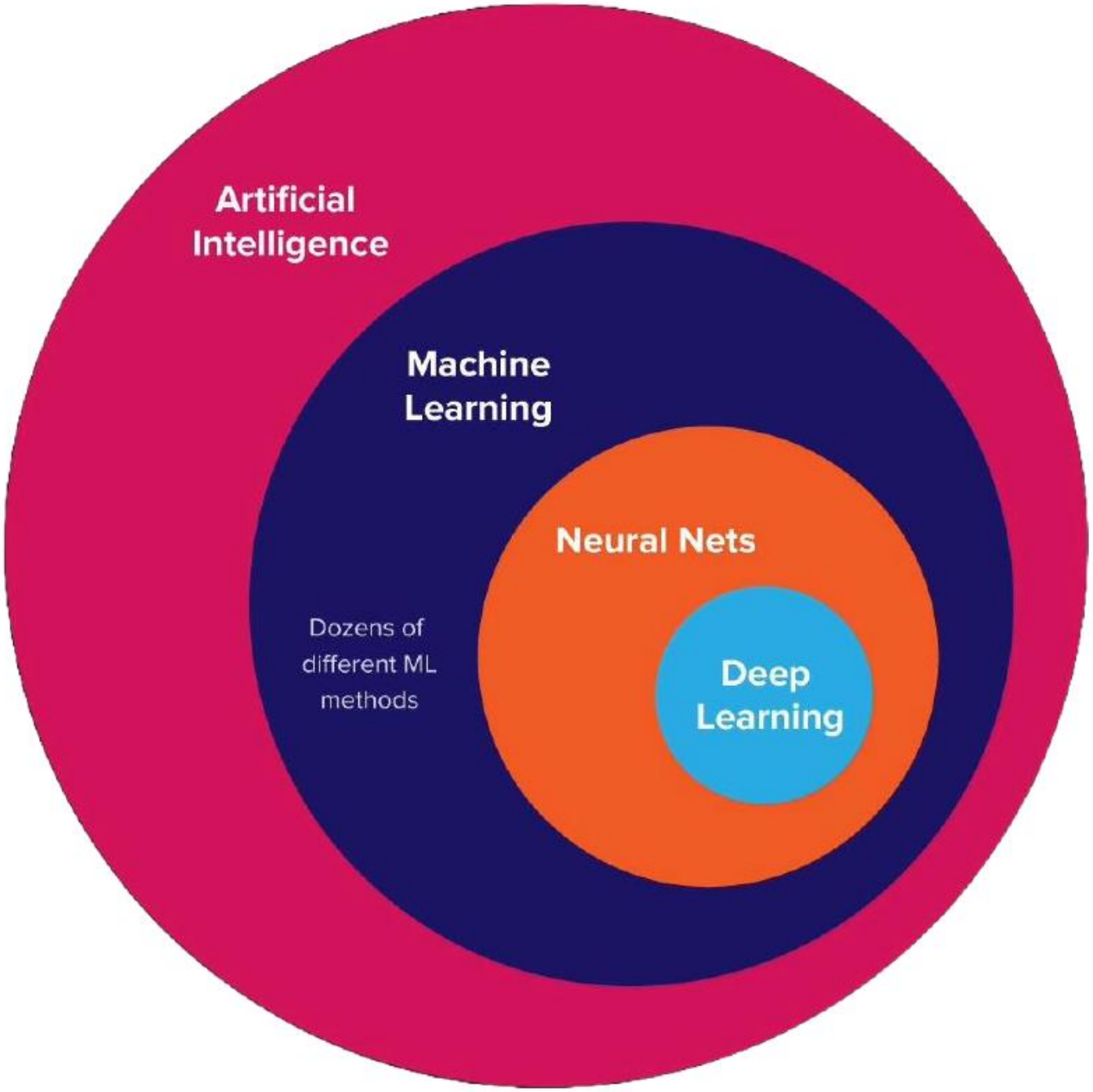
INTRODUCTION TO DEEP LEARNING

15.S60 IAP

16 JANUARY 2025

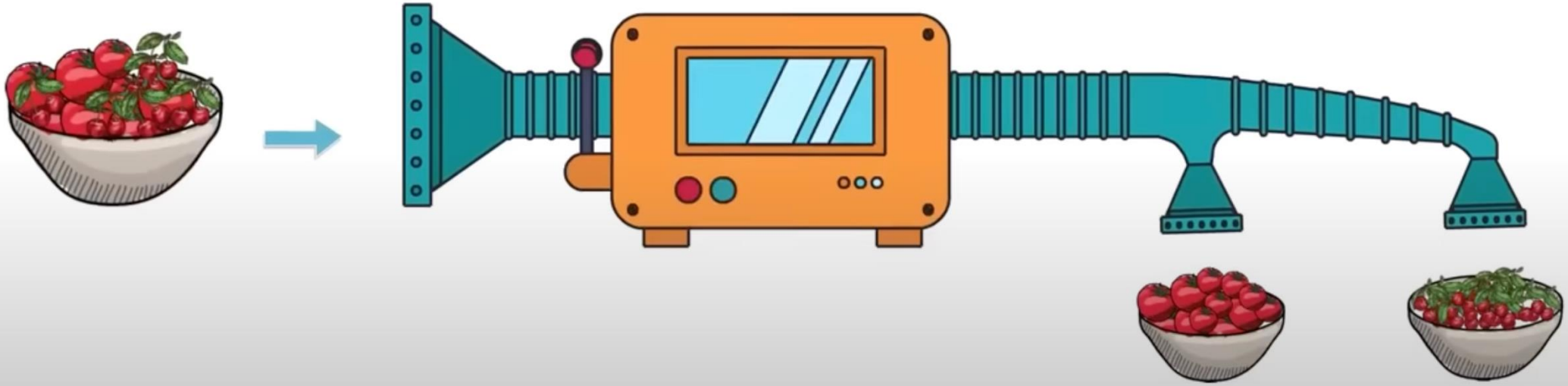


OUTLINE

- Deep learning basics
- When and why can we generalize
- How deep networks represent data & what is a good representation?
- Popular learning paradigms

Features		
	Tomato	Cherry
Size		
Type of Stem		

Deep Learning



WHY DEEP LEARNING?

- Learning directly from data instead of feature engineering
- Scalable: big data → improved performance
- Flexible architectures leverage the type/structure of data

DEEP LEARNING BASICS

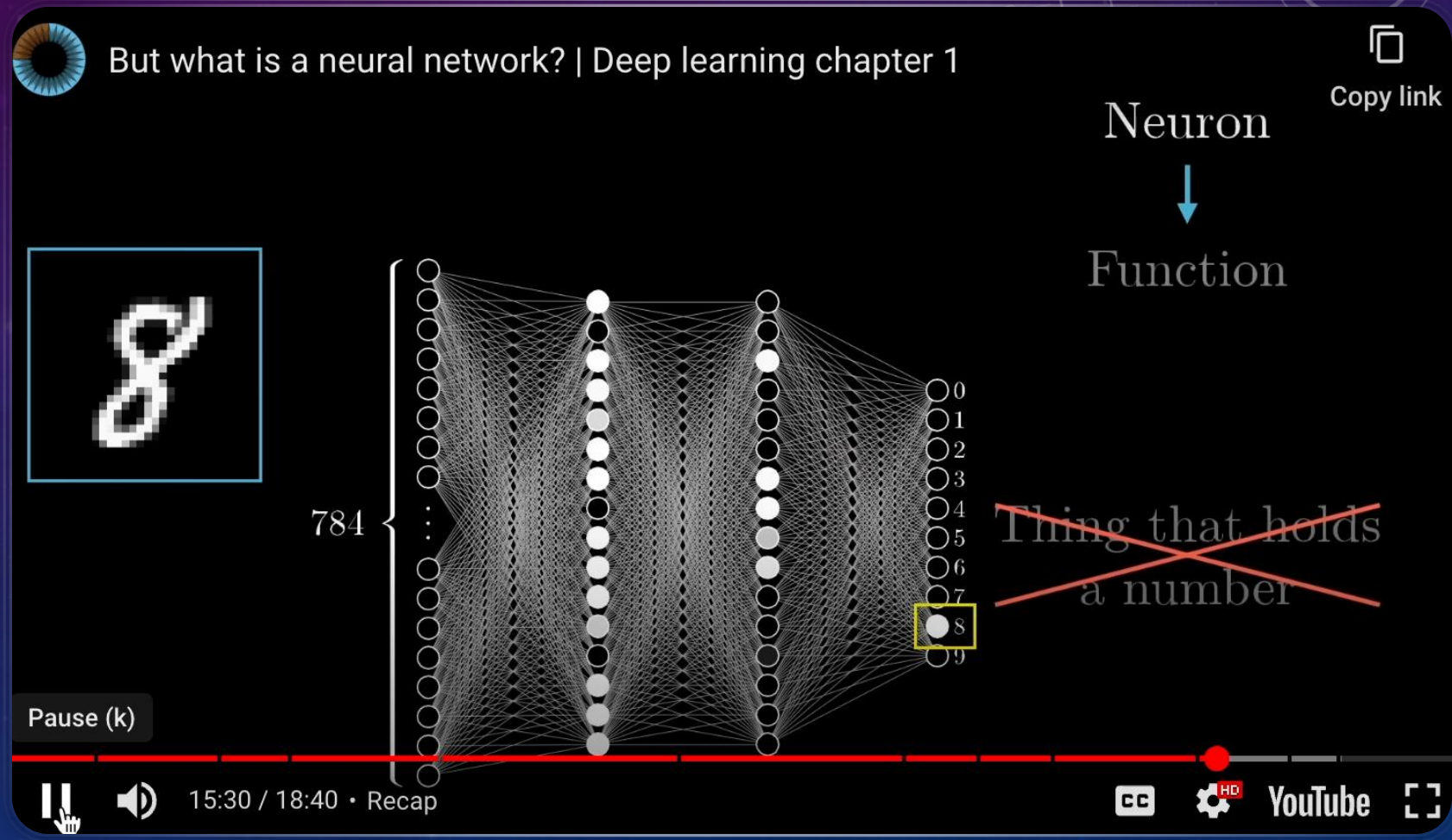
KEY COMPONENTS

A GEOMETRIC VIEW

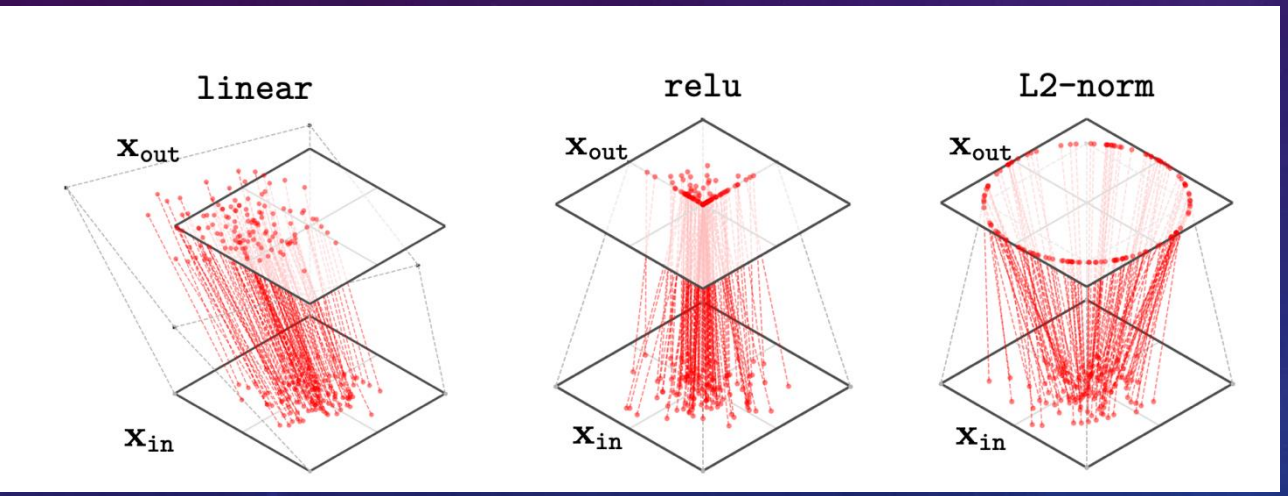
DIFFERENTIAL PROGRAMMING

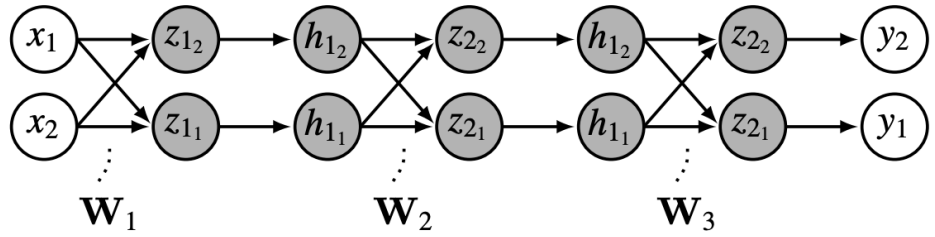
KEY COMPONENTS OF A NEURAL NET

- Neurons & Multilayer Perceptron (MLP)
- https://www.3blue1brown.com/topics/neural-networks
  - Newest upload: LLMs !!



A NEURAL NET TRANSFORMS THE DISTRIBUTION SPACE





Or expressed in math as follows:

$$\mathbf{z}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \quad \triangleleft \texttt{ linear}$$

$$\mathbf{h}_1 = \texttt{relu}(\mathbf{z}_1) \quad \triangleleft \texttt{ relu}$$

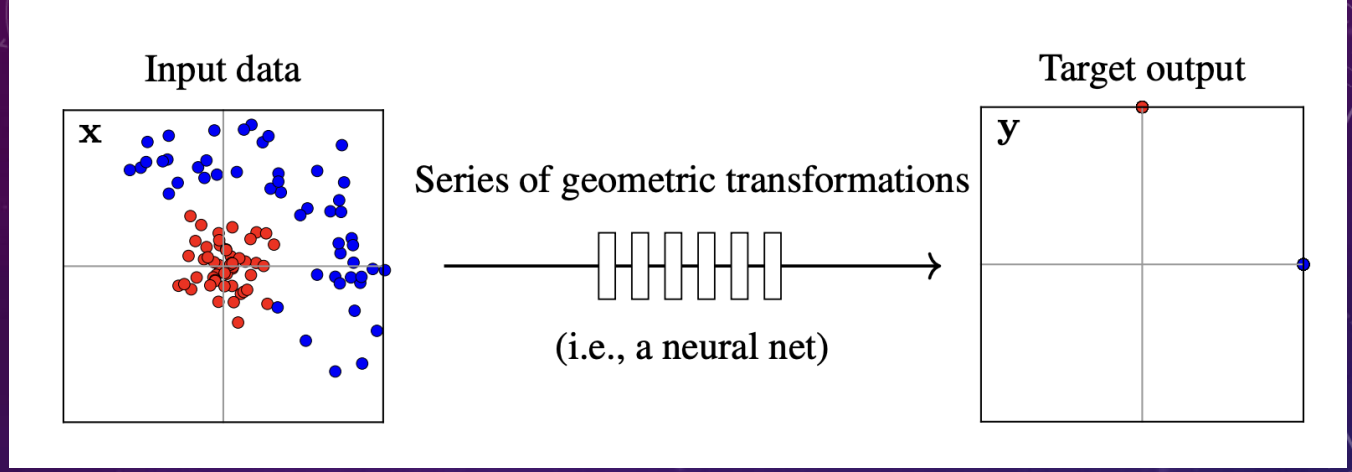
$$\mathbf{z}_2 = \mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2 \quad \triangleleft \texttt{ linear}$$

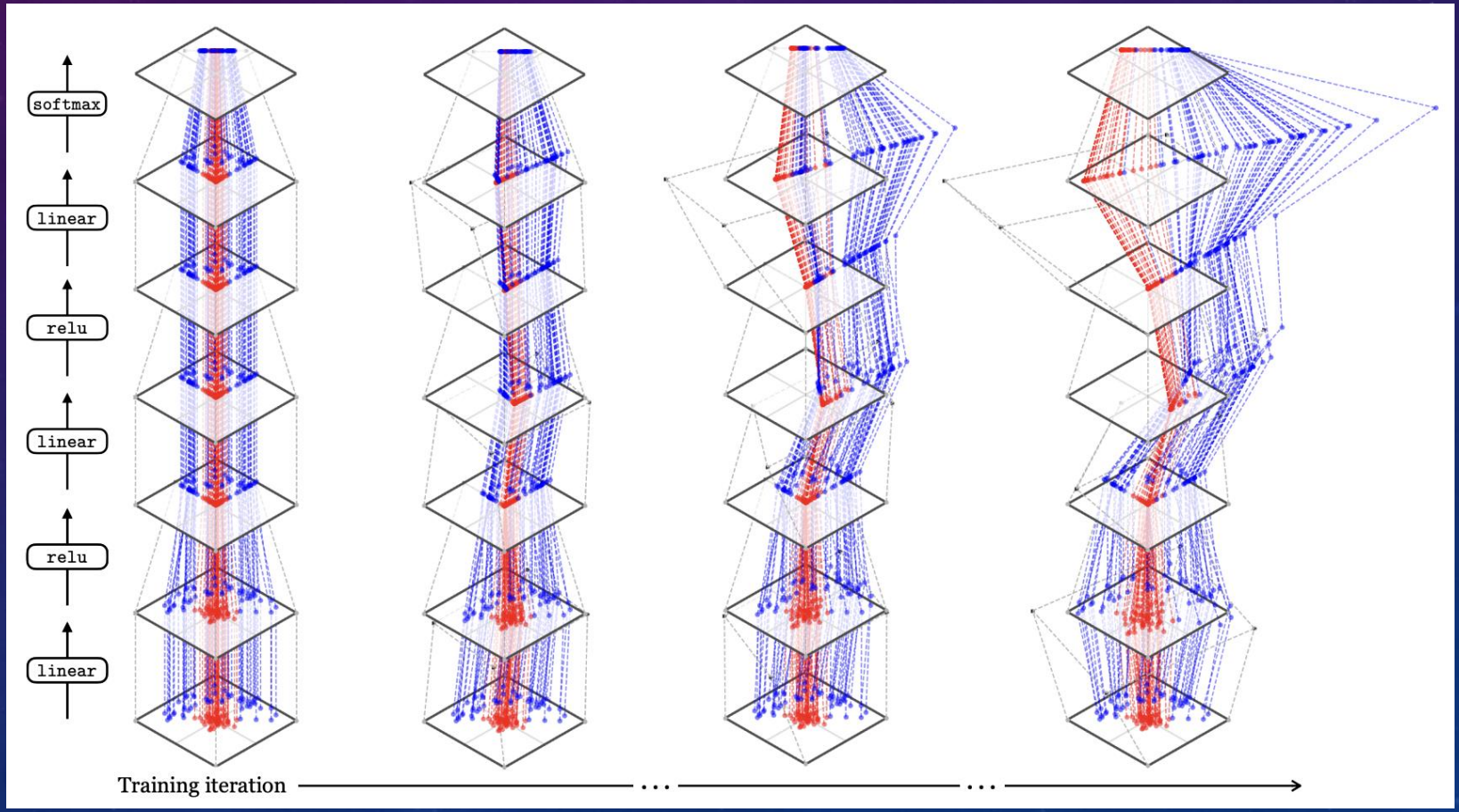
$$\mathbf{h}_2 = \texttt{relu}(\mathbf{z}_2) \quad \triangleleft \texttt{ relu}$$

$$\mathbf{z}_3 = \mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3 \quad \triangleleft \texttt{ linear}$$

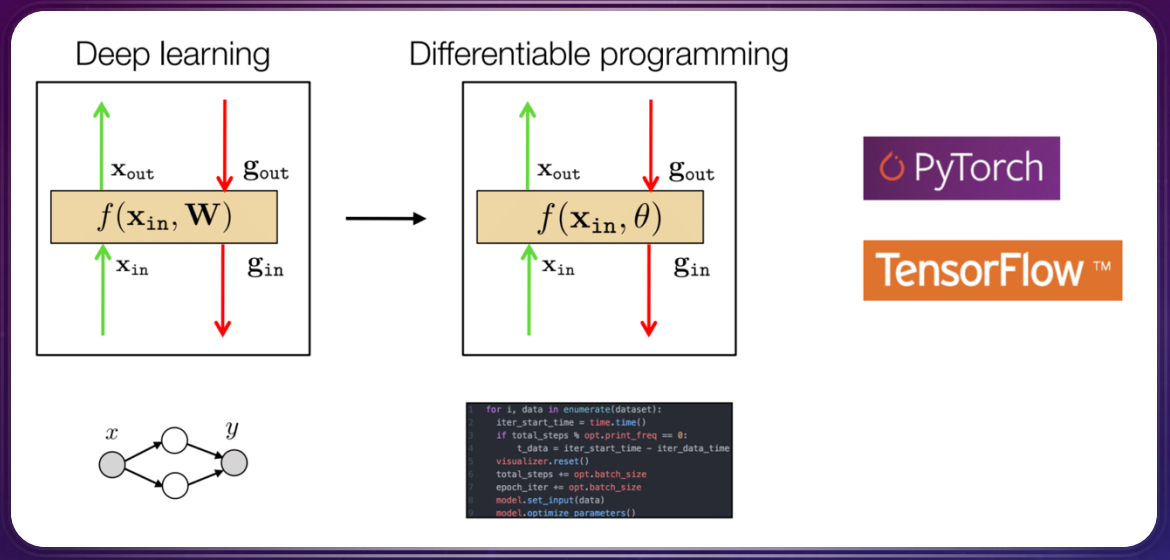
$$\mathbf{y} = \texttt{softmax}(\mathbf{z}_3) \quad \triangleleft \texttt{ softmax}$$

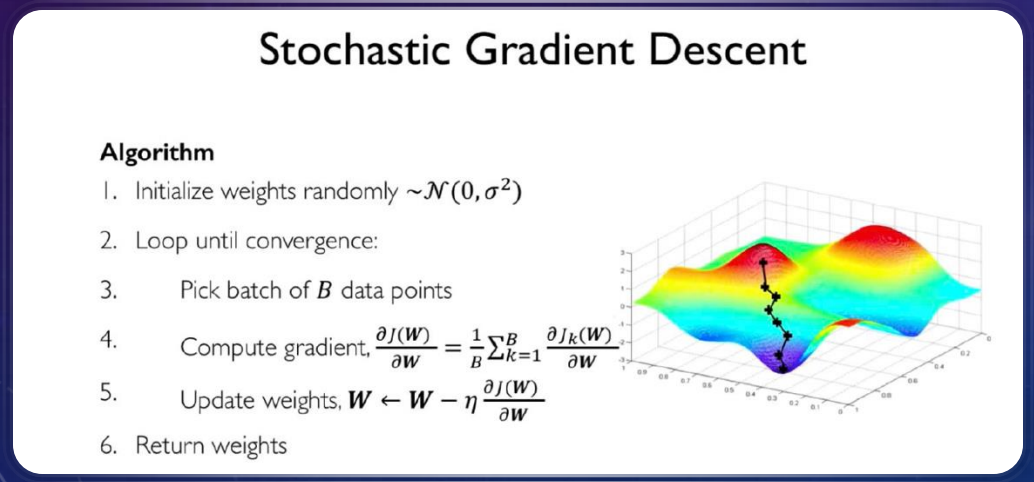
BINARY CLASSIFICATION





HOW TO TRAIN A NEURAL NET





- **Differentiable programs** are programs that rewrite themselves by optimizing along a gradient (automatic differentiation)
- Backpropagation & gradient descent to update parameters / weights

CODE EXAMPLE (CNN, PYTORCH)

$$\mathbf{z}_1[c,:,:] = \mathbf{w}[c,:,:] \star \mathbf{x} + b[c]$$

$$h[c,n,m] = \max(z_1[c,n,m], 0)$$

$$z_2[c] = \frac{1}{NM} \sum_{n,m} h[c,n,m]$$

$$\mathbf{z}_3 = \mathbf{W}\mathbf{z}_2 + \mathbf{c}$$

$$y[k] = \frac{e^{-\tau z_3[k]}}{\sum_{l=1}^{K} e^{-\tau z_3[l]}}$$

```
# first define parameterized layers
conv1 = nn.conv(channels_in=1, channels_out=C, kernel=k, stride=1)
fc1 = nn.fc(dim_in=C, dim_out=K)

# then run data through network
z1 = conv1(x)
h = nn.relu(z1)
z2 = nn.AvgPool2d(h)
z3 = fc1(z2)
y = nn.softmax(z3)
```

PYTORCH VS TENSORFLOW

```
model = VanillaNN(input_size, hidden_size, output_size)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Dummy data and training loop
inputs = torch.randn(16, input_size)  # Batch size of 16
targets = torch.randint(0, output_size, (16,))  # Random class labels

for epoch in range(5):  # 5 epochs
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    loss.backward()
    optimizer.step()
    print(f'Epoch {epoch+1}, Loss: {loss.item()}')
```

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Define the model
model = models.Sequential([
    layers.Dense(10, activation='relu', input_shape=(4,)),
    layers.Dense(3, activation='softmax')
])

# Model, loss, optimizer
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Dummy data and training loop
inputs = tf.random.normal((16, 4))  # Batch size of 16
targets = tf.random.uniform((16,), maxval=3, dtype=tf.int32)

# Fit the model
model.fit(inputs, targets, epochs=5, verbose=1)
```

A GLIMPSE INTO DEEP LEARNING THEORY

APPROXIMATION, OPTIMIZATION, GENERALIZATION

GOAL OF DL: ACHIEVE LOW TEST ERROR

3 parts of achieving low test error / risk / loss function

- How well will our trained neural network do on new data?
- We minimize the *empirical risk*: $\widehat{\mathcal{R}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$

Optimization: how well are we minimizing $\widehat{\mathcal{R}}(\theta)$?

GOAL OF DL: ACHIEVE LOW TEST ERROR

3 parts of the test error (supervised learning)

- We actually want the *population risk (test error)* to be small:

$$\mathcal{R}(\theta) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{P}}\, \mathcal{L}(f_\theta(\mathbf{x}), \mathbf{y})$$

Approximation: what is the best $\mathscr{R}(\theta^*)$ we can achieve with our model?

GOAL OF DL: ACHIEVE LOW TEST ERROR

3 parts of the test error (supervised learning)

- How well will our trained neural network do on new data?
- We minimize the *empirical risk*: $\widehat{\mathcal{R}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$
- We actually want the *population risk (test error)* to be small:

$$\mathcal{R}(\theta) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{P}}\,\mathcal{L}(f_\theta(\mathbf{x}), \mathbf{y})$$

Important questions:

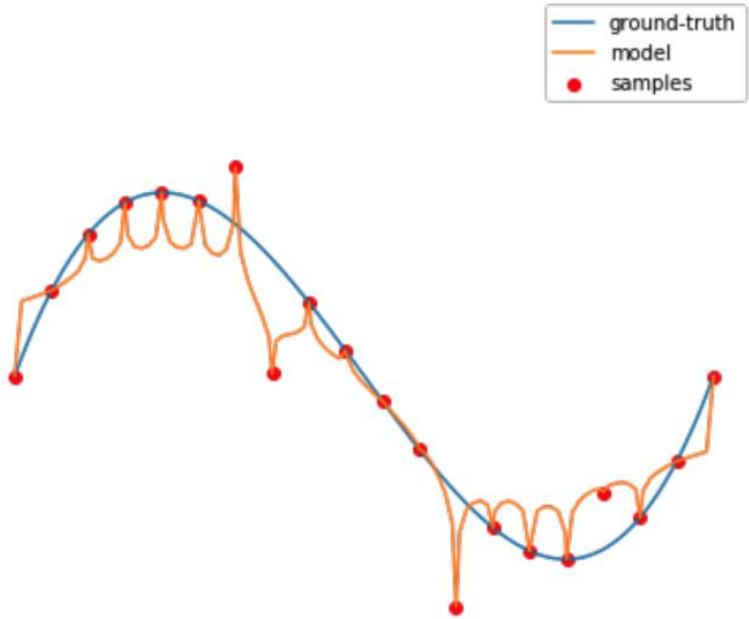
Approximation: what is the best $\mathscr{R}(\theta^*)$ we can achieve with our model?

Optimization: how well are we minimizing $\widehat{\mathscr{R}}(\theta)$?

Generalization: how different is $\widehat{\mathscr{R}}(\theta)$ from $\mathscr{R}(\theta)$?

GENERALIZATION & THE PROBLEM OF OVERFITTING

PREVENTS GENERALIZATION TO UNSEEN DATA

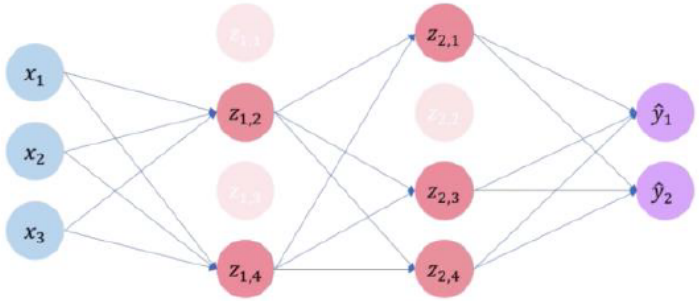


REGULARIZATION IN DEEP NETS

Regularization 1: Dropout

- During training, randomly set some activations to 0
  - Typically 'drop' 50% of activations in layer
  - Forces network to not rely on any 1 node

```
tf.keras.layers.Dropout(p=0.5)
```



Regularization 2: Early Stopping

- Stop training before we have a chance to overfit

Regularization 3: L1 or L2 penalty on the weights

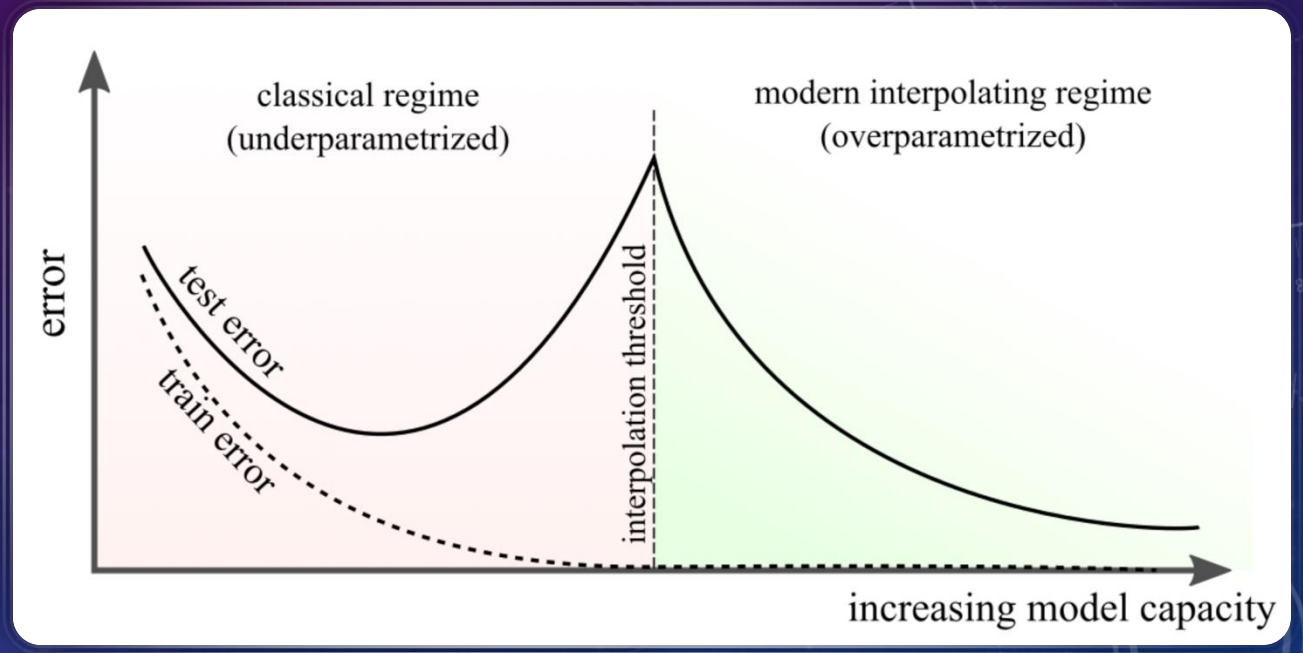
WHY DO DEEP NETS GENERALIZE?

- Deep nets violate classical generalization theory?

"The simplest model that fits the data will generalize best." (Occam's Razor)

→ Open question: *How to measure model complexity in deep learning?*

- infinite-width duality with kernel regression seem to explain the generalization of sufficiently wide NNs



UNDERSTANDING DEEP LEARNING REQUIRES RETHINKING GENERALIZATION

Chiyuan Zhang*
Massachusetts Institute of Technology
chiyuan@mit.edu

Samy Bengio
Google Brain
bengio@google.com

Moritz Hardt
Google Brain
mrtz@google.com

Benjamin Recht†
University of California, Berkeley
brecht@berkeley.edu

Oriol Vinyals
Google DeepMind
vinyals@google.com

ABSTRACT

Despite their massive size, successful deep artificial neural networks can exhibit a remarkably small difference between training and test performance. Conventional wisdom attributes small generalization error either to properties of the model family, or to the regularization techniques used during training.

Through extensive systematic experiments, we show how these traditional approaches fail to explain why large neural networks generalize well in practice. Specifically, our experiments establish that state-of-the-art convolutional networks for image classification trained with stochastic gradient methods easily fit a random labeling of the training data. This phenomenon is qualitatively unaffected by explicit regularization, and occurs even if we replace the true images by completely unstructured random noise. We corroborate these experimental findings with a theoretical construction showing that simple depth two neural networks already have perfect finite sample expressivity as soon as the number of parameters exceeds the number of data points as it usually does in practice.

WHY DO DEEP NETS GENERALIZE?

1. *It is unlikely that the regularizers are the fundamental reason for generalization.*
2. *Deep neural networks easily fit random labels.*
3. *There exists a 2-layer neural network with ReLU activations and 2n+d weights that can represent **any** function on a sample of size n in d dimensions.*

Zhang, 2017

GENERALIZATION REQUIRES INDUCTIVE BIASES

Deep nets generalize. They can make reasonable predictions on inputs they have never seen during training.

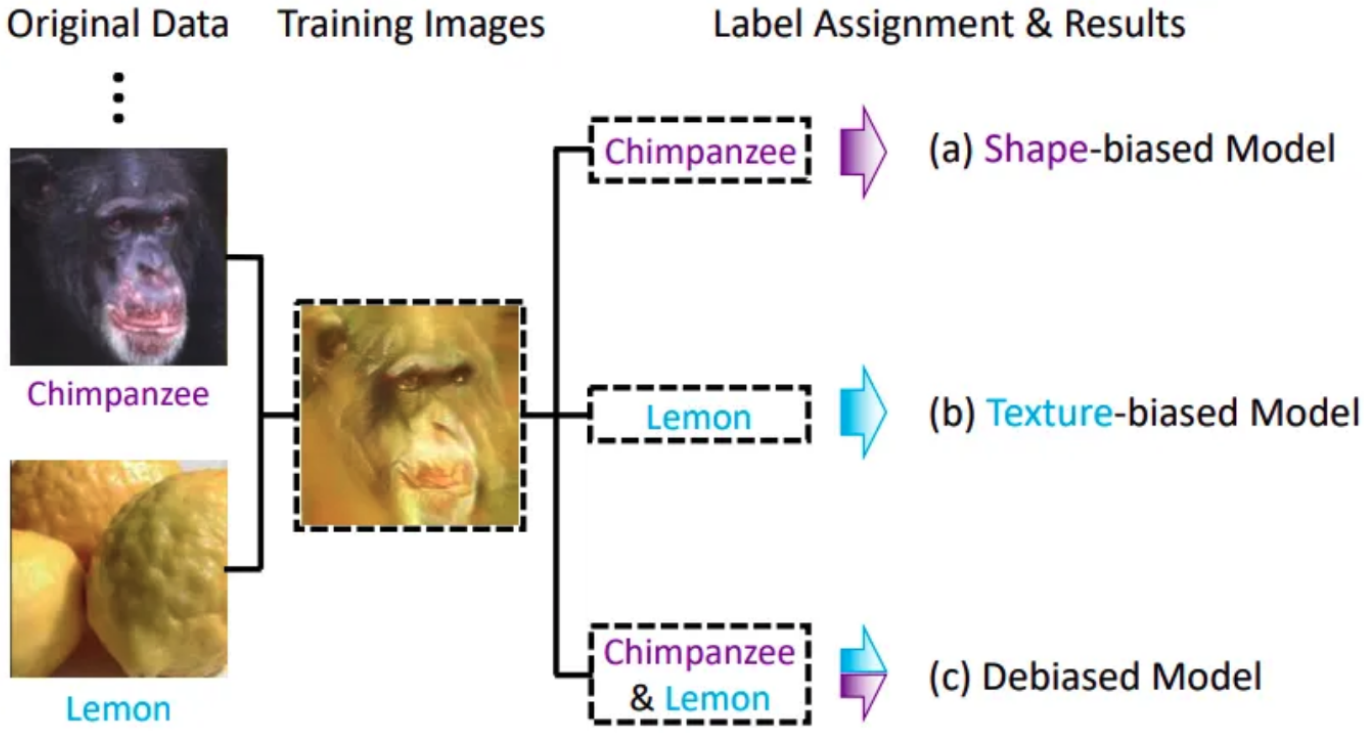
Generalization requires *inductive biases*. Can't be explained by just fitting the training data (we have to rule out the filing cabinet!).

These inductive biases can't just be about classical notions of complexity (# parameters, VC-dimension, etc don't work).

Therefore, deep learning must have some nice inductive biases that control complexity in ways we don't fully know how to characterize!

Next: **what are they?**

INDUCTIVE BIAS

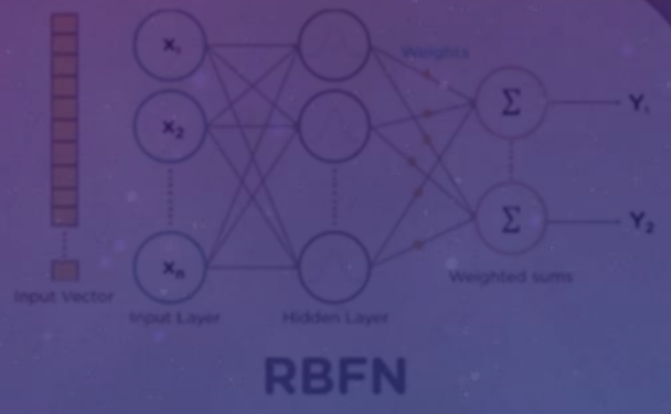


Texture-shape conflicting images, [source].

- = set of assumptions, constraints, or prior knowledge, guiding learning algorithm to favour certain hypotheses over others

WHY DO DEEP NETS GENERALIZE?
→WHAT ARE THEIR INDUCTIVE BIASES?

- Simplicity biases (Huh, Isola et al. TMLR 2023, Pérez ICLR 2019)
  - parameter-function map of many DNNs exponentially biased towards simple functions



LEARNING PARADIGMS

AUTOENCODERS

CONTRASTIVE LEARNING

ADVERSARIAL LEARNING

REPRESENTATION LEARNING (BENGIO ET AL., 2014)

Why are Deep Nets deep?

1. deep architectures promote the **re-use of features**, and
2. deep architectures lead to **more abstract features** at higher layers of representations (more removed from the data).

Open Question: *What are good criteria for learning representations?*

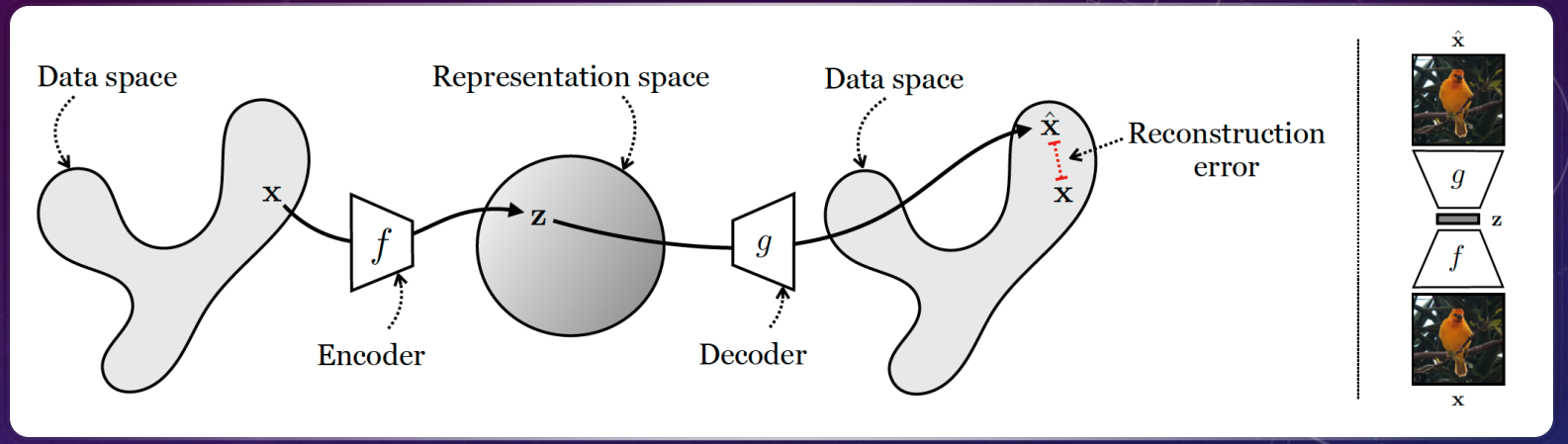
→ Properties of a "good" representation:

1. Distributed
2. Depth & abstraction
3. Disentangling underlying explanatory factors

REPRESENTATION LEARNING PARADIGMS

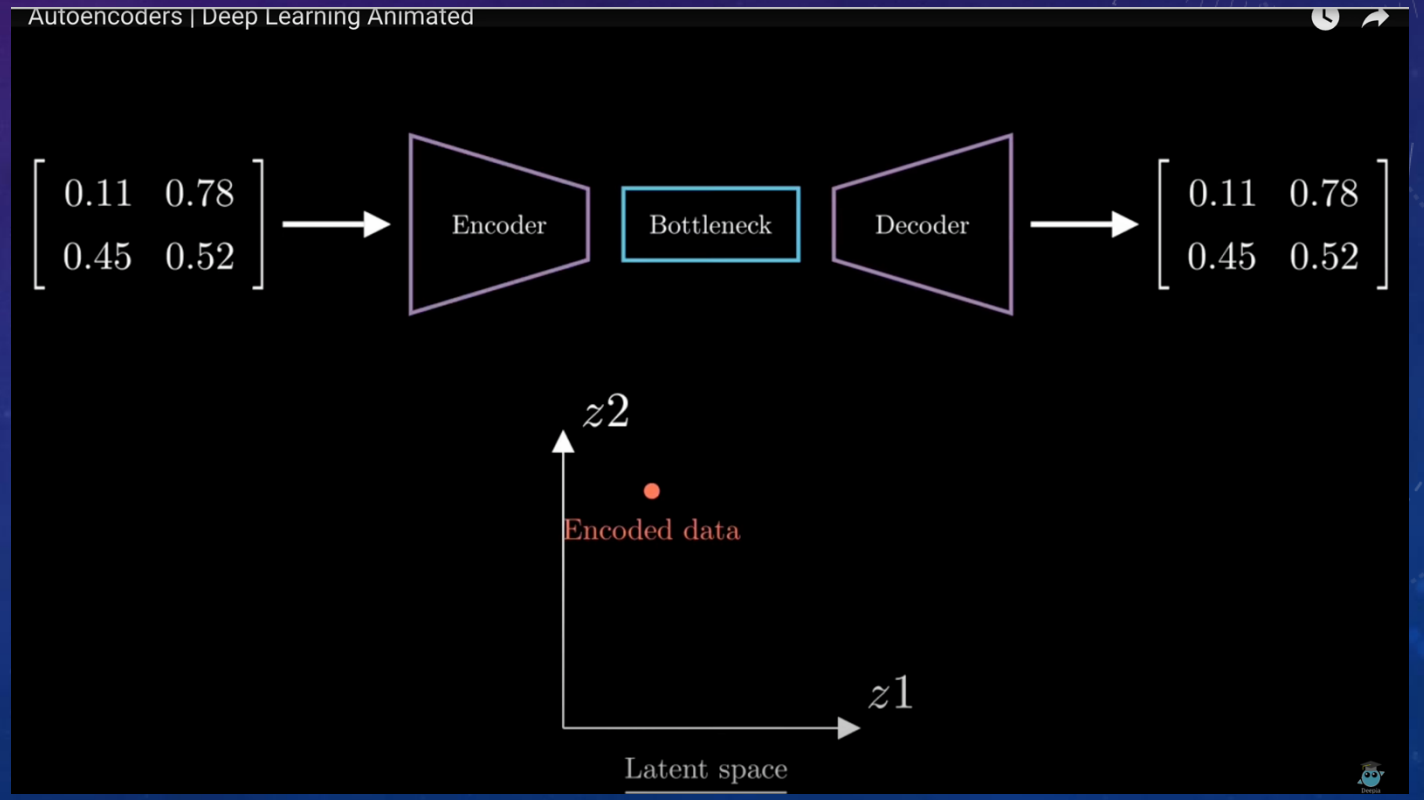
Learning Method	Learning Principle	Short Summary
Autoencoding	Compression	Remove redundant information
Contrastive	Compression	Achieve invariance to viewing transformations
Clustering	Compression	Quantize continuous data into discrete categories
Future prediction	Prediction	Predict the future
Imputation	Prediction	Predict missing data
Pretext tasks	Prediction	Predict abstract properties of your data

AUTOENCODERS

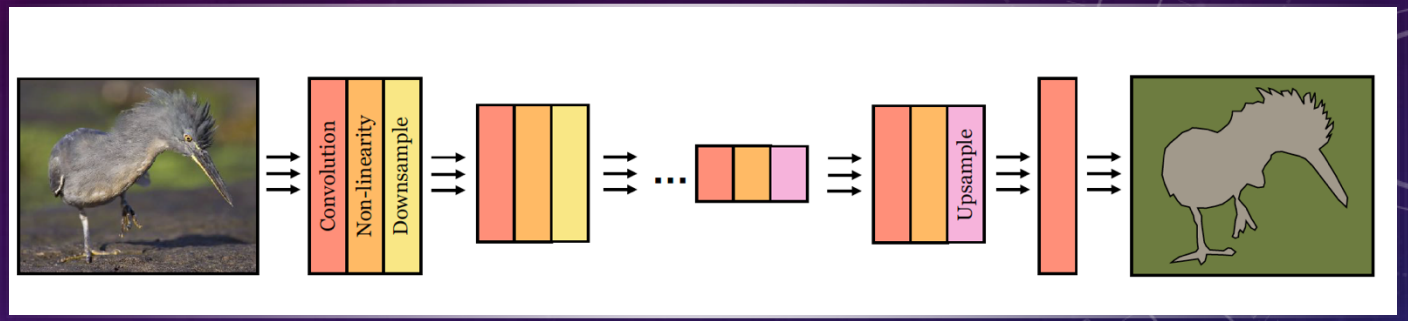


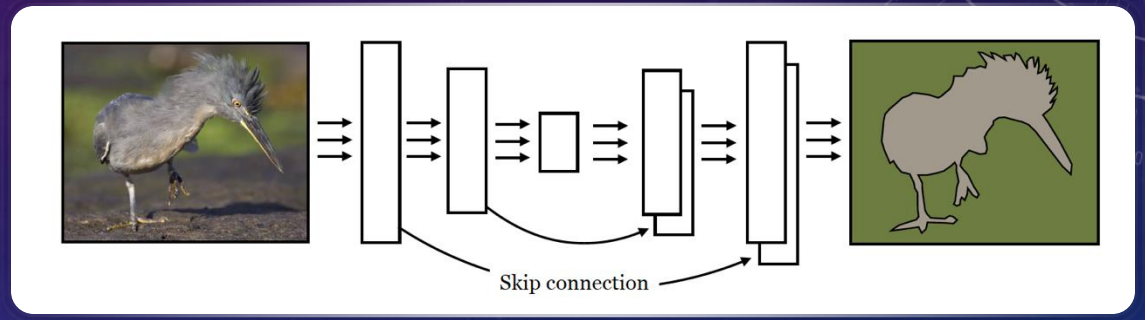
- Unsupervised learning
- map data back to itself (“auto”) via a low-dimensional representational **bottleneck**
- f and g : deep neural nets
- Encoder-Decoder Basics (Deepia)

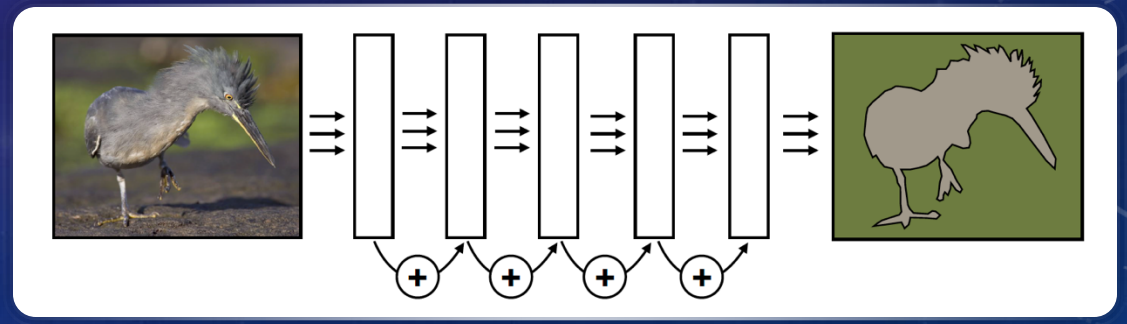
$$f^*, g^* = \arg\min_{f,g} \mathbb{E}_{\mathbf{x}} \|g(f(\mathbf{x})) - \mathbf{x}\|_2^2$$



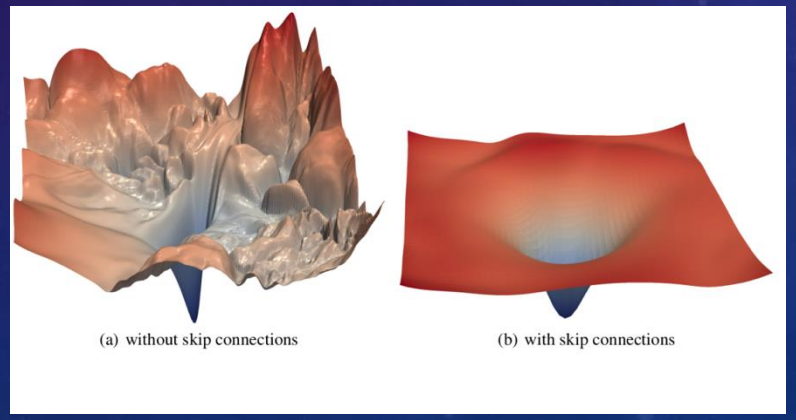
ENCODER-DECODER







- Combine with skip connections
- U-Net
- ResNet



REPRESENTATION LEARNING PARADIGMS

Learning Method	Learning Principle	Short Summary
Autoencoding	Compression	Remove redundant information
Contrastive	Compression	Achieve invariance to viewing transformations
Clustering	Compression	Quantize continuous data into discrete categories
Future prediction	Prediction	Predict the future
Imputation	Prediction	Predict missing data
Pretext tasks	Prediction	Predict abstract properties of your data

Contrastive Learning

Dog

Dog

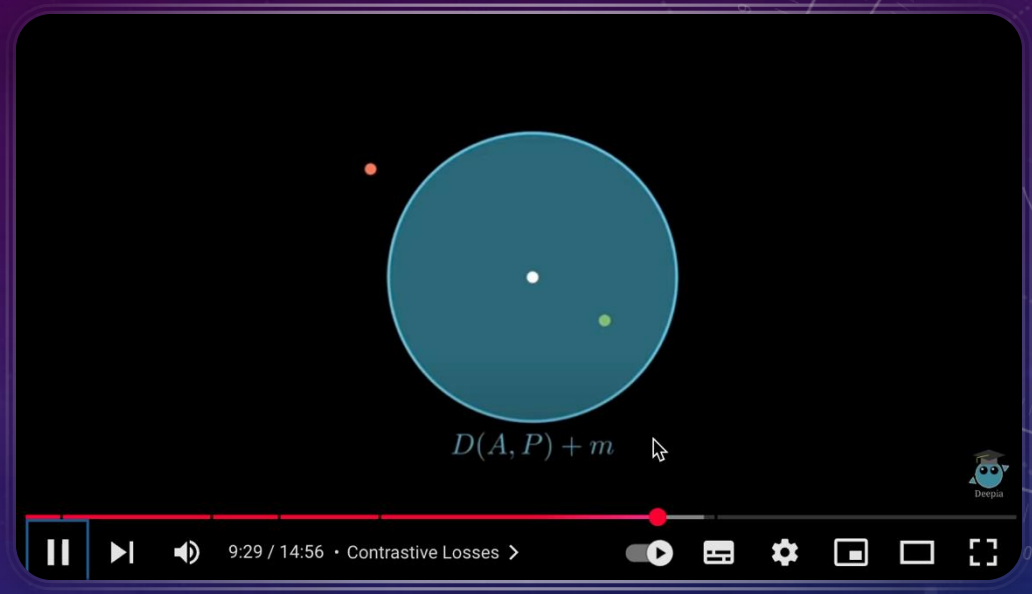
Cat

Latent Space

CONTRASTIVE LEARNING

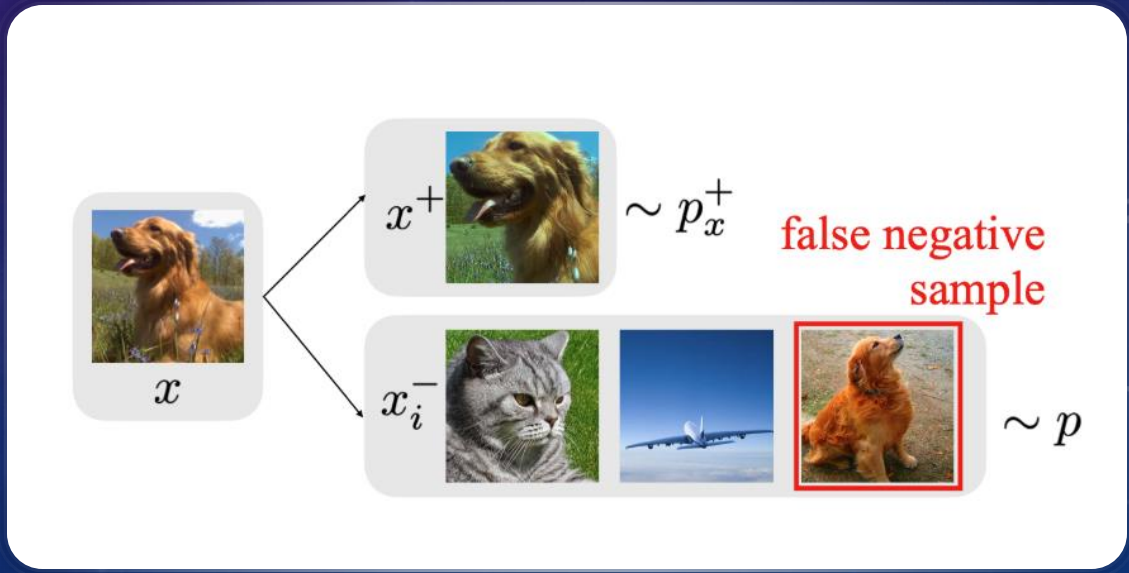
Key Ingredients:

1. Heavy data augmentation
2. Large batch size
3. Hard negative mining



Contrastive Learning with SimCLR | Deep Learning Animated

A more theoretical paper on alignment and uniformity on a hypersphere in contrastive learning: Wang & Isola, 2022



REFERENCES & OTHER HOT TOPICS …

- Variational Autoencoders
- GANs
- Domain Adaptation and Distribution shifts
- Deep Reinforcement Learning
- Distill blog on GNNS
- How Neural Networks Extrapolate: From Feedforward to Graph Neural Networks
- Deep Learning Theory (lecture notes from Telgarsky, University Illinois)

Disclaimer: pictures in this slide deck that do not contain hyperlinks to their sources were retrieved from the 6.7960 Fall 2024 deep learning class (MIT EECS).

HOMEWORK

- Think about your current or intended future project(s)
  - Any DL theory or application ideas that you find interesting to incorporate?
- If dataset available: what are the structure, type, properties of the data you are using? (this can help you define the architectural biases of your DL implementation)
- If theory-heavy: is there any connection to DL theory (approximation theory, generalization or optimization)? (e.g. distribution-based, convergence to some desired property or function, …)
- Modeling: adversarial game?
- Cite at least 1 relevant paper or textbook from the DL field