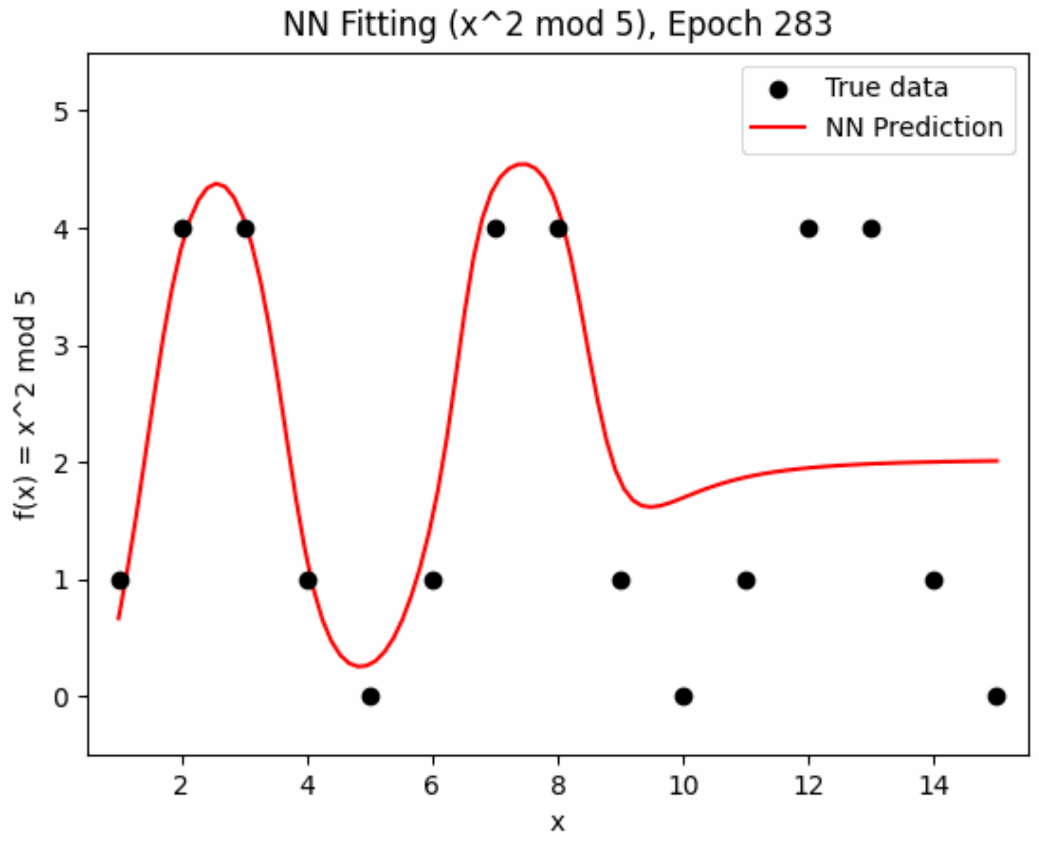
Choosing the right optimizer

Default: start with ‘adam’ (Adaptive Moment Estimation)

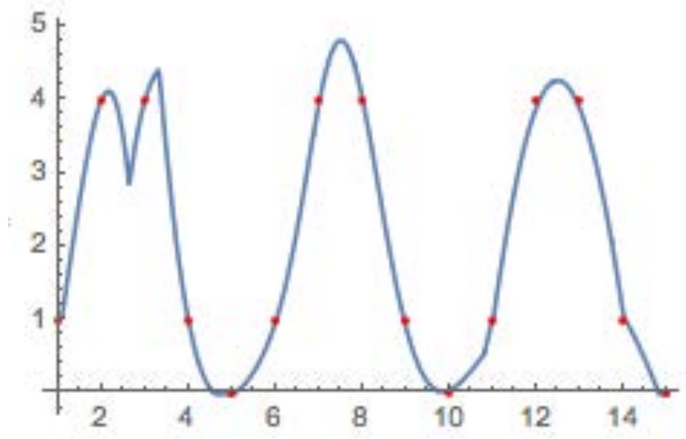
- adam has advantages: adaptive ‘lr’ and momentum
- Use SGD when you need interpretability of optimization trajectory, have specific convergence goals, or want simplicity.
- The trends for SGD, Mini-Batch GD, and Full GD are well-characterized in terms of practical behavior (averaging vs. noise effects), but the underlying science is not fully understood.

Let’s visualize the training process

Animation showing how NN learns predictions over epochs



- for animation, save NN’s predictions at each training epoch.
- ‘Callbacks’ in Keras can do this for you. Add a chunk of code model.fit( ), which Keras calls at certain points.



In what sense is the result (right panel) “correct"? Because it fits the training data - that’s all it knows.

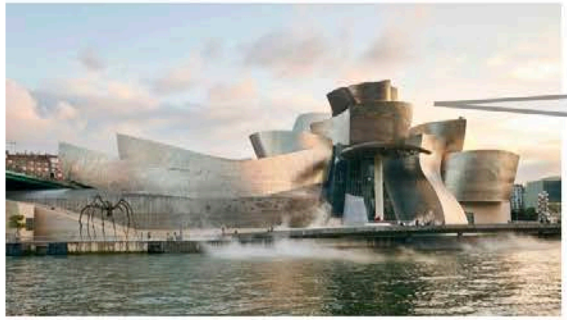
Between points, NN interpolates smoothly, not because it understands but because it's the simplest way to reduce error.

Generalization isn’t guaranteed. Outside training range, DNN behavior is unpredictable. Without more data, there’s no true “right answer,” only guesses.

Training | Validation | Test

Who designed this building?

Training: learning via backprop & weight updates

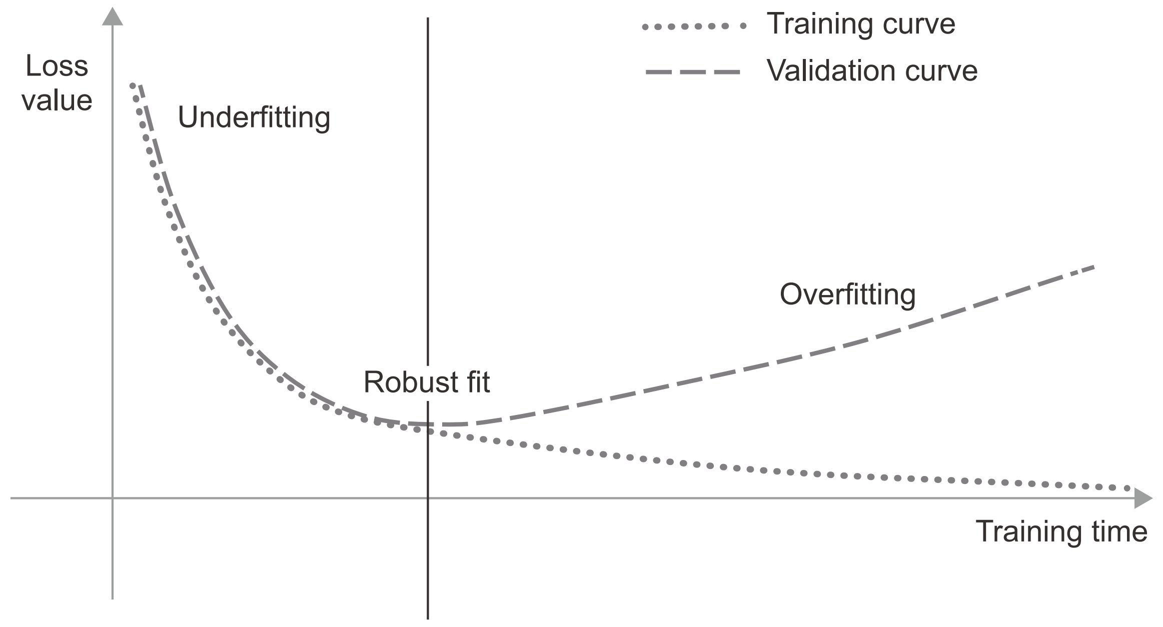


Validation: Fine tune, prevent overfit; weights not updated

Test: real-world evaluation

Underfit, robust fit & overfit

"With four parameters I can fit an elephant, and with five I can make him wiggle his trunk." - Enrico Fermi



Underfitting: The model is too simple to capture patterns —> high bias.

Overfitting: The model is too complex and memorizes training data —> poor generalization.

Robust Fit: A balanced model with optimal generalization with minimal bias and variance.

Quiz: You are training NN and validation loss starts increasing while the training loss keeps decreasing. What should you do?

A. Increase # of epochs and train longer, hoping that it will generalize better.

B. Reduce the model complexity or add regularization (e.g., dropout).

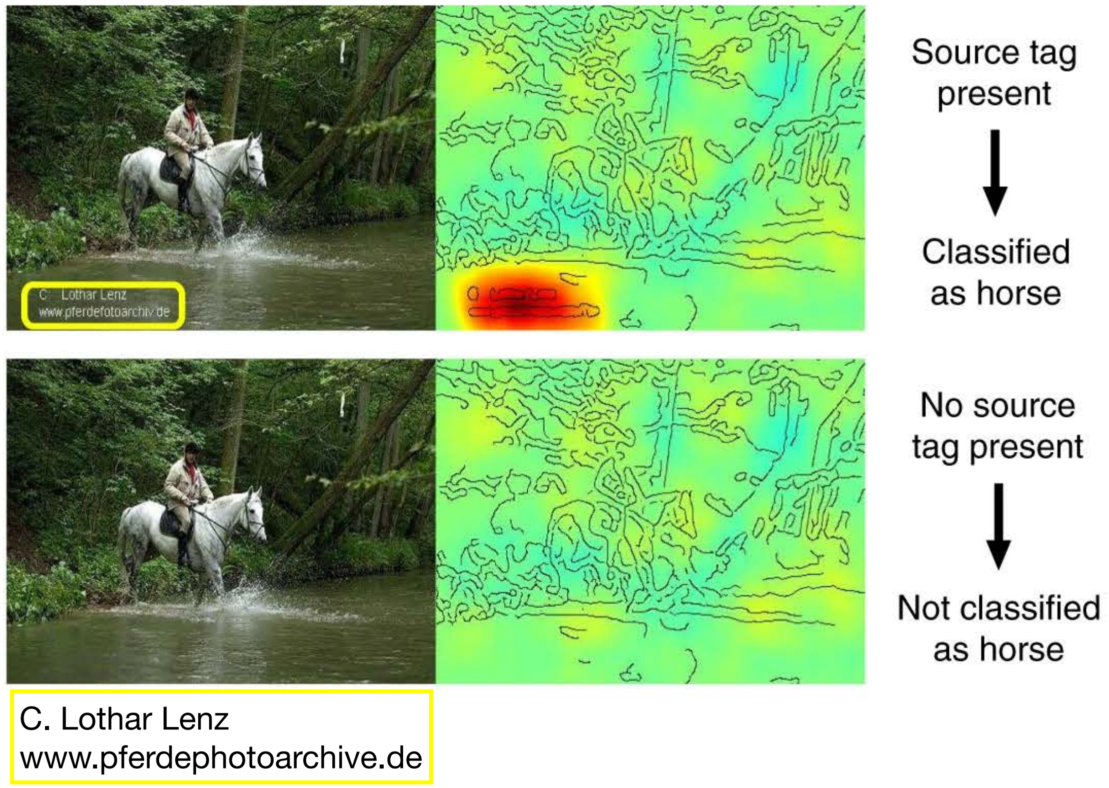
C. Early stop: Go back to an earlier epoch where the validation loss was lowest and use early stopping to select that model.

D. Reduce dataset size to prevent the model from learning too many unnecessary patterns.

Why not just have training vs. test sets?

Why do we need validation set?

Horse-picture from Pascal VOC data set



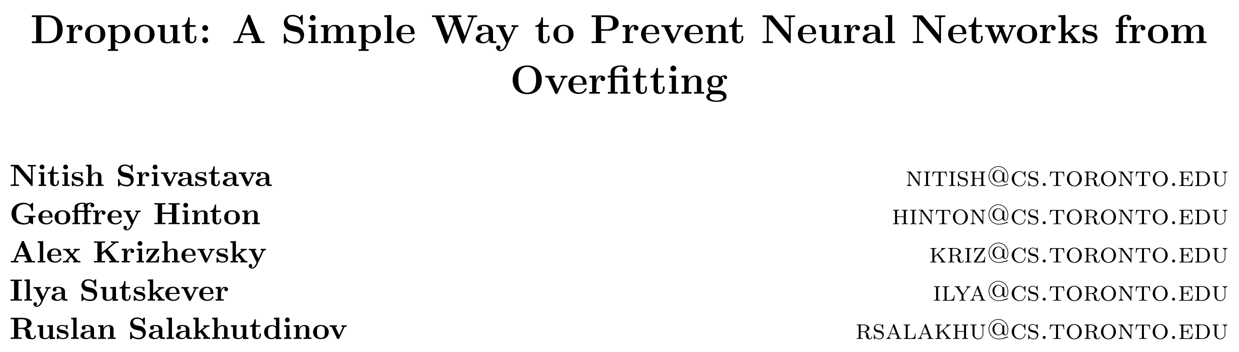
Unmasking Clever Hans predictors and assessing what machines really learn

https://www.nature.com/articles/s41467-019-08987-4

- Hyperparameter tuning (# of neurons, layers, activation func) relies on validation sets.
- Over time, DNN may unknowingly become artificially overfitted to the validation data.
- Example (left): A model trained on Pascal VOC dataset learned to recognize watermarks rather than horses. Well, the model wasn't completely wrong since it found a correlation: "pferde" (horses in German) often appeared in horse images. But this is shortcut learning, not generalization.
- Lesson: We must test on never-seen new data that is separate to ensure true generalization.

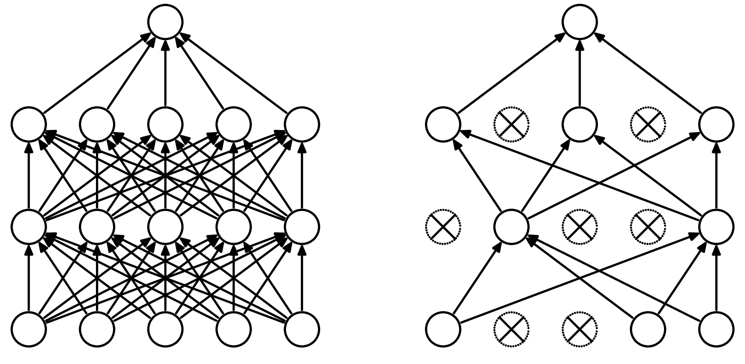
Dropout: Inspired by biology

```
layer = nn.Sequential(nn.Linear(512, 256), nn.ReLU(), nn.Dropout(p=0.2))
```



Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU



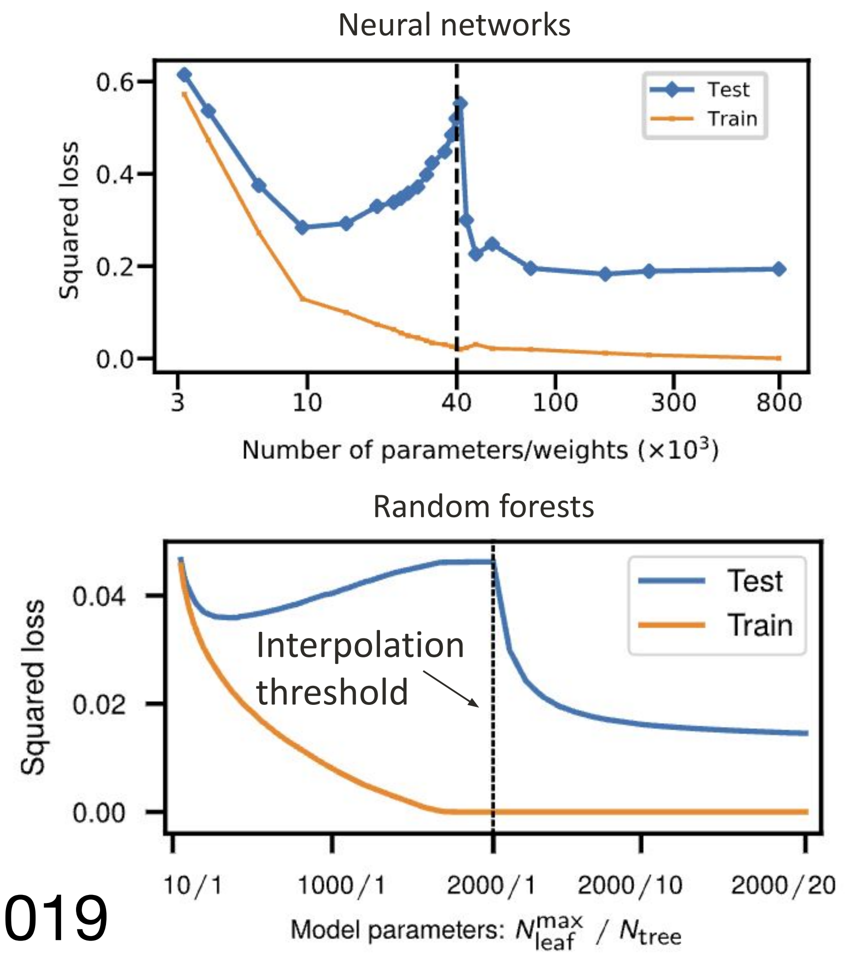
(a) Standard Neural Net (b) After applying dropout.

"A motivation for dropout comes from a theory of the role of sex in evolution."

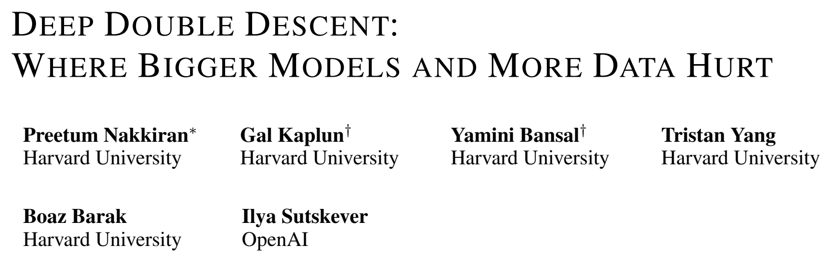
- Hinton's sexual reproduction analogy: Just as genetic crossover prevents the inheritance of harmful mutations, dropout randomly removes neurons during training, forcing NNs to develop redundant, robust features since it can't rely on specific neurons being active.
- Implicit averaging: A network with dropout can be seen as an ensemble of many sub-networks, improving stability and predictive accuracy.
- Practical tip: Use 20–50% dropout in fully connected layers; convolutional layers require less due to their intrinsic weight sharing.

Beyond Overfitting: Double Descent

- Interpolation threshold & Generalization: When model capacity exceeds a critical point, test error initially increases due to memorization but later decreases, improving generalization beyond overfitting regime.

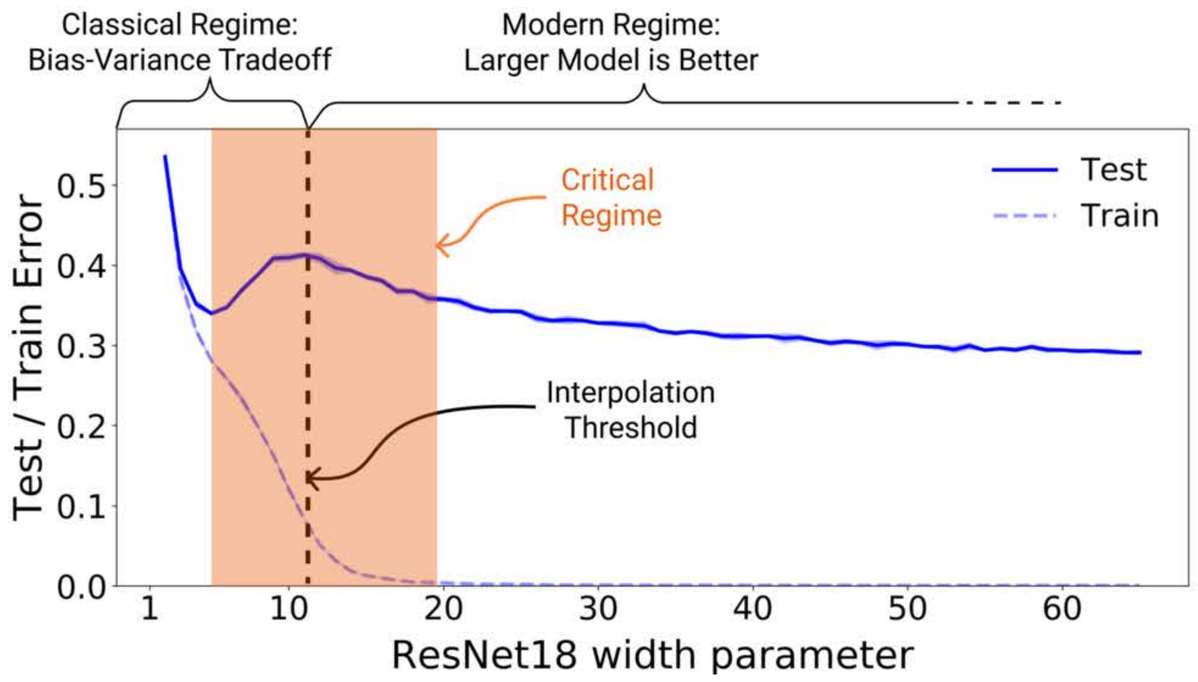


Belkin 2019



Deep Double Descent:
Where Bigger Models and More Data Hurt

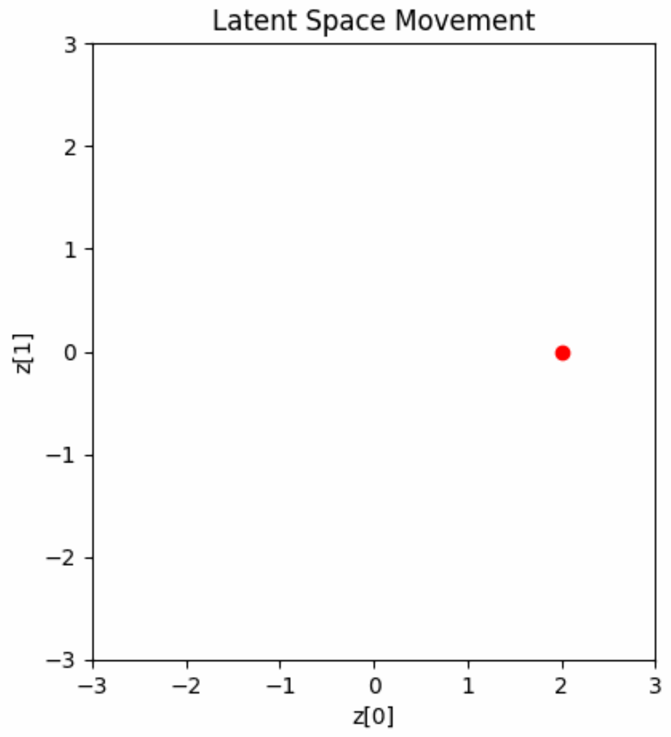
Preetum Nakkiran* Harvard University
Gal Kaplun† Harvard University
Yamini Bansal† Harvard University
Tristan Yang Harvard University
Boaz Barak Harvard University
Ilya Sutskever OpenAI

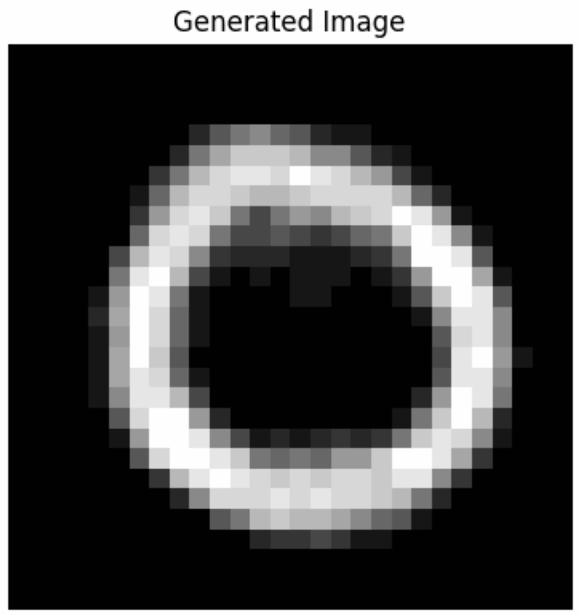


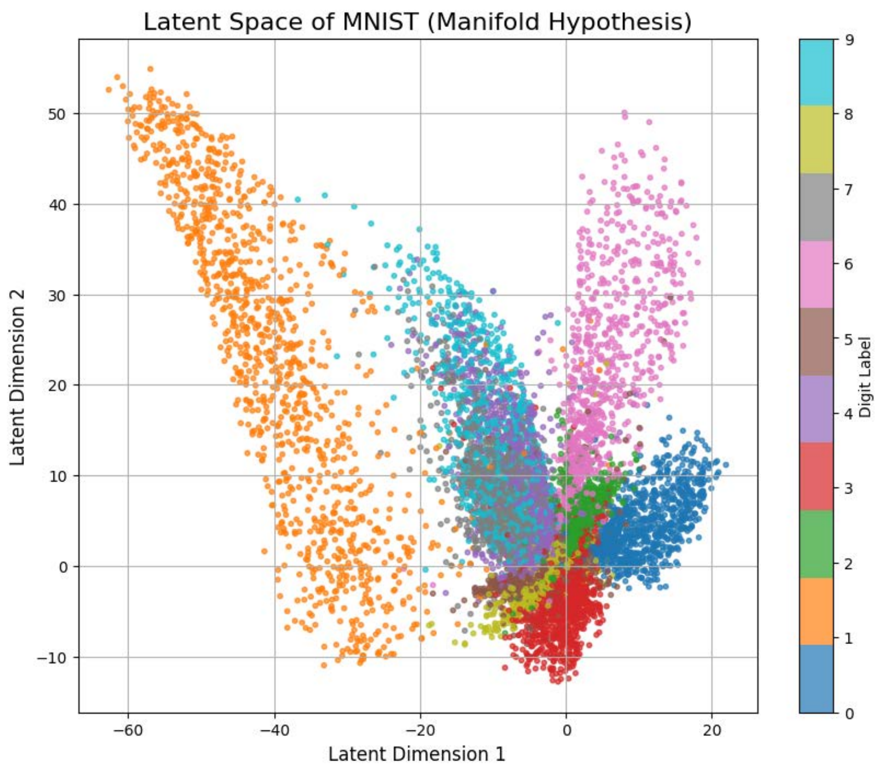
Manifold Hypothesis again: Why does deep learning work?

Why does generalization work? Is it just interpolation?

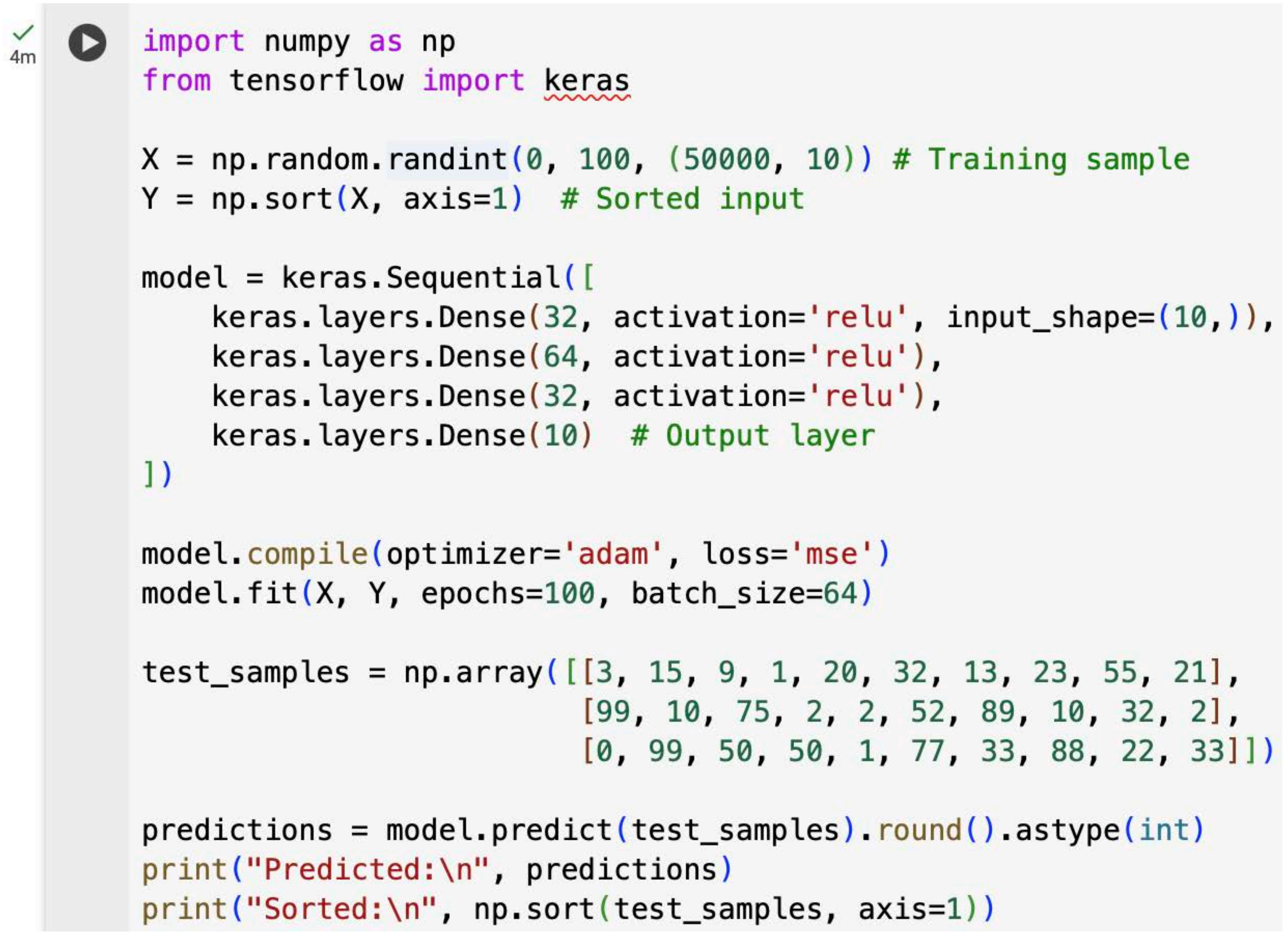
- High-dimensional data, like images and text, often lie on smooth and differentiable manifolds, making them amenable to deep learning. Neural nets exploit this structure to reduce complexity, capturing meaningful variations rather than memorizing raw data. This shows up in Variational Autoencoders (VAEs), which map inputs onto a continuous, low-dimensional latent space, enabling smooth transformations.







Can deep learning discover sorting rules?



```python
import numpy as np
from tensorflow import keras

X = np.random.randint(0, 100, (50000, 10)) # Training sample
Y = np.sort(X, axis=1)  # Sorted input

model = keras.Sequential([
    keras.layers.Dense(32, activation='relu', input_shape=(10,)),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(10)  # Output layer
])

model.compile(optimizer='adam', loss='mse')
model.fit(X, Y, epochs=100, batch_size=64)

test_samples = np.array([[3, 15, 9, 1, 20, 32, 13, 23, 55, 21],
                         [99, 10, 75, 2, 2, 52, 89, 10, 32, 2],
                         [0, 99, 50, 50, 1, 77, 33, 88, 22, 33]])

predictions = model.predict(test_samples).round().astype(int)
print("Predicted:\n", predictions)
print("Sorted:\n", np.sort(test_samples, axis=1))
```

General strategy for DNN architecture design

How many hidden layers and neurons? Which activation func?

1. Layer structure: arrange neurons in hidden layers to graduate increase (to capture more complexity for feature extraction) or decrease (toward compression, decision making).

2. Width (neurons per layer): Start small and increase # of neurons as needed. Exssesive neurons => overfit risk.

3. Depth (# of hidden layers): Start with 2-4 layers and scale as needed. Deeper architecture allows more abstract representation (good) but training is more difficult.

4. Activation function: Use ReLU by default to avoid saturation. Tanh or sigmoid are useful in probabilities or bounded outputs, but suffer from vanishing gradients in deep nets.

5. Dropout and regularization: Add dropout (20—50% rate) to prevent overfit;

Occam's razor: the simplest DNN architecture giving the desired result is the best.

End-to-end MNIST

```python
import tensorflow as tf

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Why normalize pixel values?

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),  # Input layer
    tf.keras.layers.Dense(32, activation='relu'),  # Hidden layer
    tf.keras.layers.Dropout(0.2),                  # what’s the purpose of this?
    tf.keras.layers.Dense(10, activation='softmax') # why is softmax used?
])

model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# change batch_size and see how it affects training speed and accuracy
history = model.fit(x_train, y_train, batch_size=128, epochs=20, verbose=1)
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
```