Copyright Notice

Model Resource Management Techniques

DeepLearning.AI

Welcome

DeepLearning.AI

Dimensionality Reduction

Dimensionality Effect on Performance

High-dimensional data

Before. .. when it was all about data mining

- Domain experts selected features
- Designed feature transforms
- Small number of more relevant features were enough

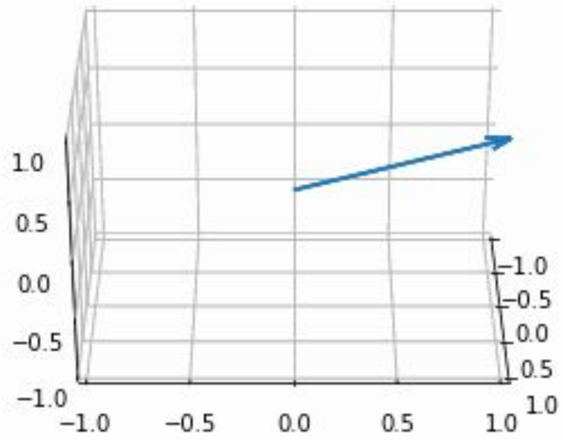
Now … data science is about integrating everything

- Data generation and storage is less of a problem
- Squeeze out the best from data
- More high-dimensional data having more features

A note about neural networks

- Yes, neural networks will perform a kind of automatic feature selection
- However, that’s not as efficient as a well-designed dataset and model
  - Much of the model can be largely “shut off” to ignore unwanted features
  - Even unused parts of the consume space and compute resources
  - Unwanted features can still introduce unwanted noise
  - Each feature requires infrastructure to collect, store, and manage

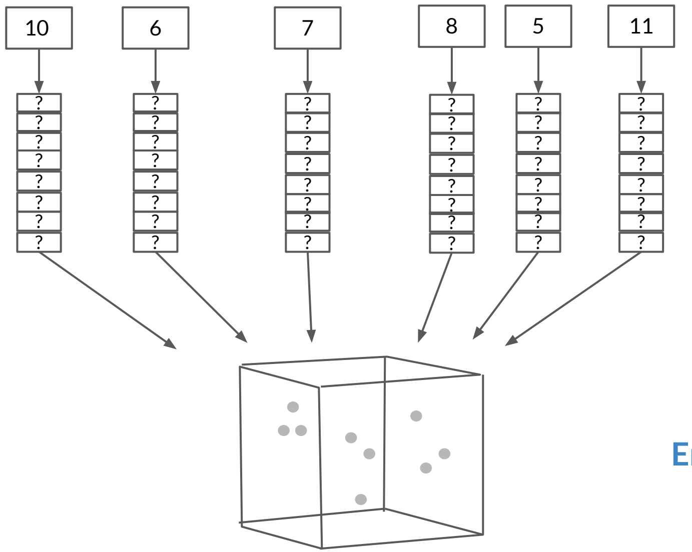
High-dimensional spaces



Word embedding - An example

Auto Embedding Weight Matrix

[“I want to search for blood pressure result history “,
“<u>Show blood pressure result for patient</u>”,...]



Input layer

(Learned vectors)

Embedding Layer

i	1
want	2
to	3
search	4
for	5
blood	6
pressure	7
result	8
history	9
show	10
patient	11
...	
LAST	20

Initialization and loading the dataset

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
from keras.datasets import reuters
from keras.preprocessing import sequence
num_words = 1000

(reuters_train_x, reuters_train_y), (reuters_test_x, reuters_test_y) =
                     tf.keras.datasets.reuters.load_data(num_words=num_words)
n_labels = np.unique(reuters_train_y).shape[0]
```

Further preprocessing

```
from keras.utils import np_utils
reuters_train_y = np_utils.to_categorical(reuters_train_y, 46)
reuters_test_y = np_utils.to_categorical(reuters_test_y, 46)

reuters_train_x =
     tf.keras.preprocessing.sequence.pad_sequences(reuters_train_x, maxlen=20)
reuters_test_x = tf.keras.preprocessing.sequence.pad_sequences(reuters_test_x,
                 maxlen=20)
```

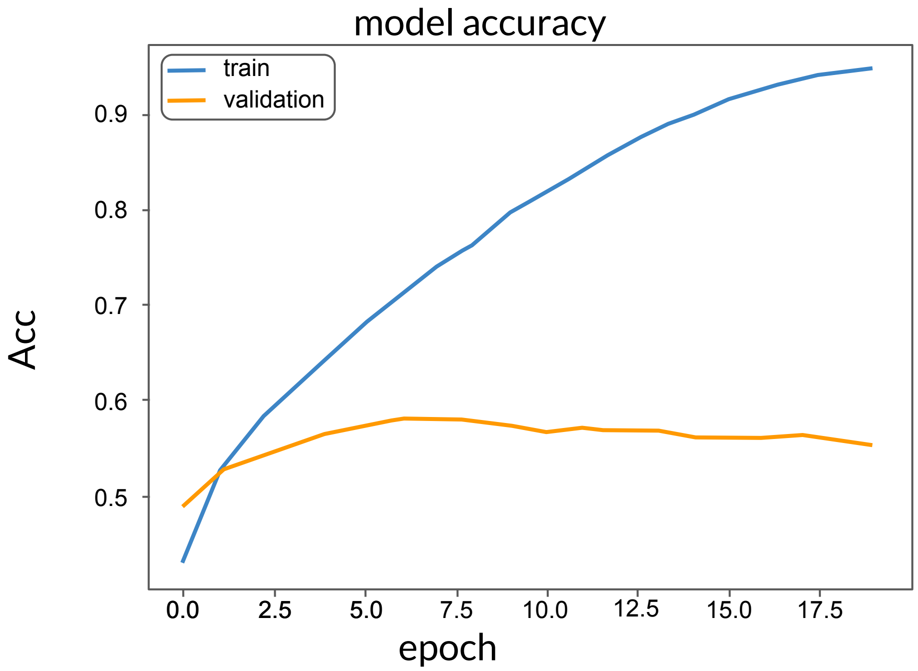
Using all dimensions

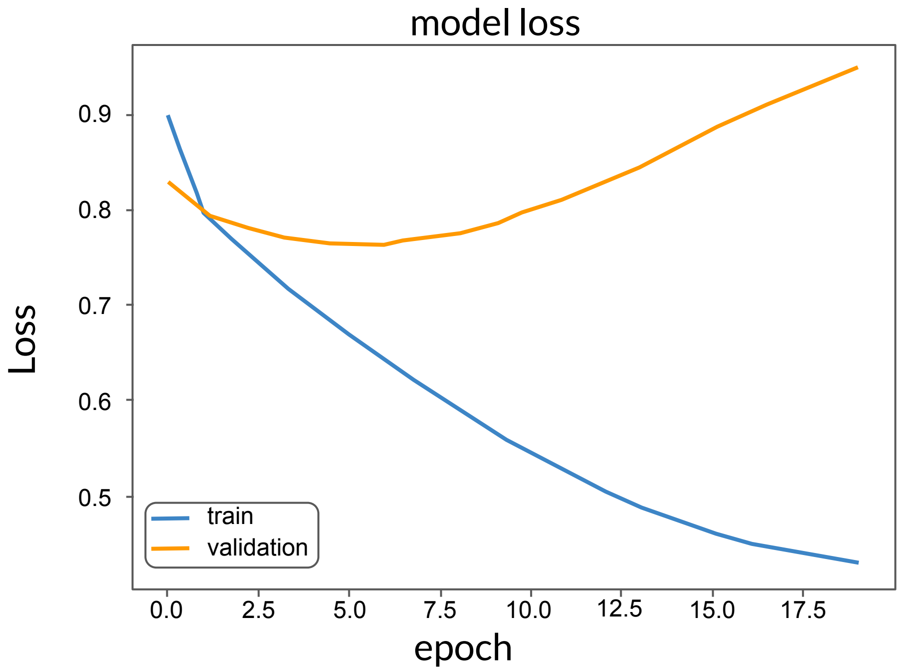
```
from tensorflow.keras import layers
model2 = tf.keras.Sequential(
    [
      layers.Embedding(num_words, 1000, input_length= 20),
      layers.Flatten(),
      layers.Dense(256),
      layers.Dropout(0.25),
      layers.Activation('relu'),
      layers.Dense(46),
      layers.Activation('softmax')
    ])
```

Model compilation and training

```
model.compile(loss="categorical_crossentropy", optimizer="rmsprop",
metrics=['accuracy'])

model_1 = model.fit(reuters_train_x, reuters_train_y,
                    validation_data=(reuters_test_x , reuters_test_y),
                    batch_size=128, epochs=20, verbose=0)
```

Example with a higher number of dimensions

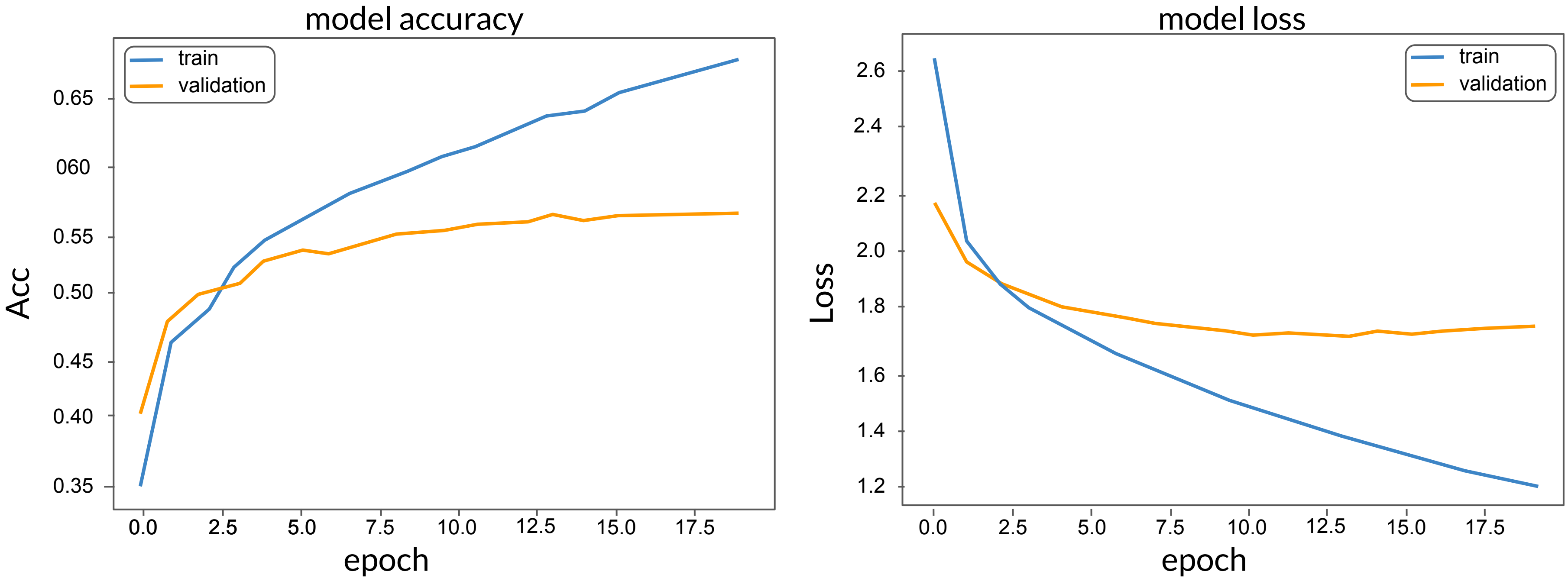




Word embeddings: 6 dimensions

```
from tensorflow.keras import layers
model = tf.keras.Sequential(
    [
      layers.Embedding(num_words, 6, input_length= 20),
      layers.Flatten(),
      layers.Dense(256),
      layers.Dropout(0.25),
      layers.Activation('relu'),
      layers.Dense(46),
      layers.Activation('softmax')
    ])
```

Word embeddings: fourth root of the size of the vocab



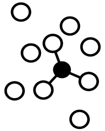
DeepLearning.AI

Dimensionality Reduction

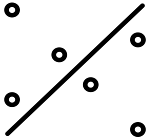
Curse of Dimensionality

Many ML methods use the distance measure

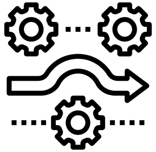
ML methods



k-Nearest Neighbours



Support Vector Machines



Recommendation systems

Distance measure

e.g., Euclidean Distance

$$d_{ij} = \sqrt{\sum_{k=1}^{n} (x_{ik} - x_{jk})^2}$$

Why is high-dimensional data a problem?

- More dimensions → more features
- Risk of overfitting our models
- Distances grow more and more alike
- No clear distinction between clustered objects
- Concentration phenomenon for Euclidean distance

Curse of dimensionality

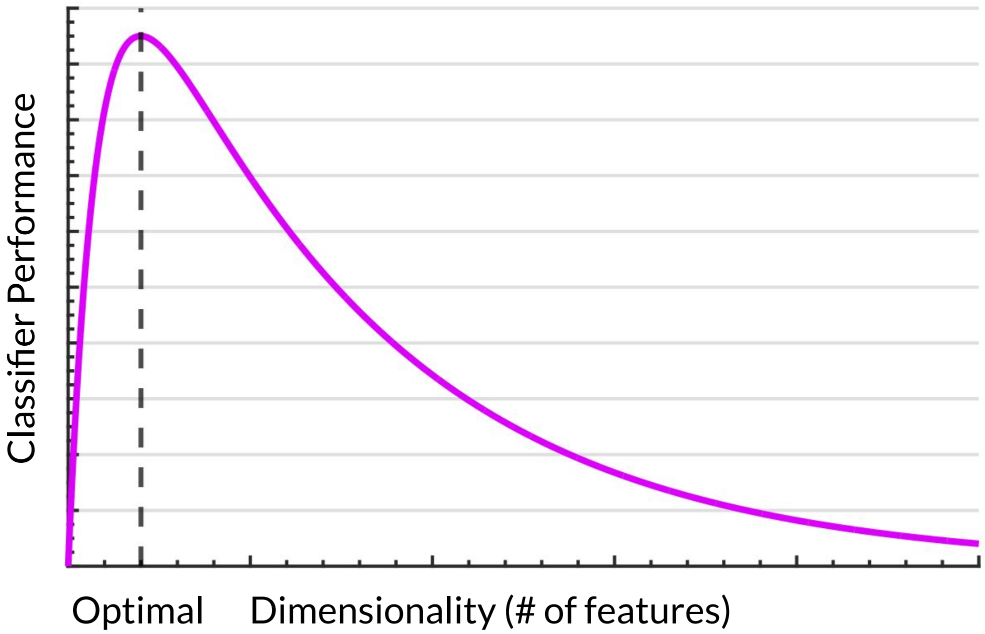
"As we add more dimensions we also increase the processing power we need to train the model and make predictions, as well as the amount of training data required"

Badreesh Shetty

Why are more features bad?

- Redundant / irrelevant features
- More noise added than signal
- Hard to interpret and visualize
- Hard to store and process data

The performance of algorithms ~ the number of dimensions



Adding dimensions increases feature space volume

1-D

1	2	3	4	5

2-D

(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)
(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)
(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)
(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)
(5, 1)	(5, 2)	(5, 3)	(5, 4)	(5, 5)

...

...

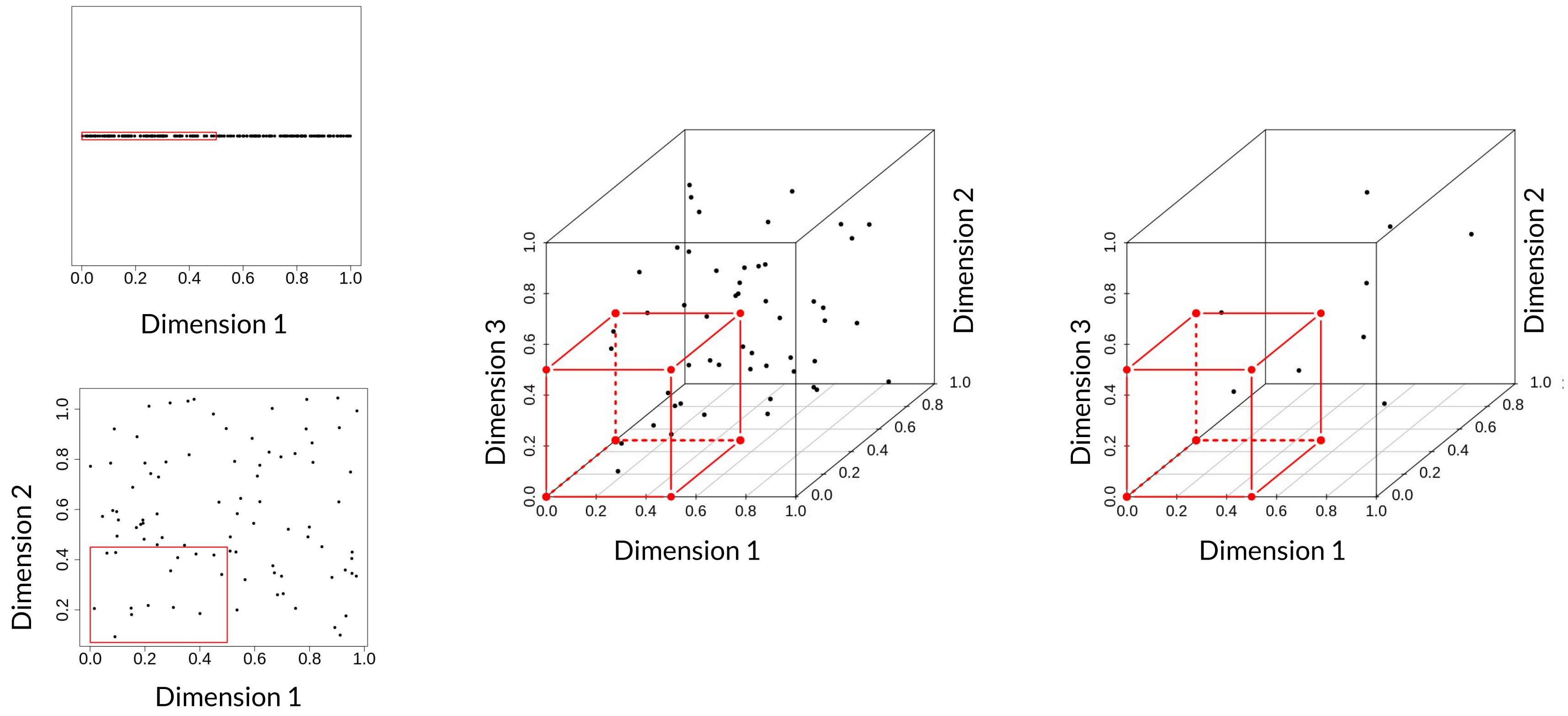
Curse of dimensionality in the distance function

Euclidean distance

$$d_{ij} = \sqrt{\sum_{k=1}^{n} (x_{ik} - x_{jk})^2}$$

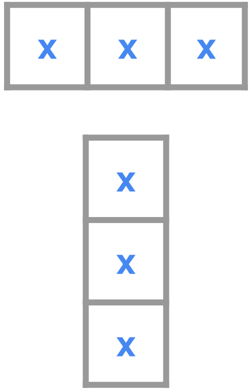
- New dimensions add non-negative terms to the sum
- Distance increases with the number of dimensions
- **For a given number of examples**, the feature space becomes increasingly sparse

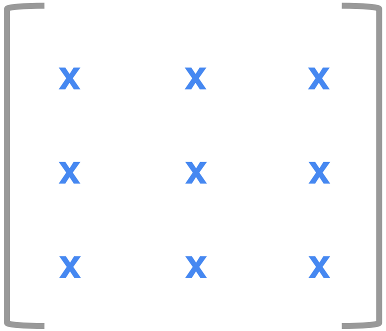
Increasing sparsity with higher dimensions

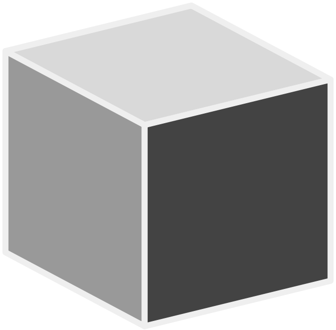


The Hughes effect

The more the features, the larger the hypothesis space







The lower the hypothesis space

- the easier it is to find the correct hypothesis
- the less examples you need

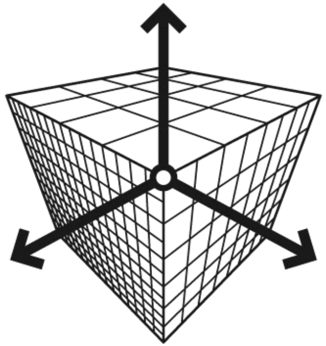
Dimensionality Reduction

DeepLearning.AI

Curse of Dimensionality: An example

How dimensionality impacts in other ways

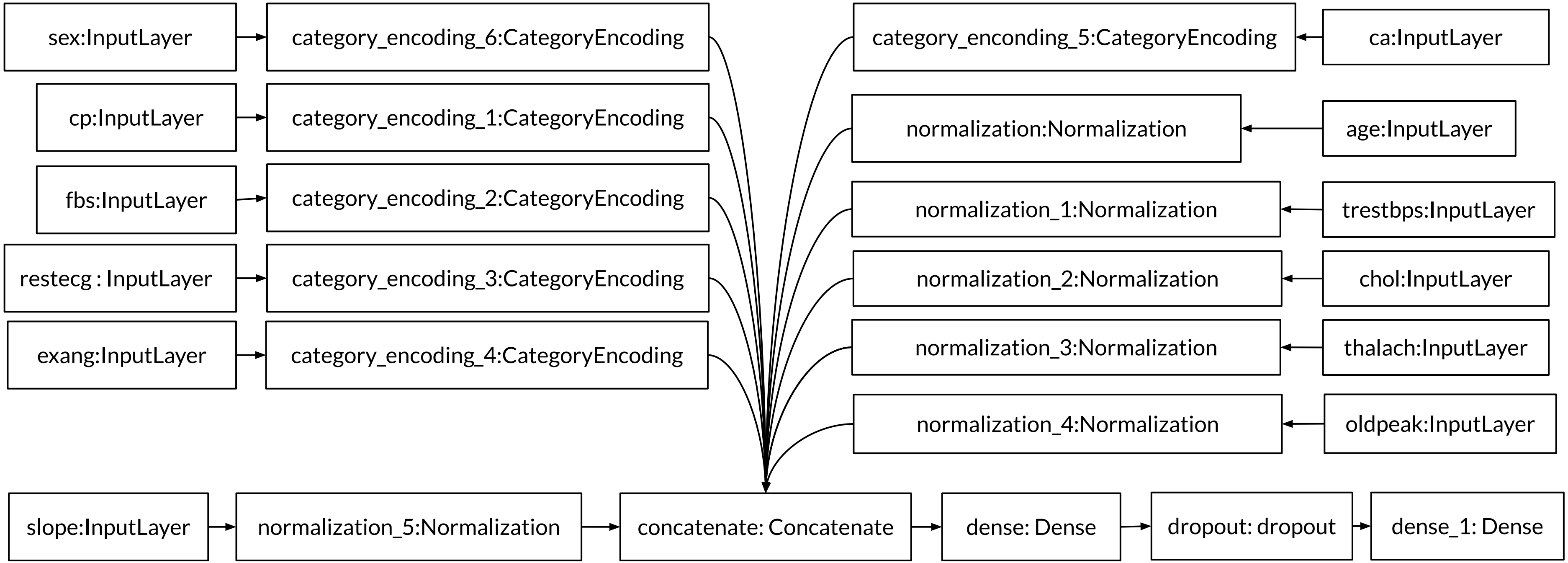
- Runtime and system memory requirements
- Solutions take longer to reach global optima
- More dimensions raise the likelihood of correlated features



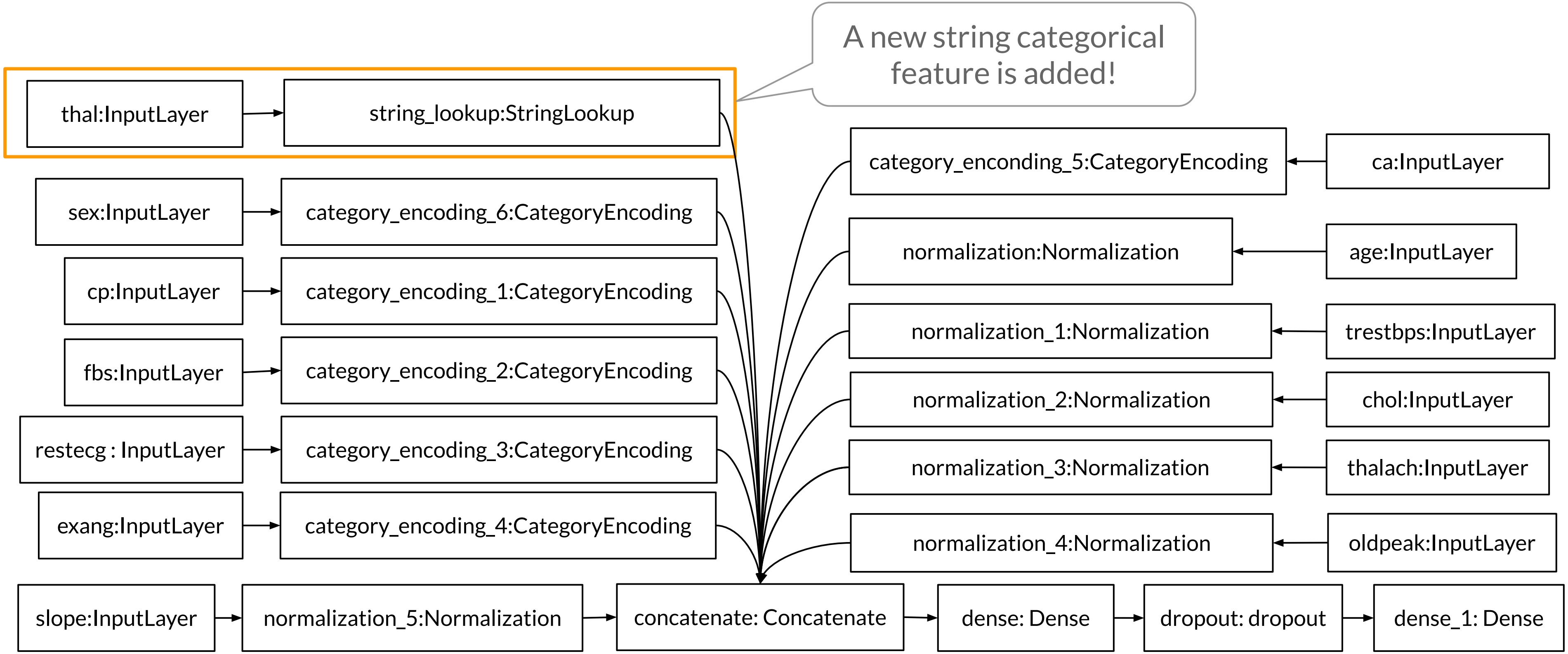
More features require more training data

- More features aren't better if they don't add predictive information
- Number of training instances needed increases exponentially with each added feature
- Reduces real-world usefulness of models

Model #1 (missing a single feature)



Model #2 (adds a new feature)



Comparing the two models' trainable variables

```
from tensorflow.python.keras.utils.layer_utils import count_params

# Number of training parameters in Model #1
>>> count_params(model_1.trainable_variables)
    833


# Number of training parameters in Model #2 (with an added feature)
>>> count_params(model_1.trainable_variables)
    1057
```

What do ML models need?

- No hard and fast rule on how many features are required
- Number of features to be used vary depending on
- Prefer uncorrelated data containing information to produce correct results

Dimensionality Reduction

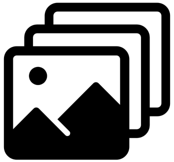
Manual Dimensionality Reduction

Increasing predictive performance

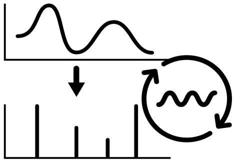
- Features must have information to produce correct results
- Derive features from inherent features
- Extract and recombine to create new features

Feature explosion

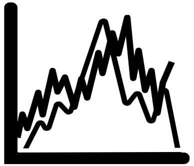
Initial features



pixels, contours, textures, etc.



samples, spectrograms, etc.



ticks, trends, reversals, etc.

dna, marker sequences, genes, etc.

words, grammatical classes and relations, etc.

Combining features

- Number of features grows very quickly
- Reduce dimensionality

Why reduce dimensionality?

Storage

Computational

Consistency

Visualization

Major techniques for dimensionality reduction

Engineering

Selection

Feature Engineering

Need for manually crafting features

Certainly provides food for thought

Engineer features

- Tabular - aggregate, combine, decompose
- Text-extract context indicators
- Image-prescribe filters for relevant structures

It's an iterative process

Come up with ideas to construct "better" features

Devising features to reduce dimensionality

Select the right features to maximize predictiveness

Evaluate models using chosen features

DeepLearning.AI

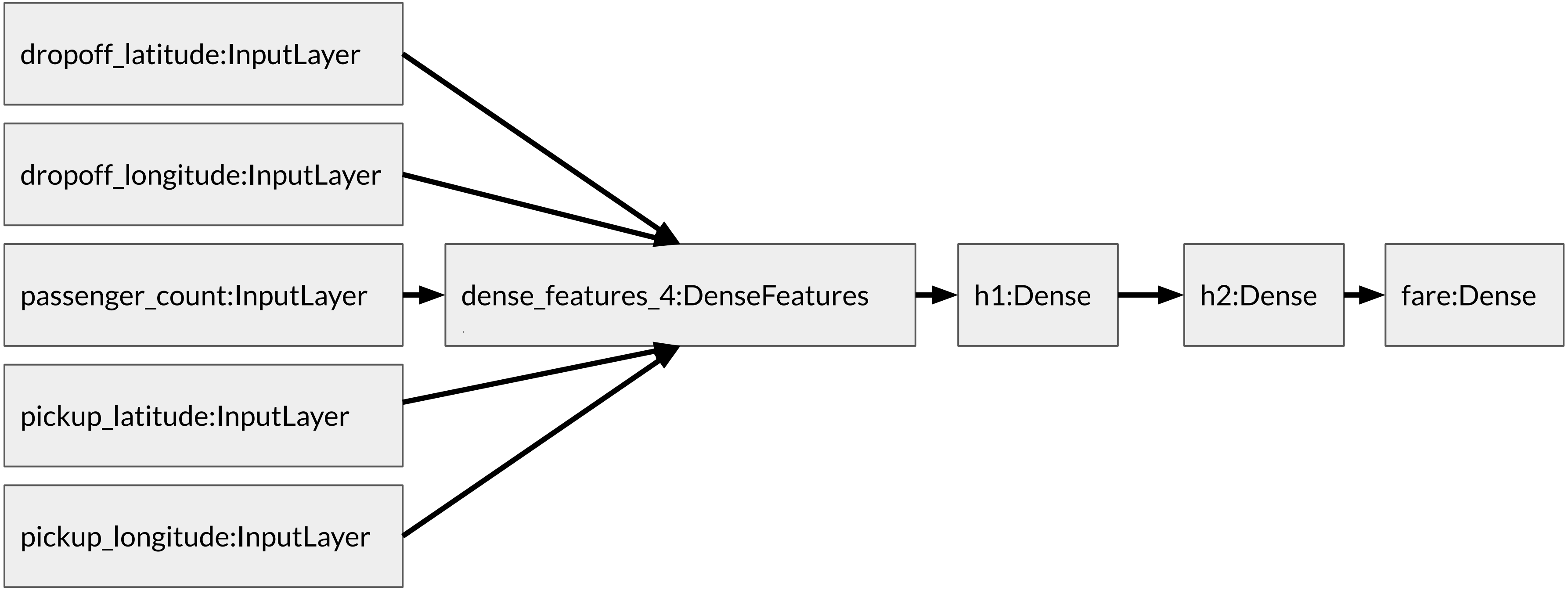
Dimensionality Reduction

Manual Dimensionality Reduction: case study

Taxi Fare dataset

```
CSV_COLUMNS = [
    'fare_amount',
    'pickup_datetime', 'pickup_longitude', 'pickup_latitude',
    'Dropoff_longitude', 'dropoff_latitude',
    'passenger_count', 'key',
]

LABEL_COLUMN = 'fare_amount'
STRING_COLS = ['pickup_datetime']
NUMERIC_COLS = ['pickup_longitude', 'pickup_latitude',
                'dropoff_longitude', 'dropoff_latitude',
                'passenger_count']

DEFAULTS = [[0.0], ['na'], [0.0], [0.0], [0.0], [0.0], [0.0], ['na']]
DAYS = ['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat']
```

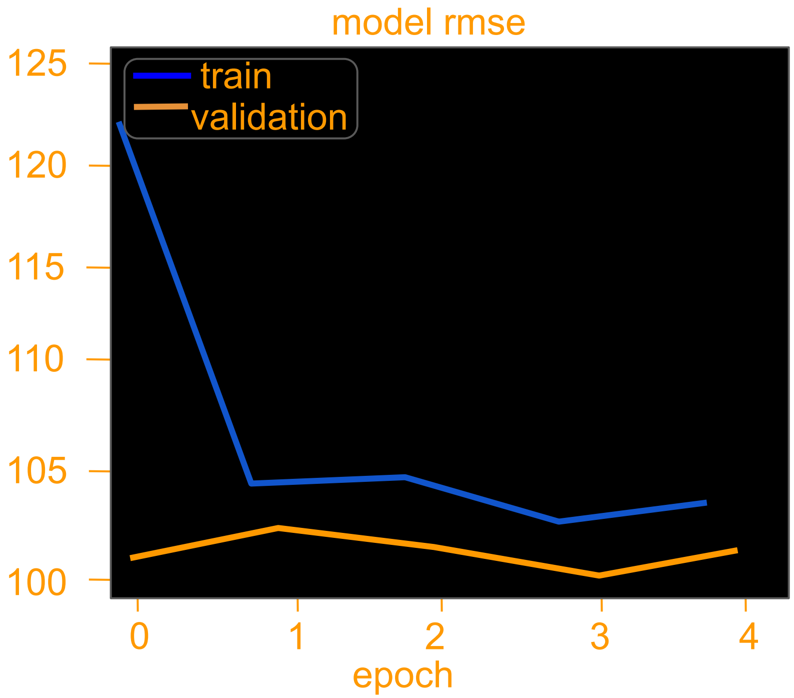
Build the model in Keras



Build a baseline model using raw features

```
from tensorflow.keras import layers
from tensorflow.keras.metrics import RootMeanSquared as RMSE

dnn_inputs = layers.DenseFeatures(feature_columns.values())(inputs)

h1 = layers.Dense(32, activation='relu', name='h1')(dnn_inputs)
h2 = layers.Dense(8, activation='relu', name='h2')(h1)

output = layers.Dense(1, activation='linear', name='fare')(h2)
model = models.Model(inputs, output)
model.compile(optimizer='adam', loss='mse',
              metrics=[RMSE(name='rmse'), 'mse'])
```

Train the model



Increasing model performance with Feature Engineering

- Carefully craft features for the data types
  - Temporal (pickup date & time)
  - Geographical (latitude and longitude)

Handling temporal features

```
def parse_datetime(s):
    if type(s) is not str:
        s = s.numpy().decode('utf-8')
    return datetime.datetime.strptime(s, "%Y-%m-%d %H:%M:%S %Z")

def get_dayofweek(s):
    ts = parse_datetime(s)
    return DAYS[ts.weekday()]

@tf.function
def dayofweek(ts_in):
    return tf.map_fn(
        lambda s: tf.py_function(get_dayofweek, inp=[s],
                                 Tout=tf.string),
        ts_in)
```

Geolocational features

```python
def euclidean(params):
    lon1, lat1, lon2, lat2 = params
    londiff = lon2 - lon1
    latdiff = lat2 - lat1
    return tf.sqrt(londiff * londiff + latdiff * latdiff)
```

Scaling latitude and longitude

```
def scale_longitude(lon_column):
    return (lon_column + 78)/8.


def scale_latitude(lat_column):
    return (lat_column - 37)/8.
```

Preparing the transformations

```
def transform(inputs, numeric_cols, string_cols, nbuckets):
    ...
    feature_columns = {
        colname: tf.feature_column.numeric_column(colname)
        for colname in numeric_cols
    }
  for lon_col in ['pickup_longitude', 'dropoff_longitude']:
        transformed[lon_col] = layers.Lambda(scale_longitude,
            ...)(inputs[lon_col])
  for lat_col in ['pickup_latitude', 'dropoff_latitude']:
        transformed[lat_col] = layers.Lambda(
            scale_latitude,
            ...)(inputs[lat_col])
    ...
```

Computing the Euclidean distance

```
def transform(inputs, numeric_cols, string_cols, nbuckets):
    ...
    transformed['euclidean'] = layers.Lambda(
        euclidean,
        name='euclidean')([inputs['pickup_longitude'],
                           inputs['pickup_latitude'],
                           inputs['dropoff_longitude'],
                           inputs['dropoff_latitude']])
    feature_columns['euclidean'] = fc.numeric_column('euclidean')
     ...
```

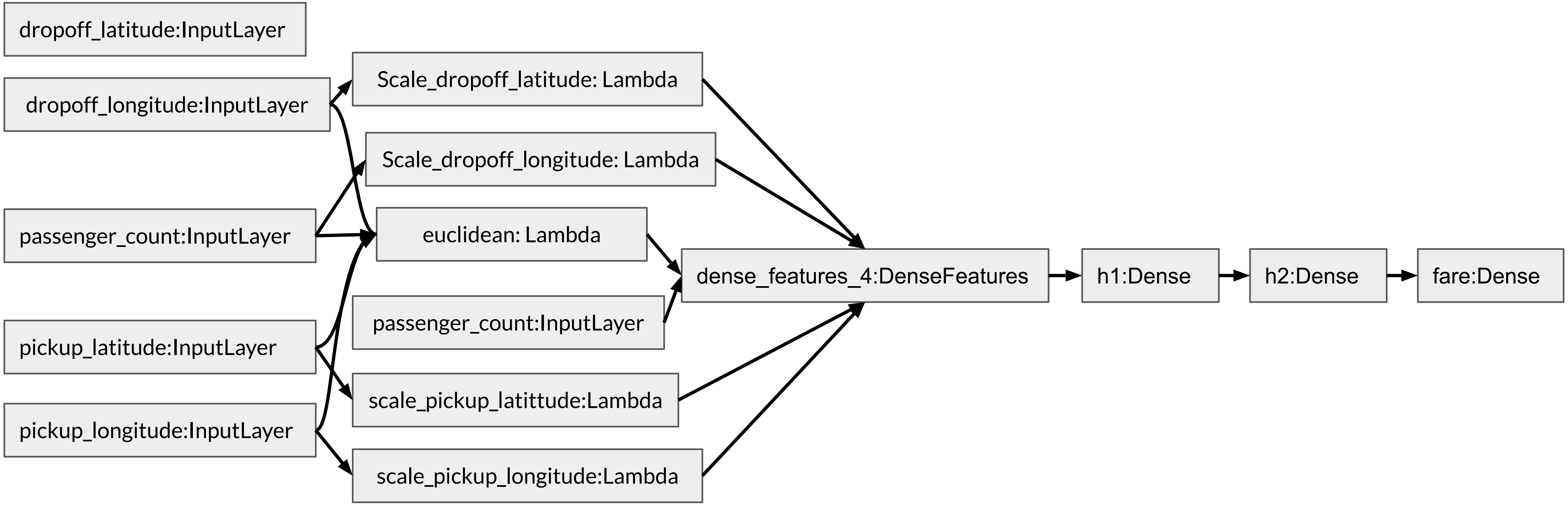
Bucketizing and feature crossing

```
def transform(inputs, numeric_cols, string_cols, nbuckets):
    ...
    latbuckets = np.linspace(0, 1, nbuckets).tolist()
    lonbuckets = ... # Similarly for longitude
    b_plat = fc.bucketized_column(
        feature_columns['pickup_latitude'], latbuckets)
    b_dlat = # Bucketize 'dropoff_latitude'
    b_plon = # Bucketize 'pickup_longitude'
    b_dlon = # Bucketize 'dropoff_longitude'
```

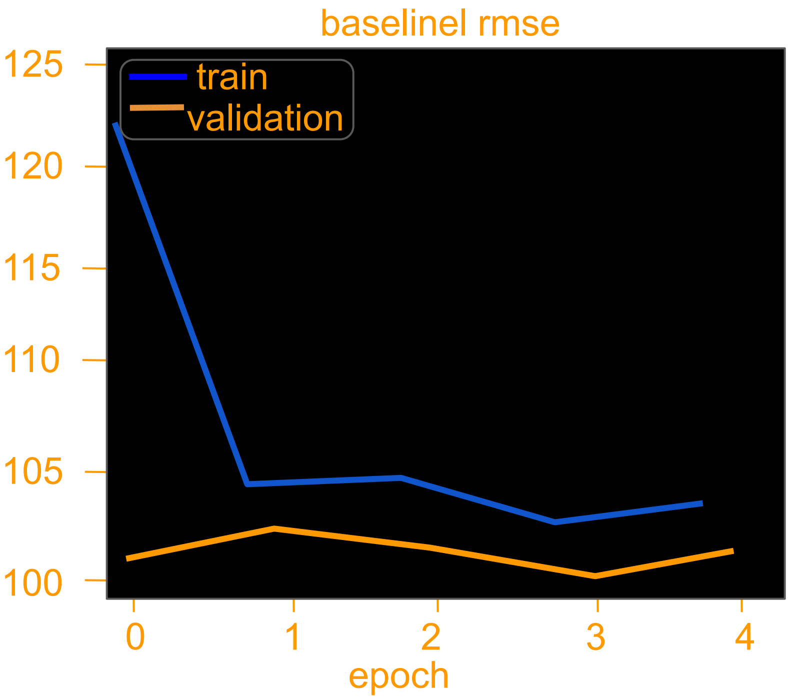
Bucketizing and feature crossing

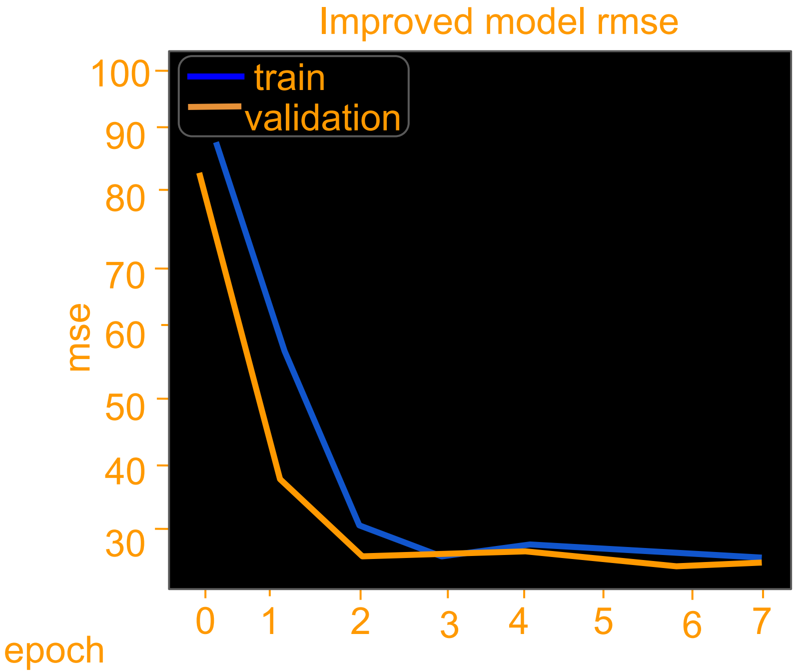
```
ploc = fc.crossed_column([b_plat, b_plon], nbuckets * nbuckets)
dloc = # Feature cross 'b_dlat' and 'b_dlon'
pd_pair = fc.crossed_column([ploc, dloc], nbuckets ** 4)


feature_columns['pickup_and_dropoff'] = fc.embedding_column(pd_pair,
100)
```

Build a model with the engineered features



Train the new feature engineered model





DeepLearning.AI

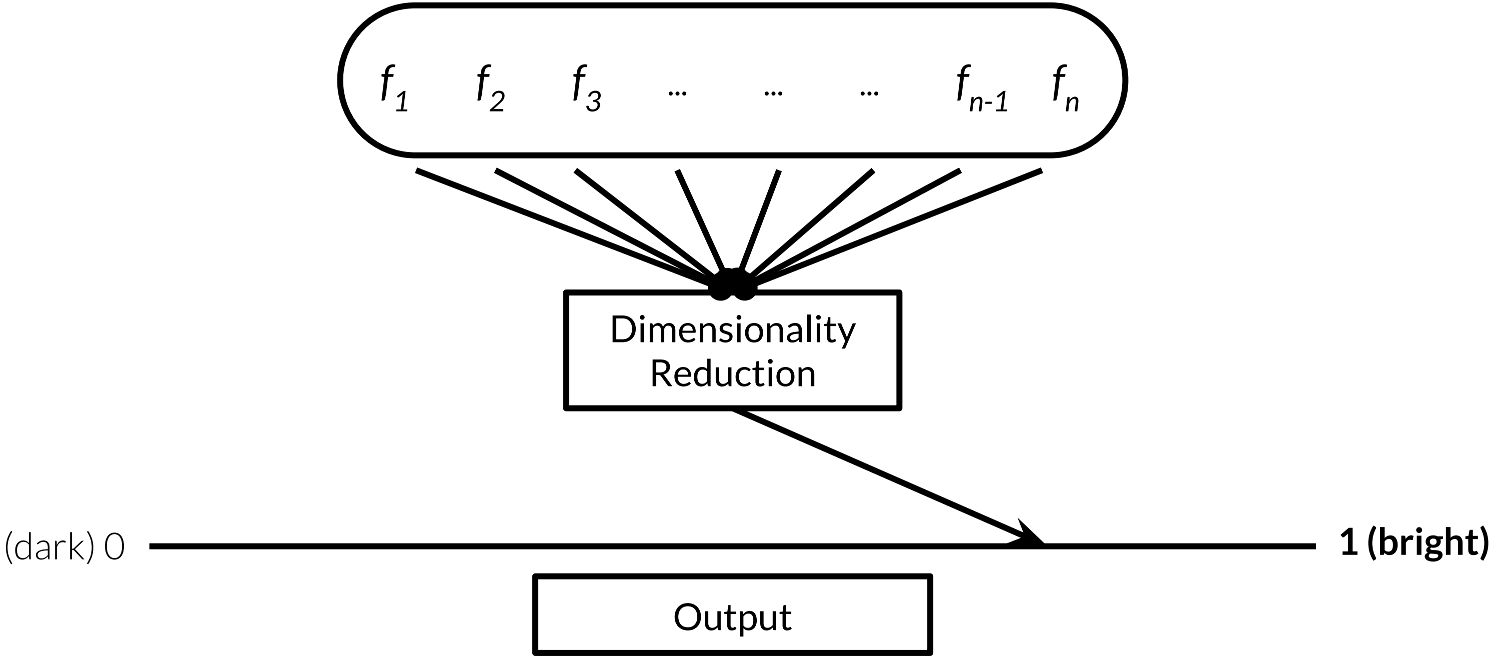
Dimensionality Reduction

Algorithmic Dimensionality Reduction

Linear dimensionality reduction

- Linearly project n-dimensional data onto a k-dimensional subspace ($k < n$, often $k << n$)
- There are infinitely many k-dimensional subspaces we can project the data onto
- Which one should we choose?

Projecting onto a line



DeepLearning.AI

Best k-dimensional subspace for projection

Classification: maximize separation among classes

Example: Linear discriminant analysis (LDA)

Regression: maximize correlation between projected data and response variable

Example: Partial least squares (PLS)

Unsupervised: retain as much data variance as possible

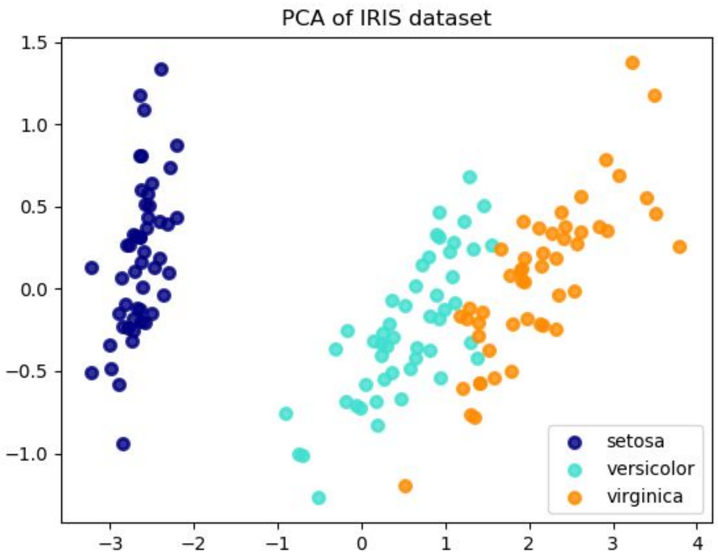
Example: Principal component analysis (PCA)

Dimensionality Reduction

Principal Component Analysis

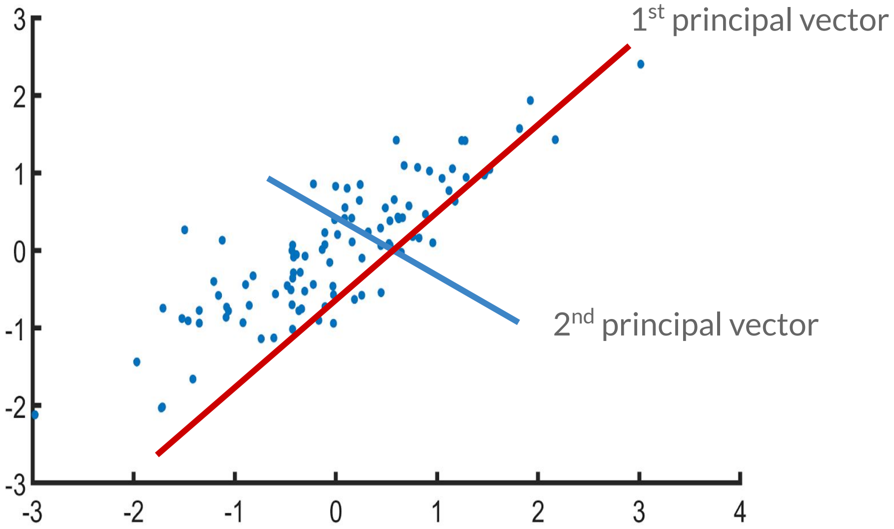
Principal component analysis (PCA)

- PCA is a minimization of the orthogonal distance
- Widely used method for unsupervised & linear dimensionality reduction
- Accounts for variance of data in as few dimensions as possible using linear projections

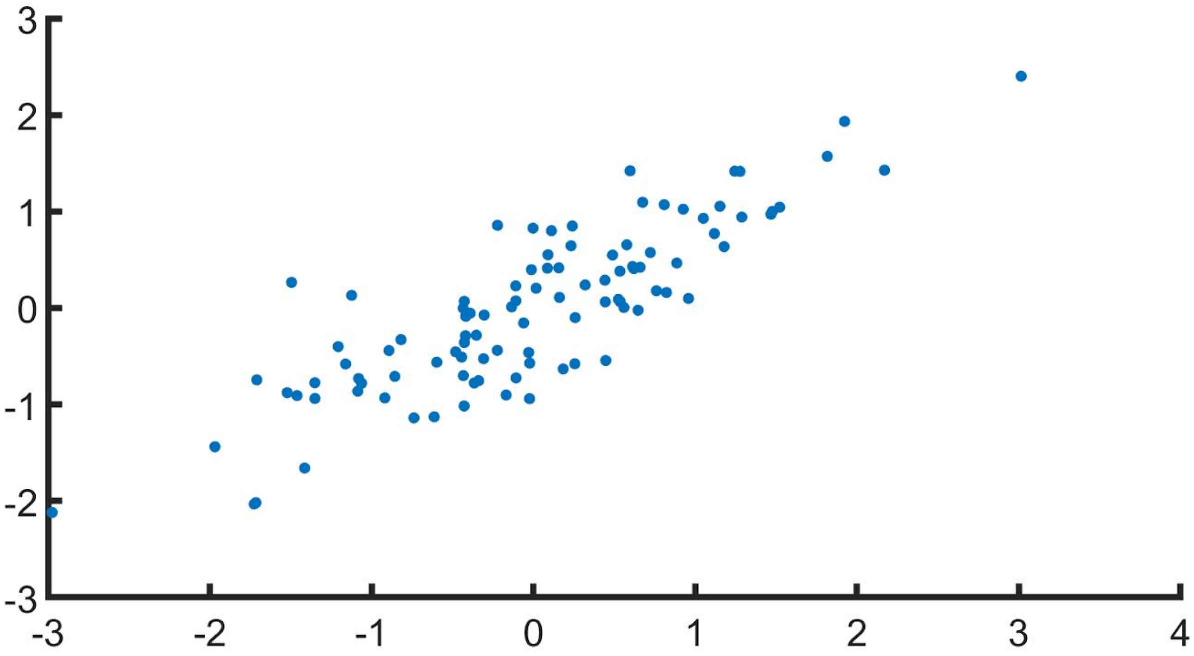


Principal components (PCs)

- PCs maximize the variance of projections
- PCs are orthogonal
- Gives the best axis to project
- Goal of PCA: Minimize total squared reconstruction error

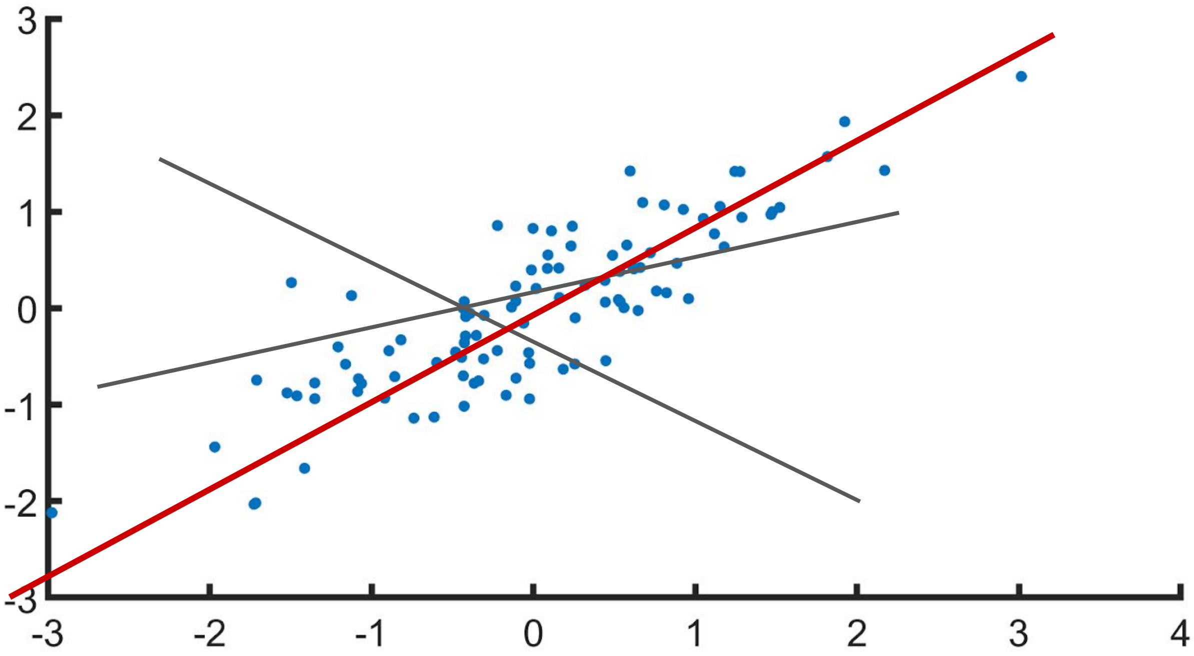


2-D data



PCA Algorithm - First Principal Component

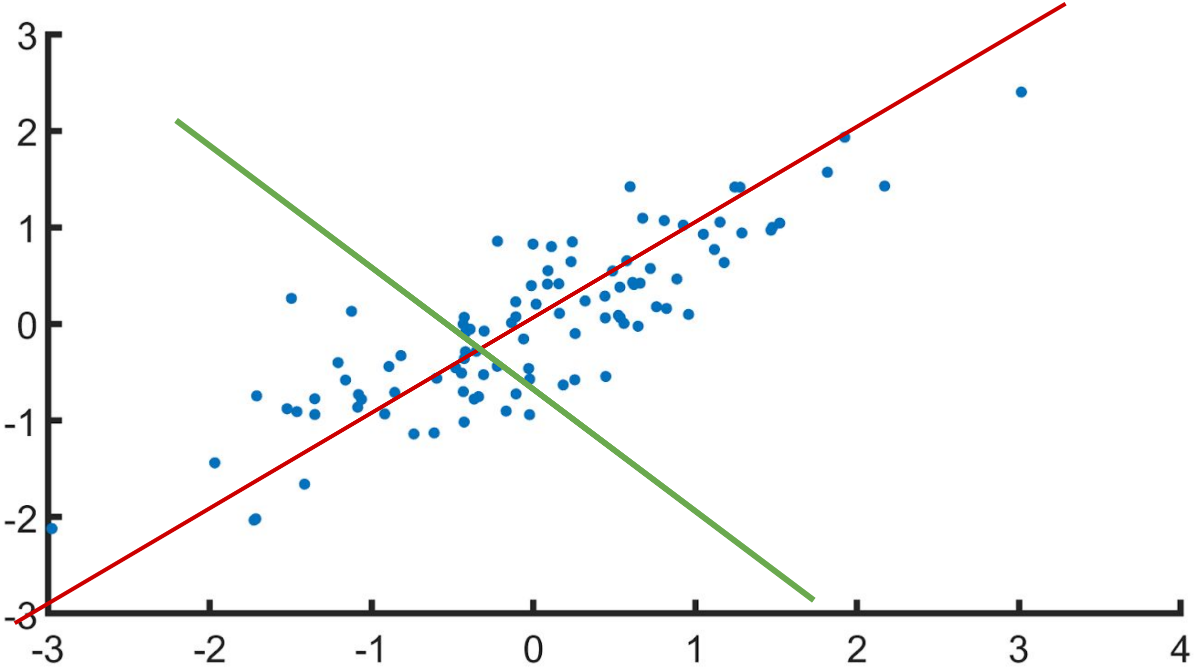
Step 1



Find a line, such that when the data is projected onto that line, it has the maximum variance

PCA Algorithm - Second Principal Component

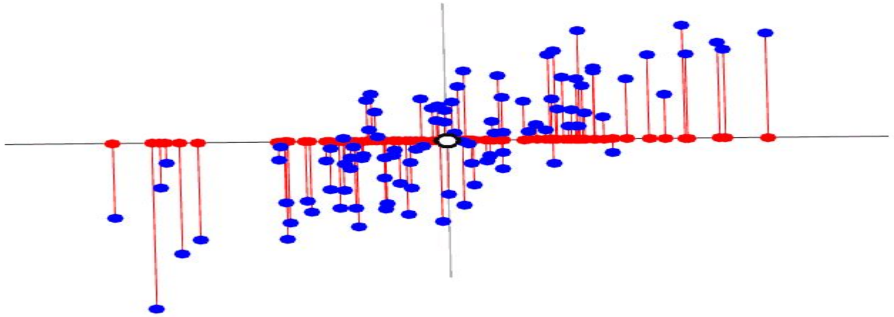
Step 2



Find a second line, orthogonal to the first, that has maximum projected variance

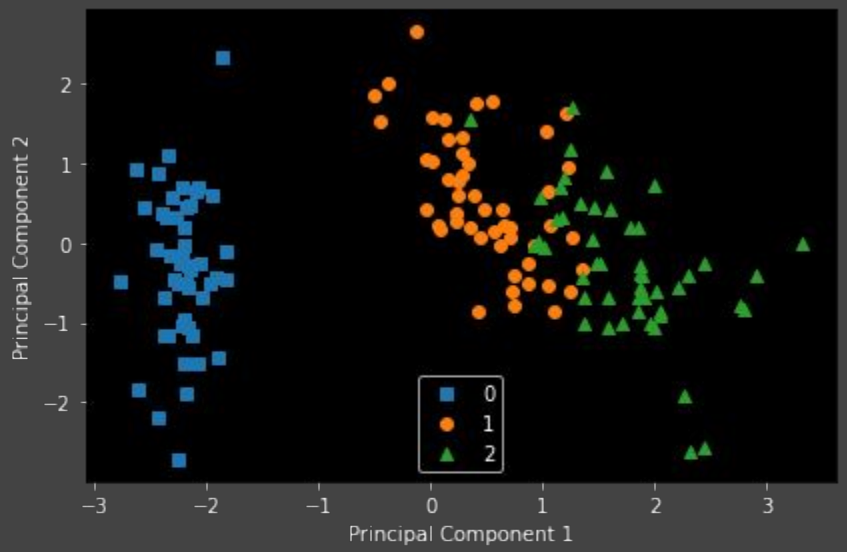
PCA Algorithm

Step 3



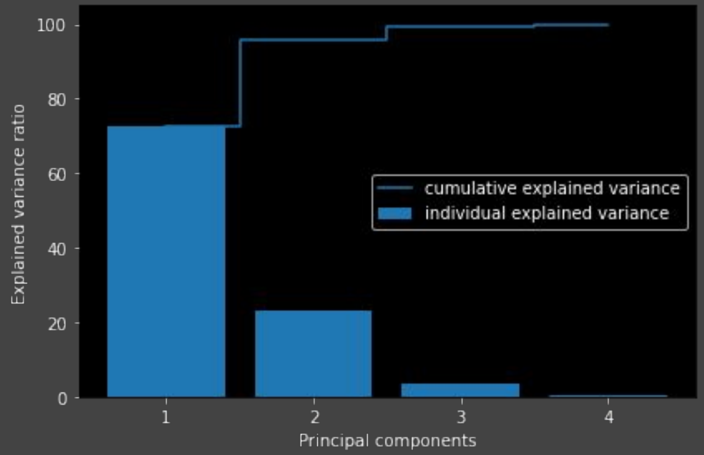
Repeat until we have k orthogonal lines

Applying PCA on Iris



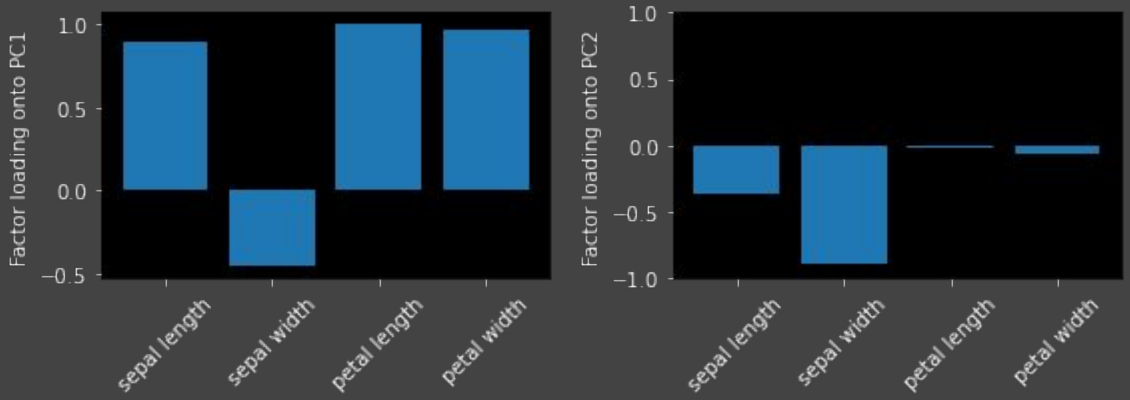

```
pca = PrincipalComponentAnalysis(n_components=2)
pca.fit(X)
X_pca = pca.transform(X)
```

Plot the explained variance




```
tot = sum(pca.e_vals_)
var_exp = [(i / tot) * 100 for i in sorted(pca.e_vals_, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
```

PCA factor loadings




```
loadings = pca.e_vecs_ * np.sqrt(pca.e_vals_)
```

PCA in scikit-learn

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn import datasets

# Load the data
digits = datasets.load_digits()

# Standardize the feature matrix
X = StandardScaler().fit_transform(digits.data)
```

PCA in scikit-learn

```
# Create a PCA that will retain 99% of the variance
pca = PCA(n_components=0.99, whiten=True)

# Conduct PCA
X_pca = pca.fit_transform(X)
```

When to use PCA?

Strengths

- A versatile technique
- Fast and simple
- Offers several variations and extensions (e.g., kernel/sparse PCA)

Weaknesses

- Result is not interpretable
- Requires setting threshold for cumulative explained variance

Dimensionality Reduction

Other Techniques

More dimensionality reduction algorithms

Unsupervised

- Latent Semantic Indexing/Analysis (LSI and LSA) (SVD)
- Independent Component Analysis (ICA)

Matrix Factorization

- Non-Negative Matrix Factorization (NMF)

Latent Methods

- Latent Dirichlet Allocation (LDA)

Singular value decomposition (SVD)

- SVD decomposes non-square matrices
- Useful for sparse matrices as produced by TF-IDF
- Removes redundant features from the dataset

Independent Components Analysis (ICA)

- PCA seeks directions in feature space that minimize reconstruction error
- ICA seeks directions that are most statistically independent
- ICA addresses higher order dependence

How does ICA work?

- Assume there exists independent signals:

$$S = [s_1 (t) , s_2(t) , \dots , s_N (t)]$$

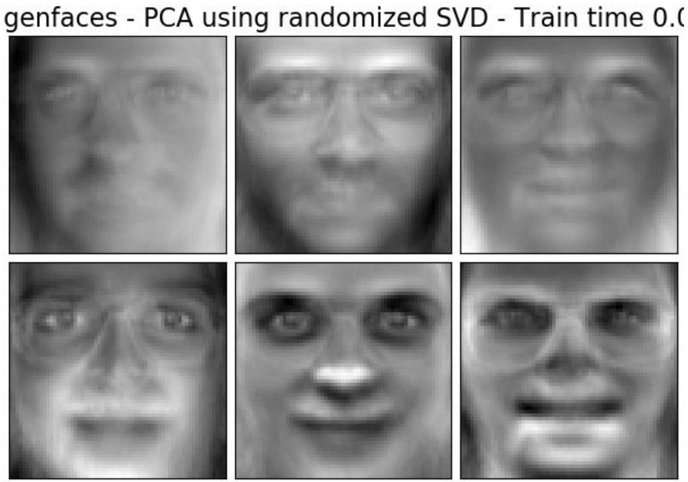
- Linear combinations of signals: $Y(t) = A\ S(t)$
  - Both A and S are unknown
  - A - mixing matrix
- Goal of ICA: recover original signals, $S(t)$ from $Y(t)$

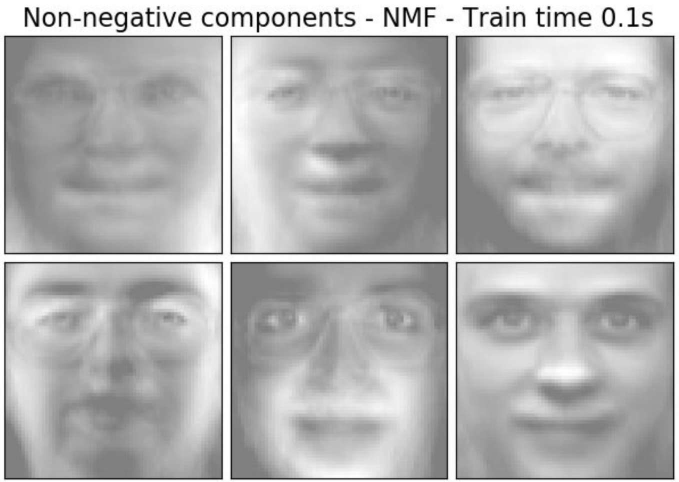
Comparing PCA and ICA

	PCA	ICA
Removes correlations	✓	✓
Removes higher order dependence		✓
All components treated fairly?		✓
Orthogonality	✓	

Non-negative Matrix Factorization (NMF)

- NMF models are interpretable and easier to understand
- NMF requires the sample features to be non-negative





DeepLearning.AI

Quantization & Pruning

Mobile, IoT, and Similar Use Cases

Trends in adoption of smart devices

Factors driving this trend

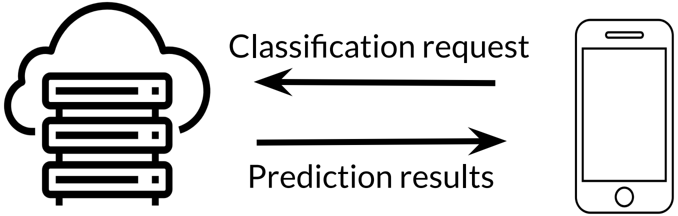
- Demands move ML capability from cloud to on-device
- Cost-effectiveness
- Compliance with privacy regulations

Online ML inference

- To generate real-time predictions you can:
  - Host the model on a server
  - Embed the model in the device
- Is it faster on a server, or on-device?
- Mobile processing limitations?

Mobile inference

Inference on the cloud/server



Pros

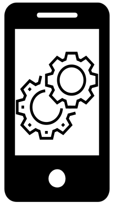
- Lots of compute capacity
- Scalable hardware
- Model complexity handled by the server
- Easy to add new features and update the model
- Low latency and batch prediction

Cons

- Timely inference is needed

Mobile inference

On-device Inference



Pro

- Improved speed
- Performance
- Network connectivity
- No to-and-fro communication needed

Cons

- Less capacity
- Tight resource constraints

Model deployment

Options	On-device inference	On-device personalization	On-device training	Cloud-based web service	Pretrained models	Custom models
ML Kit	✓	✓		✓	✓	✓
Core ML	✓	✓	✓		✓	✓
TensorFlow Lite *	✓	✓	✓		✓	✓

* Also supported in TFX

DeepLearning.AI

Quantization & Pruning

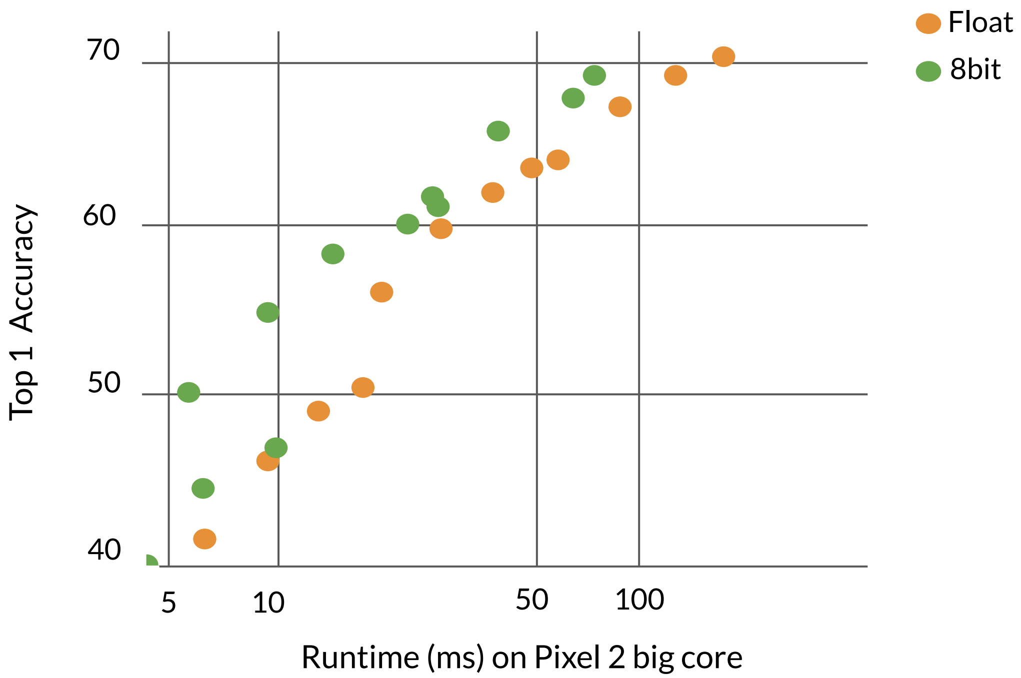
Benefits and Process of Quantization

Quantization

Why quantize neural networks?

- Neural networks have many parameters and take up space
- Shrinking model file size
- Reduce computational resources
- Make models run faster and use less power with low-precision

MobileNets: Latency vs Accuracy trade-off

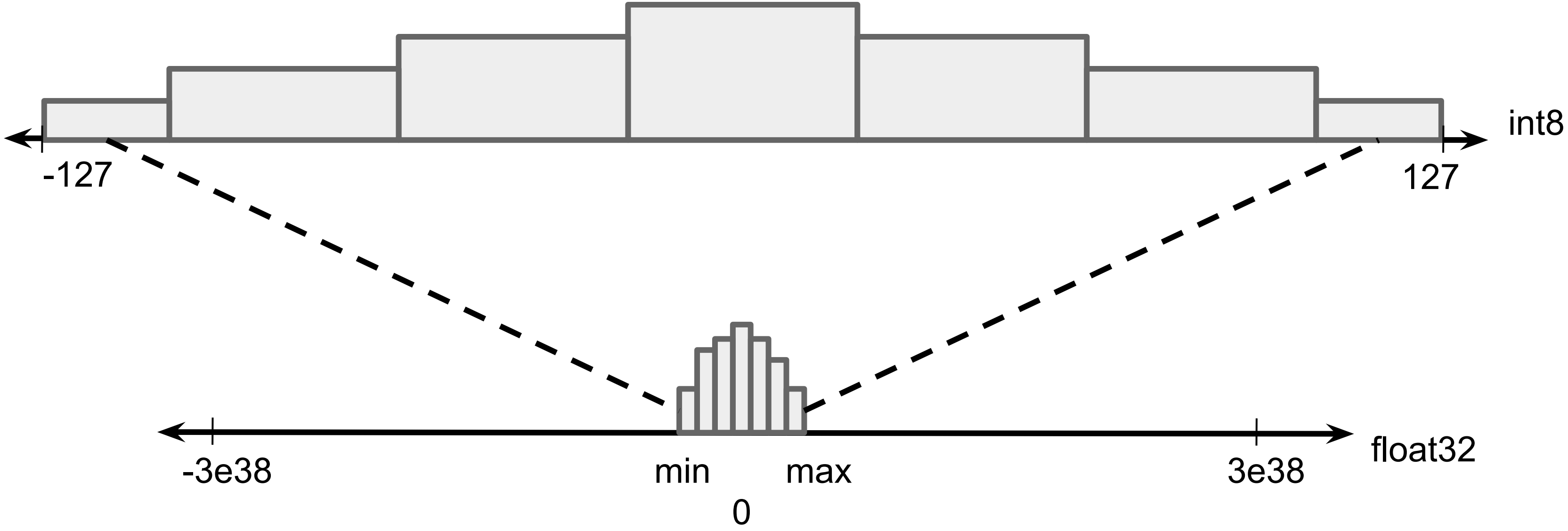


Snapdragon 835

Benefits of quantization

- Faster compute
- Low memory bandwidth
- Low power
- Integer operations supported across CPU/DSP/NPUs

The quantization process



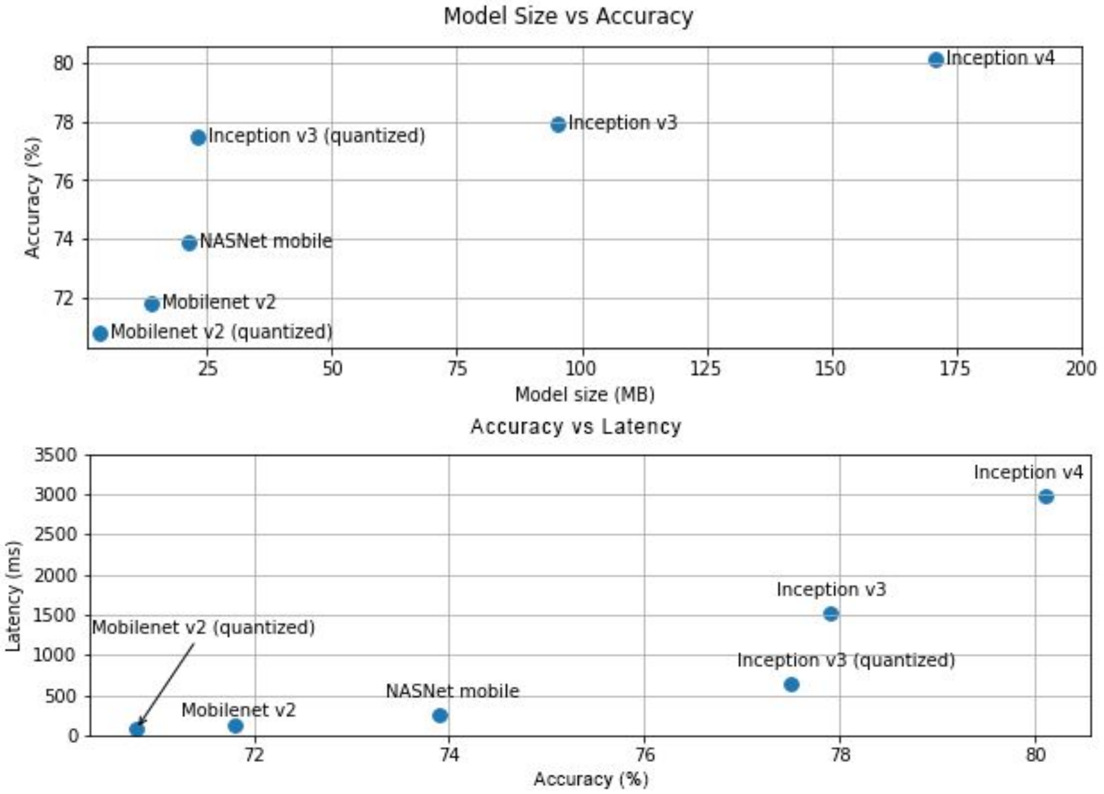
What parts of the model are affected?

- Static values (parameters)
- Dynamic values (activations)
- Computation (transformations)

Trade-offs

- Optimizations impact model accuracy
    - Difficult to predict ahead of time
- In rare cases, models may actually gain some accuracy
- Undefined effects on ML interpretability

Choose the best model for the task



DeepLearning.AI

Quantization & Pruning

Post Training Quantization

Post-training quantization

- Reduced precision representation
- Incur small loss in model accuracy
- Joint optimization for model and latency

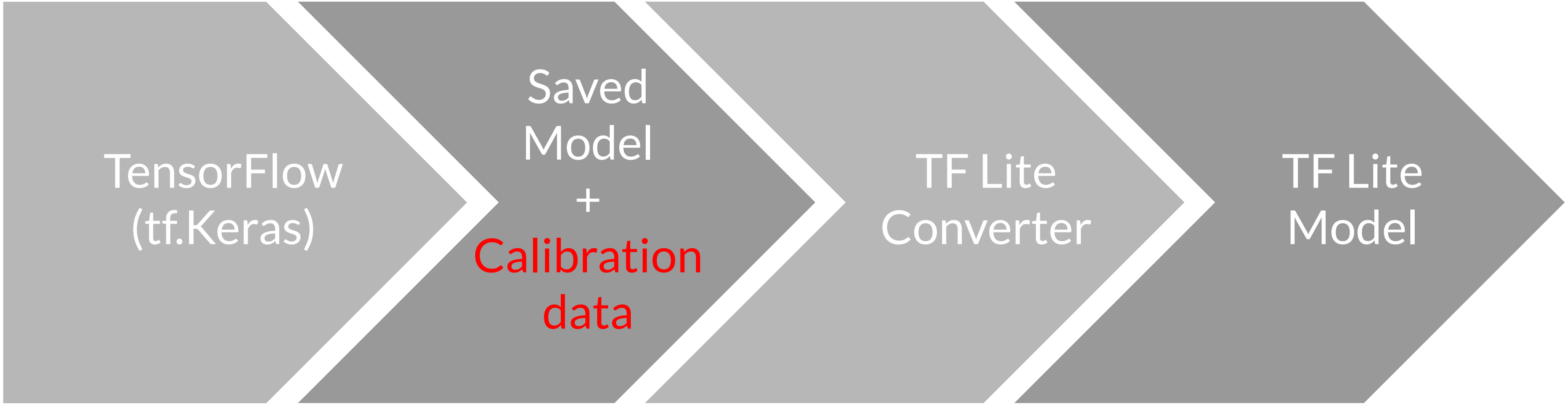
Post-training quantization

Technique	Benefits
Dynamic range quantization	4x smaller, 2x-3x speedup
Full integer quantization	4x smaller, 3x+ speedup
float16 quantization	2x smaller, GPU acceleration

Post training quantization

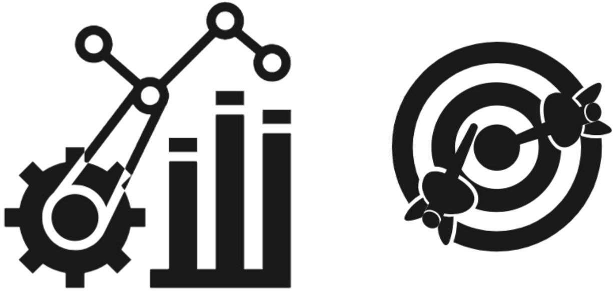
```
import tensorflow as tf

converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)

converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]


tflite_quant_model = converter.convert()
```

Post-training integer quantization



Model accuracy

- Small accuracy loss incurred (mostly for smaller networks)
- Use the benchmarking tools to evaluate model accuracy
- If the loss of accuracy drop is not within acceptable limits, consider using quantization-aware training



Quantization & Pruning

Quantization Aware Training

Quantization-aware training (QAT)

- Inserts fake quantization (FQ) nodes in the forward pass
- Rewrites the graph to emulate quantized inference
- Reduces the loss of accuracy due to quantization
- Resulting model contains all data to be quantized according to spec

Quantization-aware training (QAT)

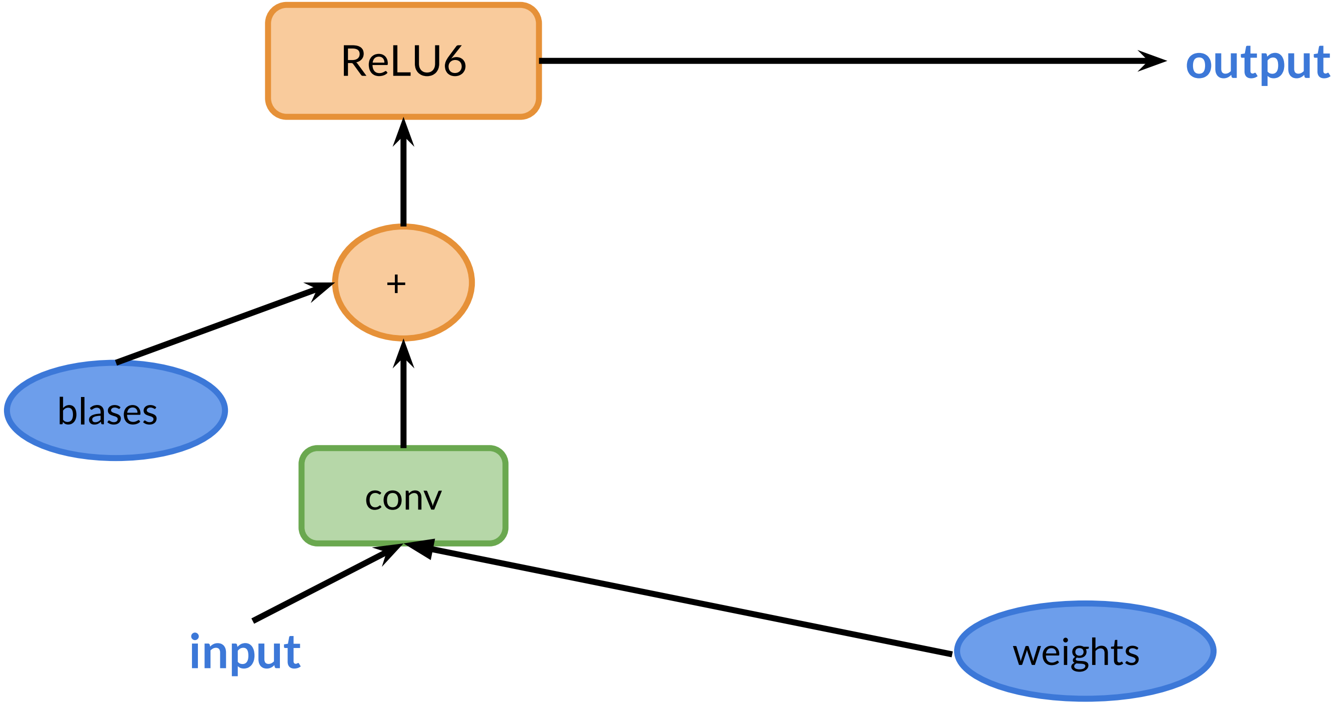
TensorFlow
(tf.Keras)

Apply QAT
+
Train model

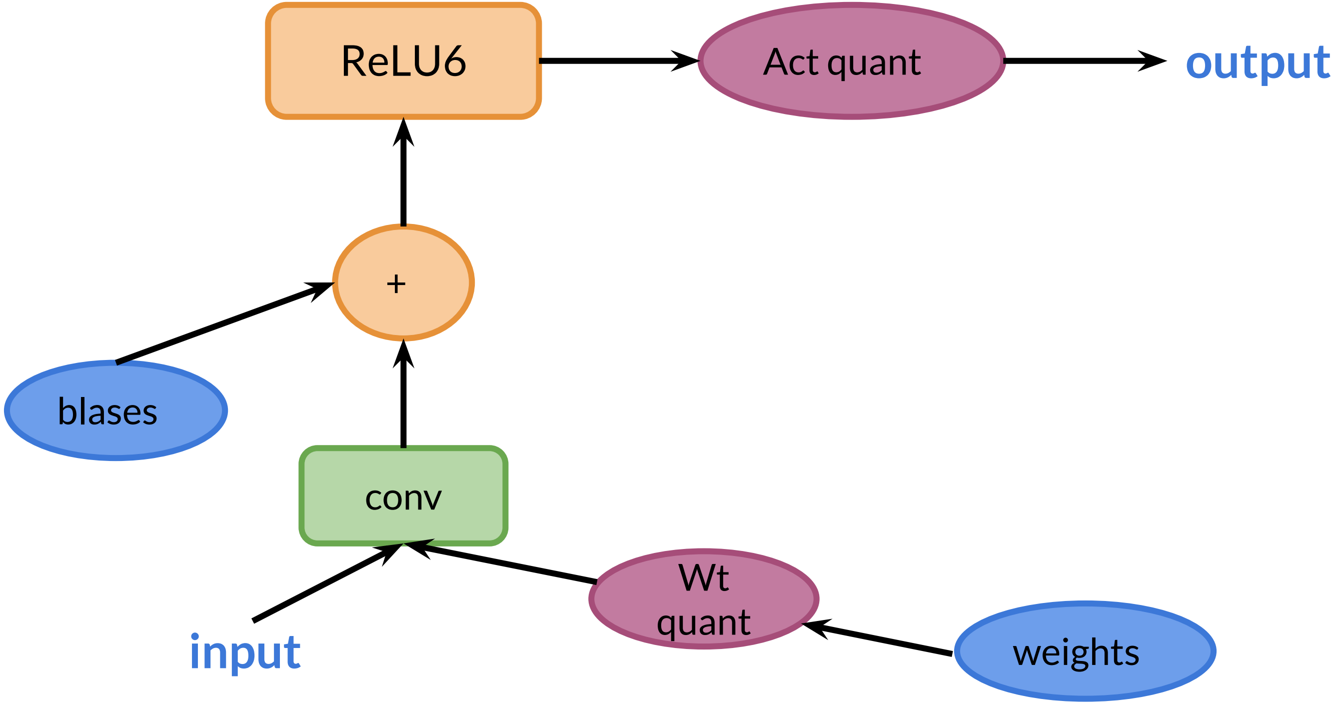
Convert &
Quantize
(TF Lite)

TF Lite
Model

Adding the quantization emulation operations



Adding the quantization emulation operations



QAT on entire model

```
import tensorflow_model_optimization as tfmot

model = tf.keras.Sequential([
    ...
])
# Quantize the entire model.
quantized_model = tfmot.quantization.keras.quantize_model(model)

# Continue with training as usual.
quantized_model.compile(...)
quantized_model.fit(...)
```

Quantize part(s) of a model

```
import tensorflow_model_optimization as tfmot
quantize_annotate_layer = tfmot.quantization.keras.quantize_annotate_layer
model = tf.keras.Sequential([
   ...
   # Only annotated layers will be quantized.
   quantize_annotate_layer(Conv2D()),
   quantize_annotate_layer(ReLU()),
   Dense(),
   ...
])
# Quantize the model.
quantized_model = tfmot.quantization.keras.quantize_apply(model)
```

Quantize custom Keras layer

```
quantize_annotate_layer =
tfmot.quantization.keras.quantize_annotate_layer
quantize_annotate_model =
tfmot.quantization.keras.quantize_annotate_model
quantize_scope = tfmot.quantization.keras.quantize_scope

model = quantize_annotate_model(tf.keras.Sequential([
   quantize_annotate_layer(CustomLayer(20, input_shape=(20,)),
                                DefaultDenseQuantizeConfig()),
   tf.keras.layers.Flatten()
]))
```

Quantize custom Keras layer

```
# `quantize_apply` requires mentioning `DefaultDenseQuantizeConfig` with
`quantize_scope`
with quantize_scope(
  {'DefaultDenseQuantizeConfig': DefaultDenseQuantizeConfig,
   'CustomLayer': CustomLayer}):
  # Use `quantize_apply` to actually make the model quantization aware.
  quant_aware_model = tfmot.quantization.keras.quantize_apply(model)
```

Model Optimization Results - Accuracy

Model	Top-1 Accuracy (Original)	Top-1 Accuracy (Post Training Quantized)	Top-1 Accuracy (Quantization Aware Training)
Mobilenet-v1-1-224	0.709	0.657	0.70
Mobilenet-v2-1-224	0.719	0.637	0.709
Inception_v3	0.78	0.772	0.775
Resnet_v2_101	0.770	0.768	N/A

Model Optimization Results - Latency

Model	Latency (Original) (ms)	Latency (Post Training Quantized) (ms)	Latency (Quantization Aware Training) (ms)
Mobilenet-v1-1-224	124	112	64
Mobilenet-v2-1-224	89	98	54
Inception_v3	1130	845	543
Resnet_v2_101	3973	2868	N/A

Model Optimization Results

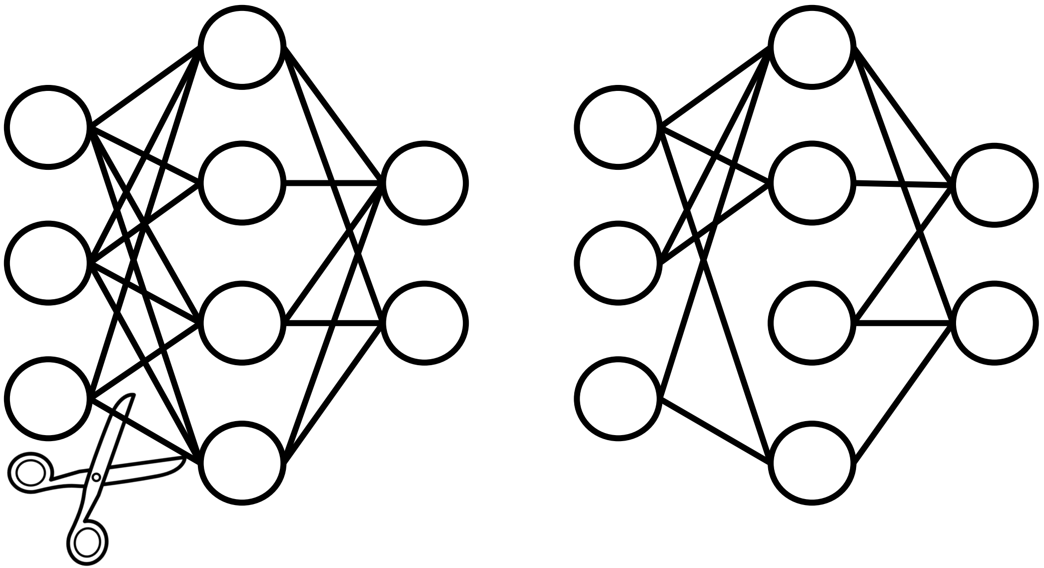
Model	Size (Original) (MB)	Size (Optimized) (MB)
Mobilenet-v1-1-224	16.9	4.3
Mobilenet-v2-1-224	14	3.6
Inception_v3	95.7	23.9
Resnet_v2_101	178.3	44.9

DeepLearning.AI

Quantization & Pruning

Pruning

Connection pruning



Before pruning

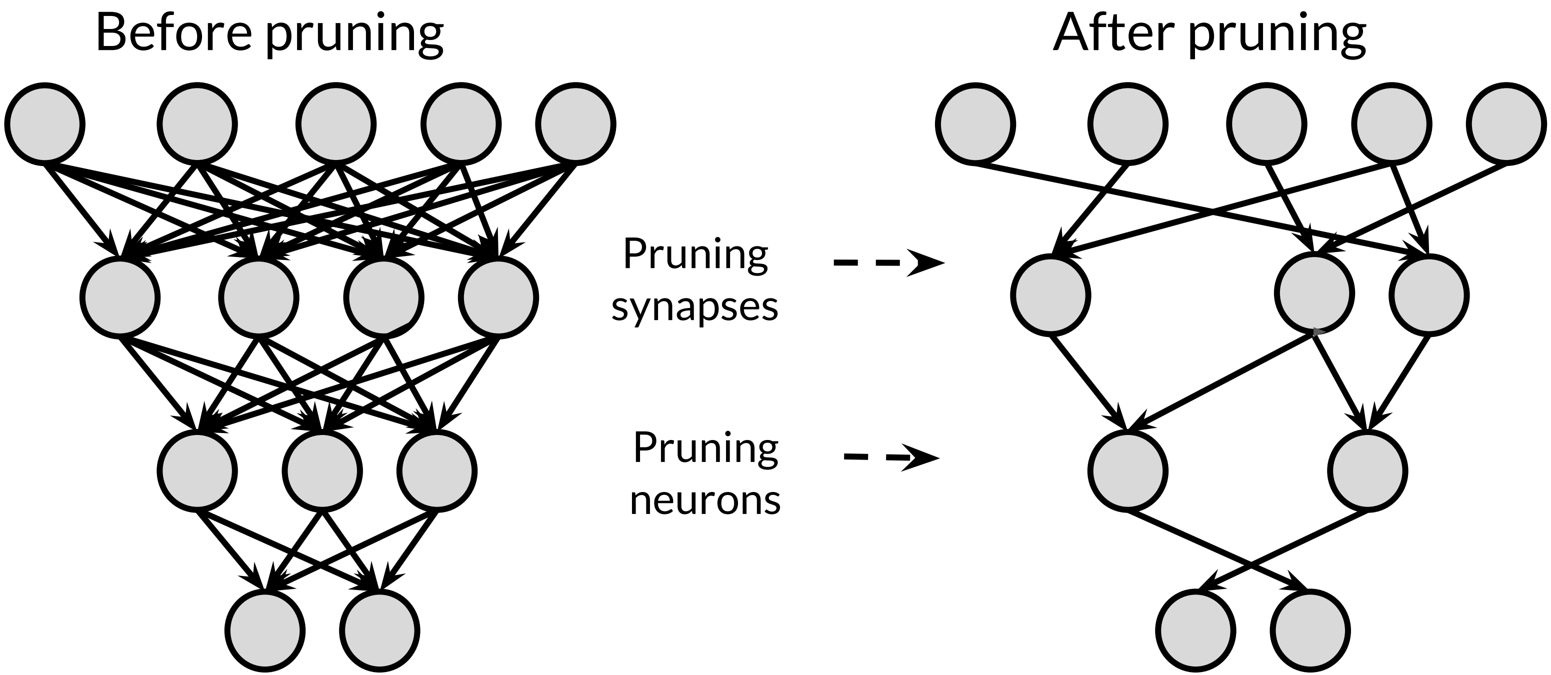
After pruning

Model sparsity

Larger models → More memory → Less efficient

Sparse models → Less memory → More efficient

Origins of weight pruning



The Lottery Ticket Hypothesis

$$p = \frac{1}{3000000}$$

$$\bar{p} = 1 - p$$

$$p_n = 1 - (1 - p)^n$$

Finding Sparse Neural Networks

"A randomly-initialized, dense neural network contains a subnetwork that is initialized such that — when trained in isolation — it can match the test accuracy of the original network after training for at most the same number of iterations"

Jonathan Frankle and Michael Carbin

Pruning research is evolving

- The new method didn't perform well at large scale
- The new method failed to identify the randomly initialized winners
- It's an active area of research

Eliminate connections based on their magnitude

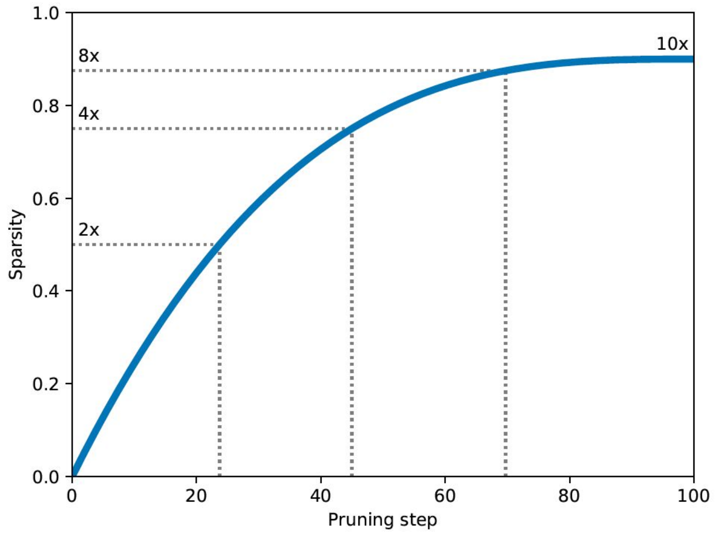
3	2	7	4
9	6	3	8
4	4	1	3
2	3	2	5

0	2	0	4
0	6	3	0
4	0	0	3
0	3	0	5

0	0	7	4
9	6	0	0
0	0	1	3
2	3	0	0

Tensors with no sparsity (left), sparsity in blocks of 1x1 (center), and the sparsity in blocks 1x2 (right)

Apply sparsity with a pruning routine



Example of sparsity ramp-up function with a schedule to start pruning from step 0 until step 100, and a final target sparsity of 90%.

Sparsity increases with training

Black cells indicate where the non-zero weights exist

Animation of pruning applied to a tensor

What's special about pruning?

- Better storage and/or transmission
- Gain speedups in CPU and some ML accelerators
- Can be used in tandem with quantization to get additional benefits
- Unlock performance improvements

Pruning with TF Model Optimization Toolkit

TensorFlow (tf.Keras) → Sparsify + Train model → Convert & Quantize (TF Lite) → TF Lite Model

Pruning with Keras

```
import tensorflow_model_optimization as tfmot
model = build_your_model()
pruning_schedule = tfmot.sparsity.keras.PolynomialDecay(
                        initial_sparsity=0.50, final_sparsity=0.80,
                        begin_step=2000, end_step=4000)

model_for_pruning = tfmot.sparsity.keras.prune_low_magnitude(
                        model,
                        pruning_schedule=pruning_schedule)
...
model_for_pruning.fit(...)
```

Results across different models & tasks

Model	Non-sparse Top-1 acc.	Sparse acc.	Sparsity
Inception V3	78.1%	78.0%	50%
		76.1%	75%
		74.6%	87.5%
Mobilenet V1 224	71.04%	70.84%	50%

Model	Non-sparse BLEU	Sparse BLEU	Sparsity
GNMT EN-DE	26.77	26.86	80%
		26.52	85%
		26.19	90%
GNMT DE-EN	29.47	29.50	80%
		29.24	85%
		28.81	90%