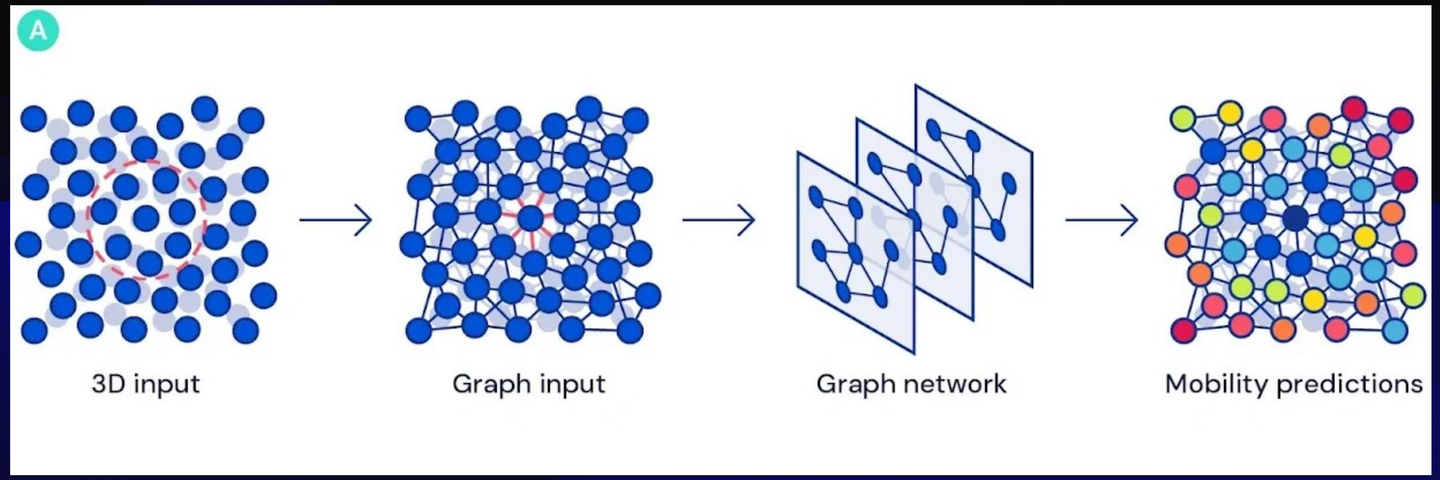
PHYSICS SIMULATION

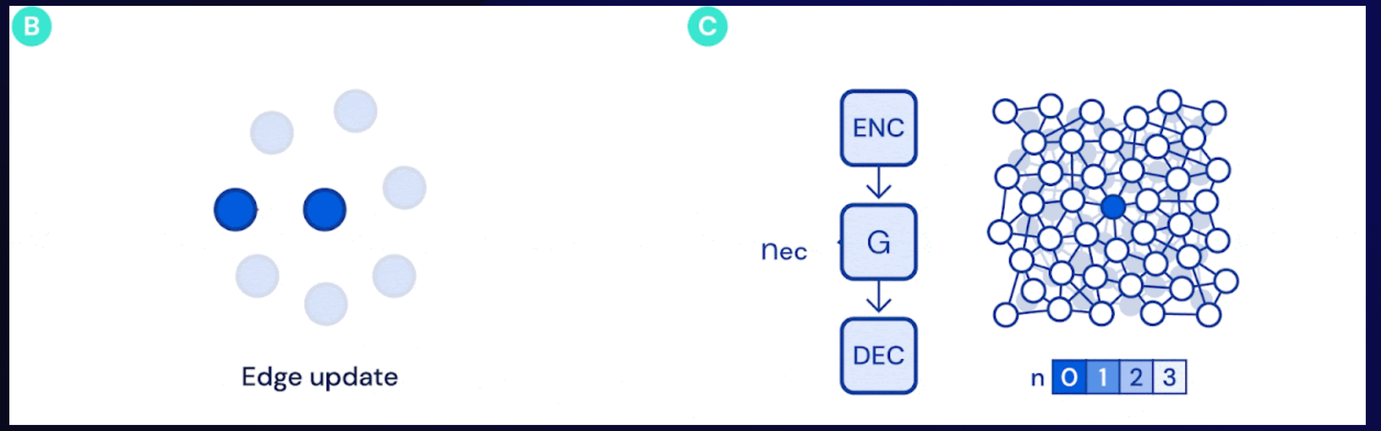
$GNN($  $) \rightarrow$ 

[1612.00222] Interaction Networks for Learning about Objects, Relations and Physics (arxiv.org)

Graph Neural Networks

Message Passing NNs (MPNNs)





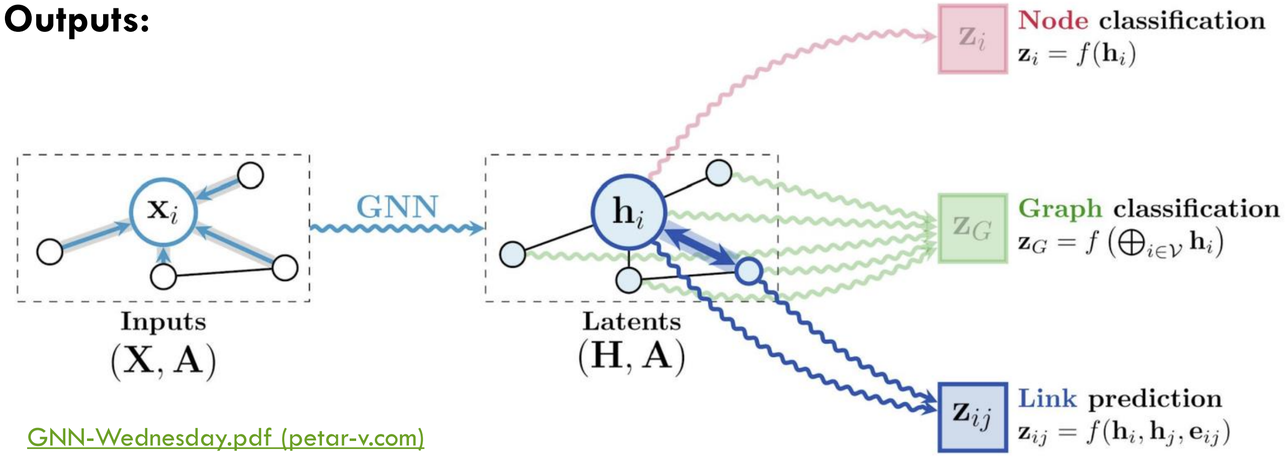
https://deepmind.google/discover/blog/towards-understanding-glasses-with-graph-neural-networks/

Graph Neural Networks
Message Passing NNs (MPNNs)

Generic MPNN Layers:

$$\boldsymbol{h}_u^{(k)} = \phi^{(k)}\left[\boldsymbol{h}_u^{(k-1)}, \bigoplus_{v \in N(u)} \psi^{(k)}\left(\boldsymbol{h}_u^{(k-1)}, \boldsymbol{h}_v^{(k-1)}, \boldsymbol{e}_{uv}^{(k-1)}\right)\right]$$

Outputs:



GNN-Wednesday.pdf (petar-v.com)

GNN Notebook!

GNN Notebook

neural-network-zoo_GNNs.ipynb

https://tinyurl.com/2fx8smzk

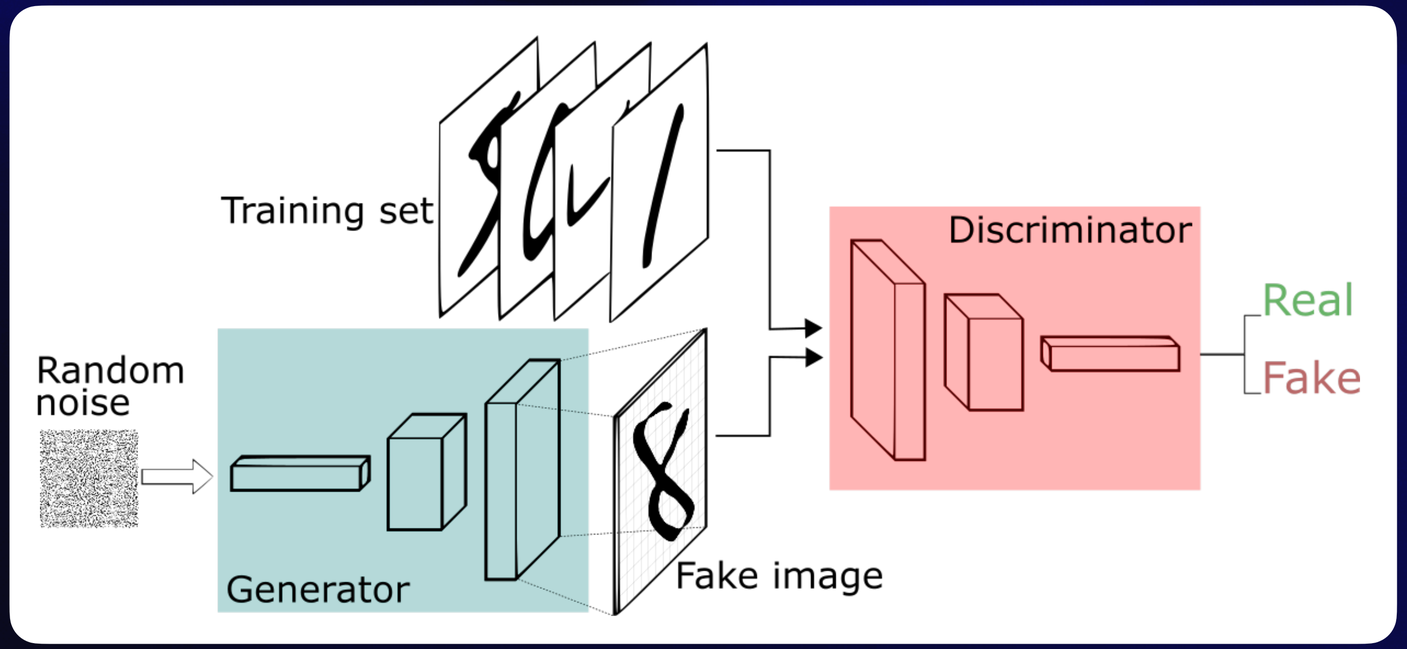
https://github.com/GageDeZoort/neural-network-zoo/tree/main

Generative Adversarial Networks

Generative Adversarial Networks (GANs)

Intro to GANs

- “Generative” AI: use ML to create new images, sounds, etc.
- GANs: two agents (the *generator* and the *discriminator*) are given competing tasks:



https://sthalles.github.io/intro-to-gans/

GAN Notebook

neural-network-zoo_GANs.ipynb

https://tinyurl.com/2fx8smzk

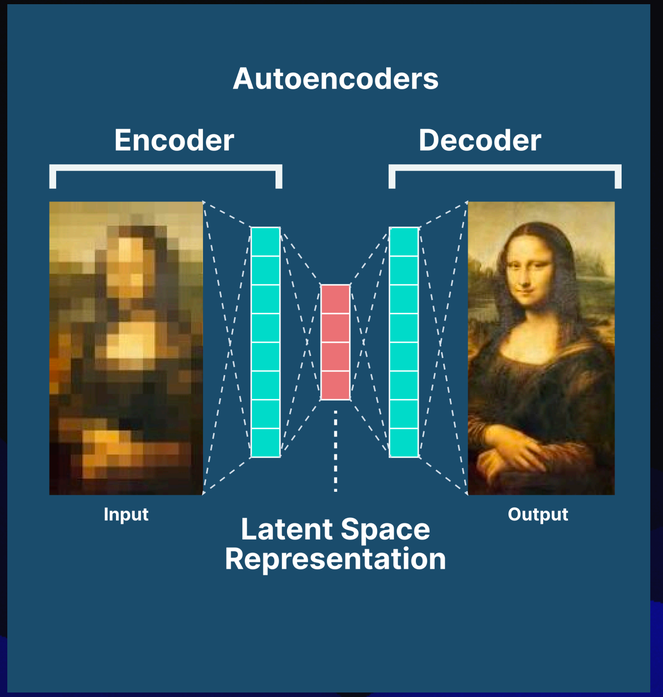
https://github.com/GageDeZoort/neural-network-zoo/tree/main

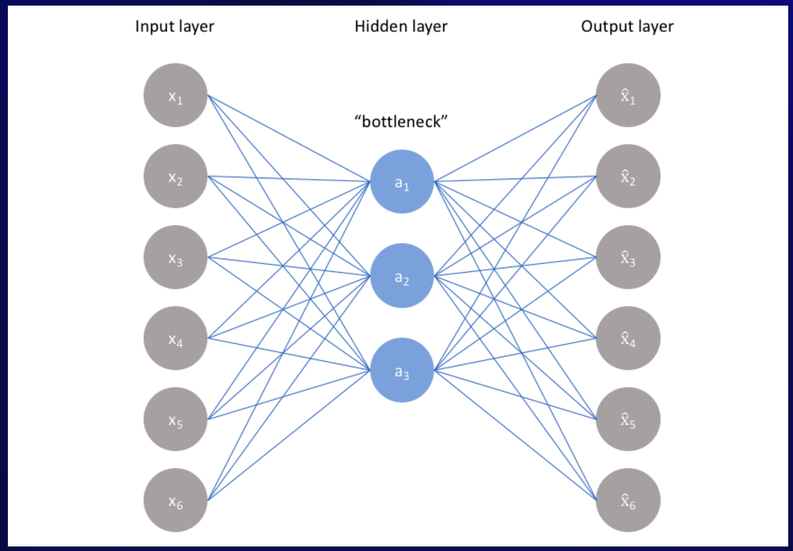
Autoencoders

Autoencoders

Learning Efficient Codings

- Autoencoders are used to produce compressed data representations (*dimensionality reduction)*
  - **Encoder**: produces a lower-dimensional (compressed) “latent” representation of the input data
  - **Decoder**: given the compressed representation, reconstruct the original data
- Decoded representations typically less noisy,
- Uses: efficient encoding, image denoting, generative modeling, anomaly detection

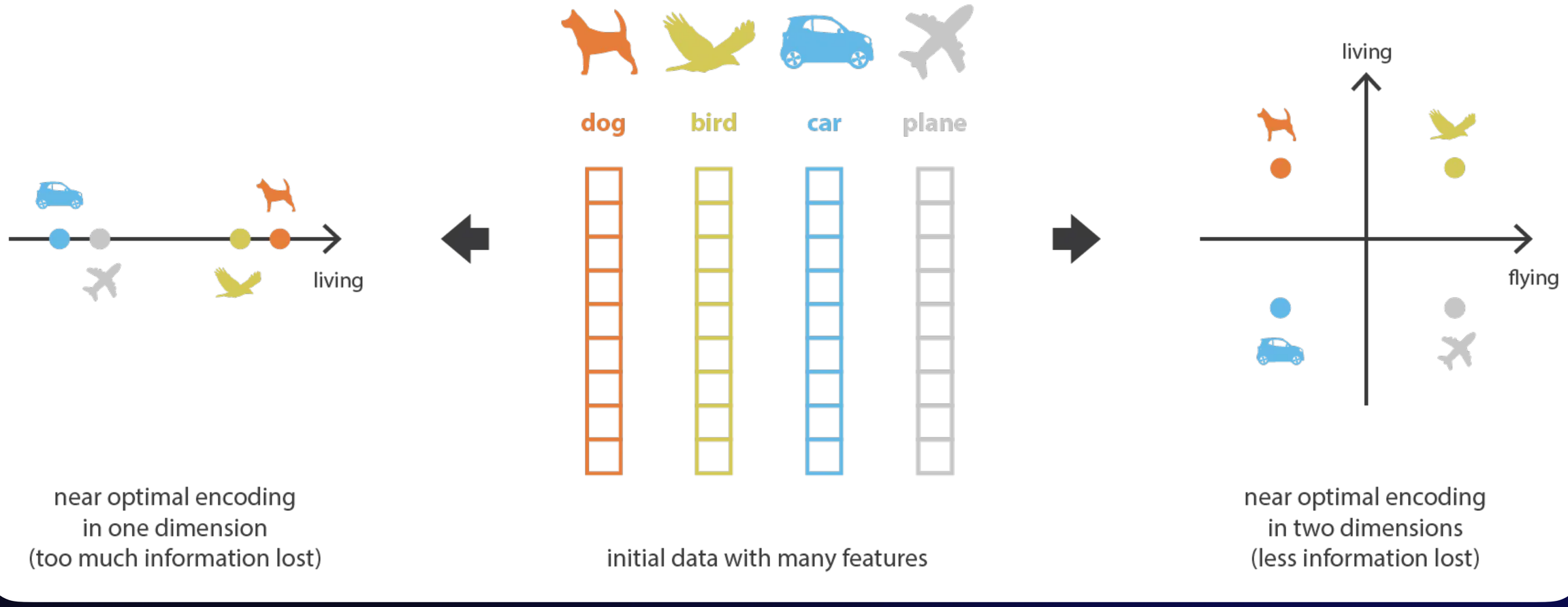




https://www.v7labs.com/blog/autoencoders-guide#:~:text=An%20autoencoder%20is%20an%20unsupervised,even%20generation%20of%20image%20data.

Autoencoders

Learning Efficient Codings



Autoencoder Notebook!

AE Notebook

neural-network-zoo_AEs.ipynb

https://tinyurl.com/2fx8smzk

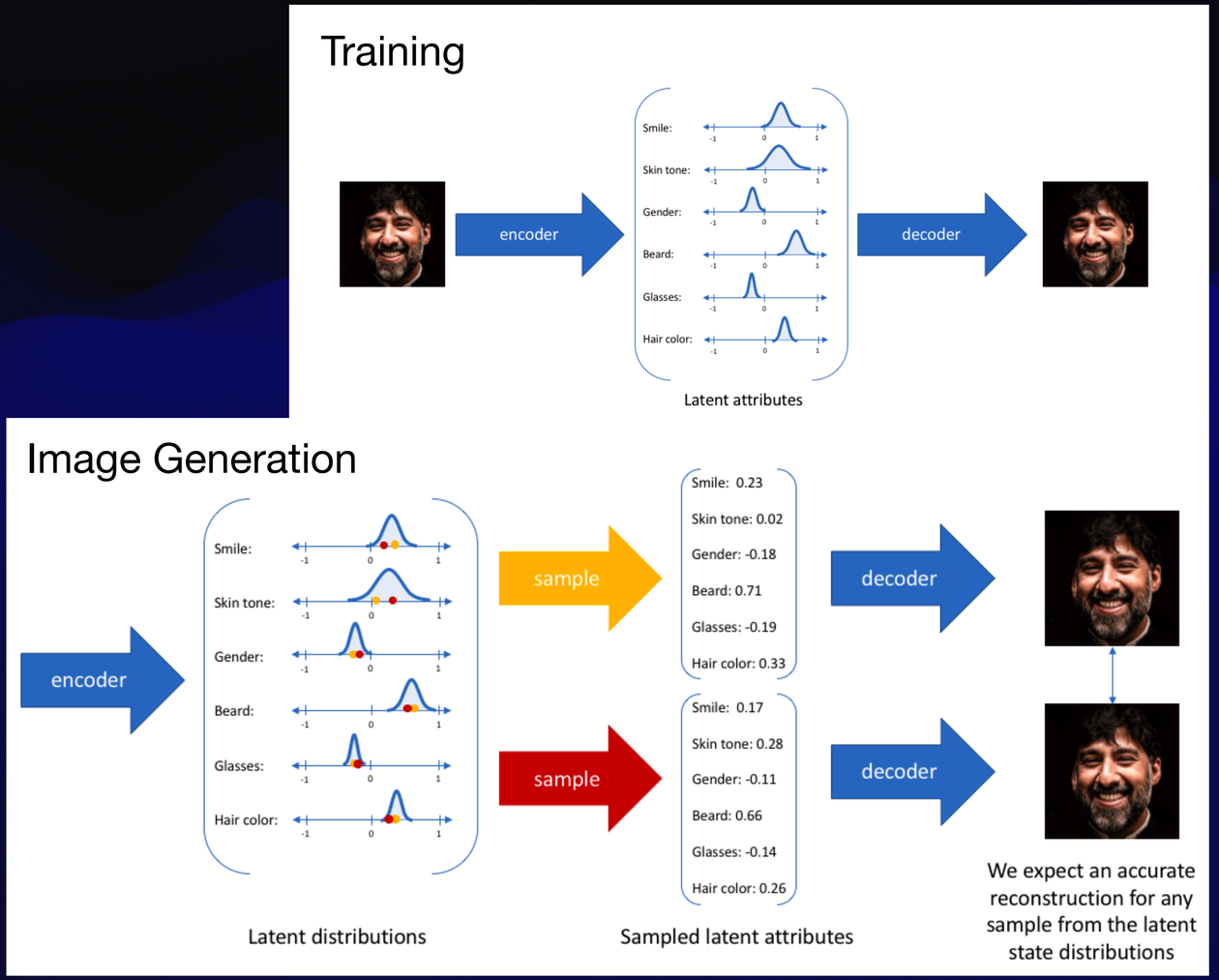
https://github.com/GageDeZoort/neural-network-zoo/tree/main

Variational Autoencoders

Variational Autoencoder (VAEs)

Generative Modeling via Autoencoders

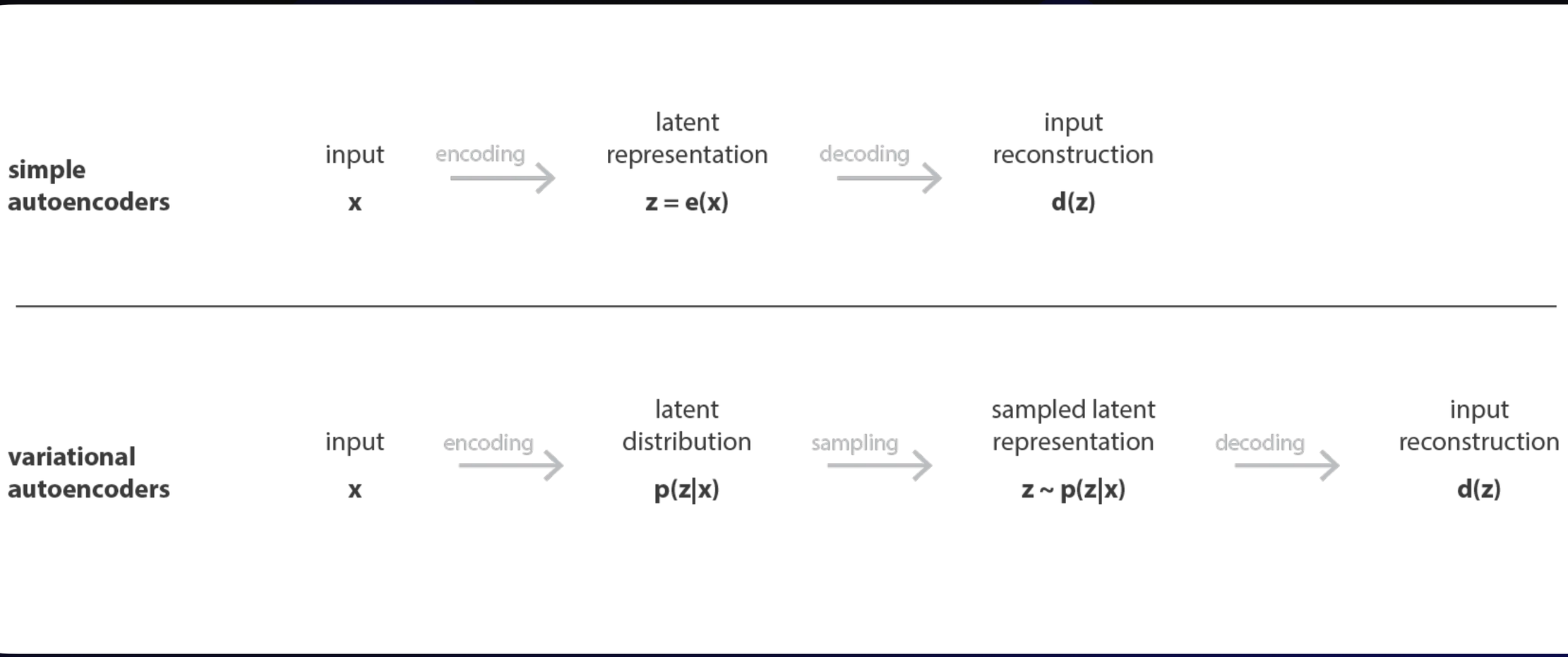
- Generate realistic images from random noise
  - **Encoder**: predict means and standard deviations of a *probability distribution* over the latent features
  - **Decoder:** given a random sample from the latent distributions, produce the corresponding output



https://www.jeremyjordan.me/variational-autoencoders/

Variational Autoencoder (VAEs)

Generative Modeling via Autoencoders



VAE Notebook

neural-network-zoo_VAEs.ipynb

https://tinyurl.com/2fx8smzk

https://github.com/GageDeZoort/neural-network-zoo/tree/main