

## 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])
    ...
```

- A continuación, creemos una función de transformación geográfica. Esta función pasa las características de su columna numérica y de cadena como entradas al modelo y luego escala la longitud y la latitud como vimos en la última diapositiva.

## 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')
    ...
```

- Luego calculamos la distancia euclidiana en función de los parámetros de ubicación geográfica.

## 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'
```

- A menos que la geometría específica de la Tierra sea relevante para sus datos, es probable que haya una versión dividida en cubos del mapa. ser más útil que las entradas en bruto. Esto requiere
- dividir en cubos las dimensiones de latitud y longitud por separado y luego cruzarlas de manera efectiva haciendo una clasificación en cubos bidimensional de los datos de ubicación. En este ejemplo, clasifica estas características de latitud y longitud y crea cruces de características a partir de las características de ubicación geográfica.
- Aquí el código crea columnas del tamaño de un cubo para recoger y dejar la latitud y la longitud

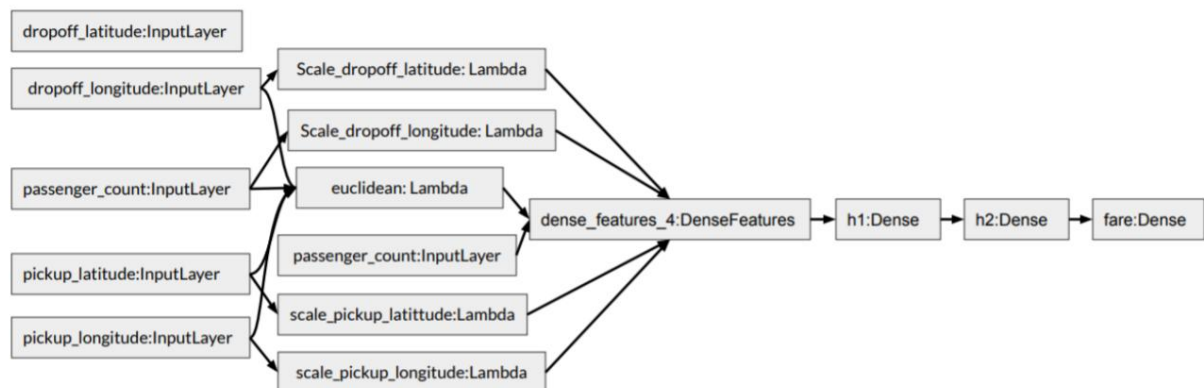
## Bucketizing and feature crossing

```
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)
```

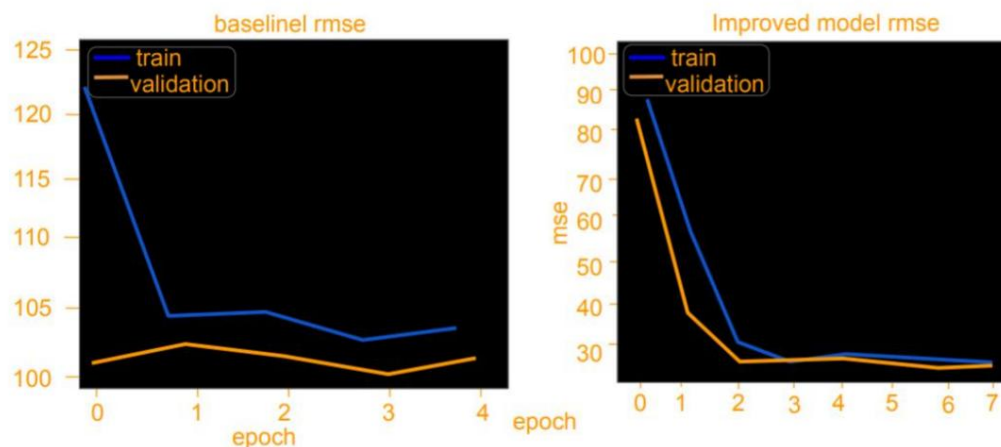
- Luego se procede a crear columnas cruzadas para cada uno. El código combina estos resultados en una incrustación peine.

## Build a model with the engineered features



- Esta es la nueva arquitectura de su modelo. Al igual que con su primer intento de modelo, debe crear un modelo en Keras utilizando la API funcional. Esto, por supuesto, aprovechará toda la ingeniería de funciones que ha realizado hasta ahora. Echemos un vistazo al rendimiento de este nuevo modelo.
-

## Train the new feature engineered model



- Mirando los resultados de entrenamiento y validación, está claro que el modelo con ingeniería de características en el la derecha es una mejora significativa sobre el modelo de referencia de la izquierda.

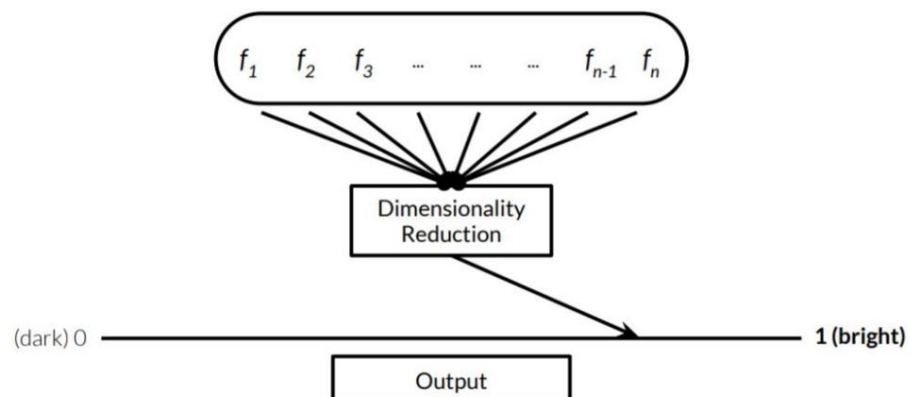
## Reducción de dimensionalidad algorítmica

### Linear dimensionality reduction

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

- Además de reducir manualmente la dimensionalidad de sus conjuntos de datos, también puede aplicar varios enfoques algorítmicos para reducir la dimensionalidad. Veamos algunos de ellos ahora. Veamos las técnicas que puede usar para reducir la dimensionalidad automáticamente. Primero, desarrollemos algo de intuición sobre cómo funciona realmente la reducción de la dimensionalidad lineal .
- En este enfoque, usted proyecta linealmente datos  $n$  dimensionales en un subespacio  $k$  dimensional más pequeño. Aquí,  $k$  suele ser mucho más pequeño que  $n$ .
- Hay infinitos subespacios dimensionales en los que podemos proyectar datos. ¿Qué subespacio tenemos?  
¿escoger?

## Projecting onto a line



- Para entender cómo se eligen los subespacios, demos un paso atrás y veamos cómo se pueden proyectar datos en una línea.
- Para empezar, pensemos en las características como vectores que existen en un espacio de alta dimensión. Visualizarlos revelaría mucho sobre la distribución de los datos, aunque es imposible para nosotros, los humanos, ver tantas dimensiones a la vez. En su lugar, debe proyectar los datos en una dimensión más baja, lo que podría permitirle visualizar los datos más directamente. Este tipo de proyección se llama incrustación.
- Calcular esto requiere tomar cada muestra y calcular un solo número para describirla. - Una ventaja de reducir a una dimensión es que los números y los ejemplos se pueden ordenar en una línea.
- En este ejemplo, tomamos imágenes y reducimos la información que contienen a una sola dimensión: su brillo promedio de píxeles, que luego podemos visualizar como un punto en una línea.

## 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)

- Volviendo a los subespacios, hay varias formas de elegir estos subespacios kdimensionales. - Por ejemplo, en una prueba de clasificación, normalmente desea tener la máxima separación entre clases. - El análisis discriminante lineal, o LDA, generalmente funciona bien para eso. - Para la regresión, desea maximizar la correlación entre los datos proyectados y la salida, donde

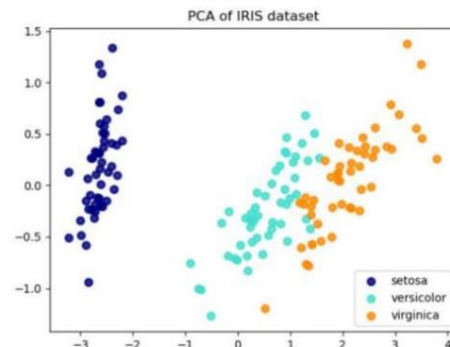
Los mínimos cuadrados parciales, o PLS, funcionan

- bien. Por último, y en tareas no supervisadas, por lo general queremos conservar la mayor cantidad posible de varianza. El análisis de componentes principales, o PCA, es la técnica más utilizada para hacerlo.

## Análisis de componentes principales

## Principal component analysis (PCA)

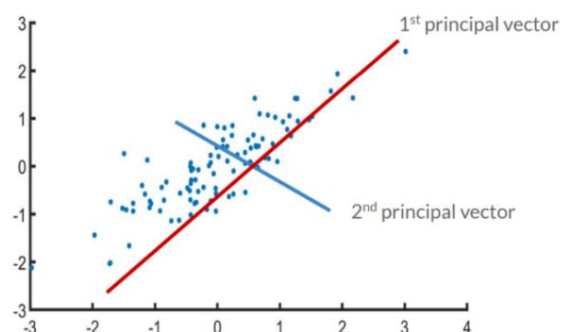
- PCA is a minimization of the orthogonal distance
- Widely used method for unsupervised & linear dimensionality reduction
- Accounts for variance of data in as few dimensions as possible using linear projections



- Hay varios enfoques algorítmicos para hacer reducción de dimensionalidad y componente principal análisis, o PCA es uno de los más utilizados.
- Para empezar, veamos cómo funciona el análisis de componentes principales o PCA.
  - Este es un algoritmo no supervisado que crea combinaciones lineales de las características originales. - PCA realiza la reducción de dimensionalidad en dos pasos, comenzando con la descorrelación, donde no cambia la dimensionalidad de los datos en absoluto. En
  - el primer paso, PCA rota las muestras para que estén alineadas con los ejes de coordenadas. De hecho, hay más que esto. PCA también cambia las muestras para que tengan una media de cero. - Estos diagramas de dispersión muestran el efecto de PCA aplicado a tres características del conjunto de datos de IRIS.

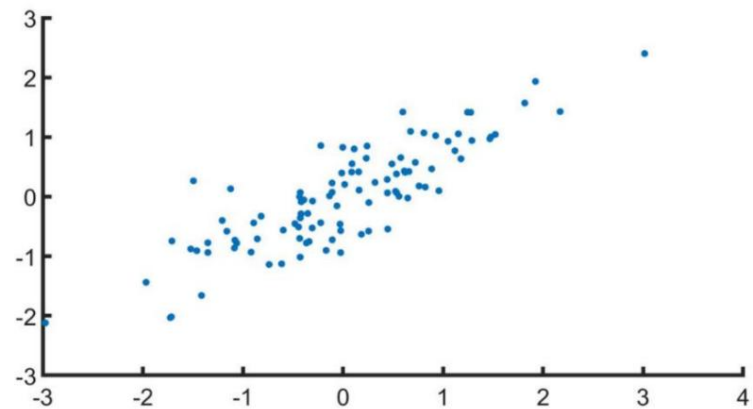
## Principal components (PCs)

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



- Finalmente, PCA se llama análisis de componentes principales porque aprende los componentes principales de los datos.
- Estas son las direcciones en las que las muestras varían más, representadas aquí en rojo.
- Son los componentes principales los que PCA alinea con el eje de coordenadas. El objetivo de PCA es que trata de encontrar una superficie dimensional más baja sobre la cual proyectar los datos, para minimizar el error de proyección al cuadrado. En otras palabras, para minimizar el cuadrado de la distancia entre cada punto y la ubicación de donde se proyecta.
- El resultado será maximizar la varianza de las proyecciones. - El primer componente principal es la dirección de proyección que maximiza la varianza de los datos proyectados. - La segunda componente principal es la dirección de proyección que es ortogonal a la primera principal y maximiza la varianza restante de los datos proyectados.

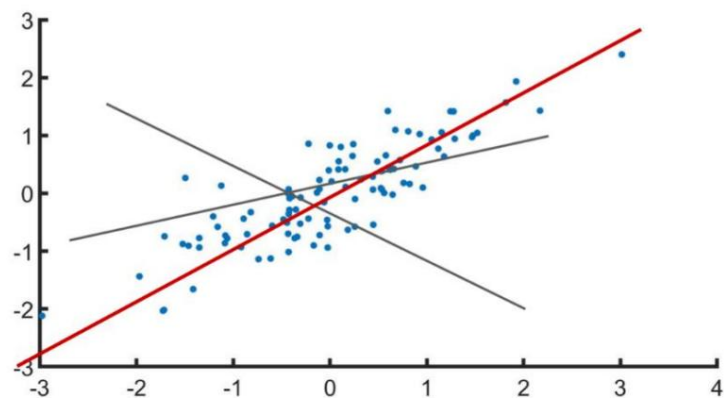
## 2-D data



- He aquí un ejemplo de juguete que consiste en una nube de puntos en 2D. Intentemos aplicar PCA a este bidimensional datos y ver qué pasa.

## PCA Algorithm - First Principal Component

Step 1



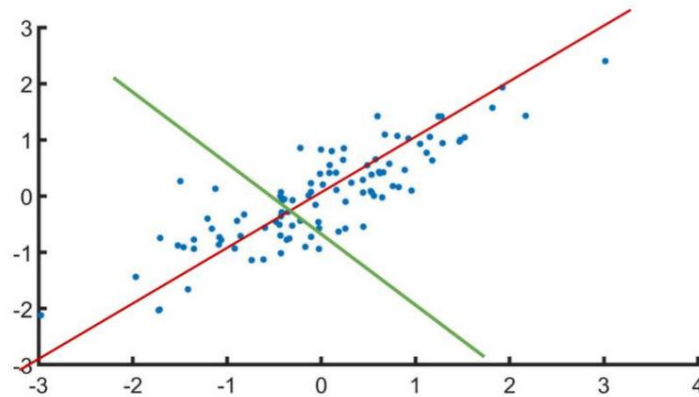
Find a line, such that when the data is projected onto that line, it has the maximum variance

- El primer componente principal es una dirección de proyección que maximiza la varianza de los datos proyectados.
- En la trama, puedes ver tres intentos de producir tal línea. En este caso, es
- bastante obvio que la varianza se maximiza en la dirección indicada por la línea roja.



## PCA Algorithm - Second Principal Component

Step 2

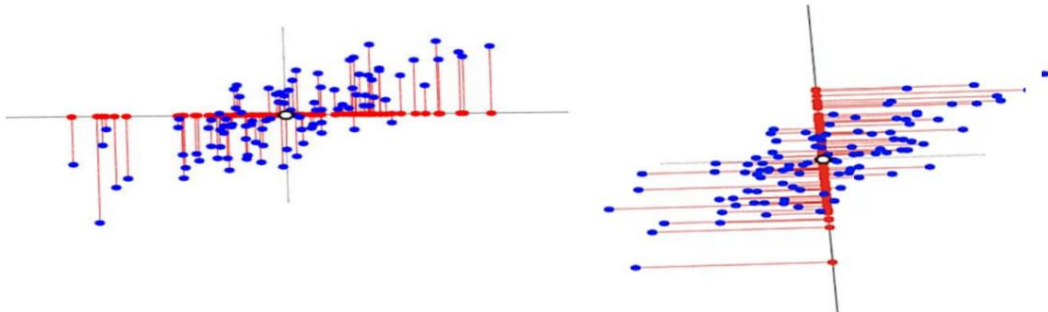


Find a second line, orthogonal to the first, that has maximum projected variance

- El segundo componente principal es la dirección de proyección que es ortogonal al primer componente principal y maximiza la varianza restante de los datos proyectados indicados aquí por la línea verde.

## PCA Algorithm

Step 3



Repeat until we have k orthogonal

- El conjunto completo de componentes principales comprende una nueva base ortogonal para el espacio de características cuyos ejes siguen las variaciones máximas de los datos originales.
- Estas proyecciones se transforman simplemente al nuevo espacio reducido kdimensional.
- Esto significa que cuando está proyectando sus datos originales en los primeros k componentes principales, está reduciendo la dimensionalidad de los datos. Posteriormente, se puede recuperar el espacio original a partir de esta proyección de dimensionalidad reducida. Esta reconstrucción, por supuesto, tendrá una cierta cantidad de error, pero esto a menudo es insignificante y aceptable dados los otros beneficios de la reducción de la dimensionalidad. Además, observe cómo cambian los puntos rojos a medida que gira la línea. Esa es la varianza.
- ¿Puedes ver cuando alcanza su máximo? - En segundo lugar, si reconstruimos las dos características originales, los puntos azules, a partir de las nuevas, los puntos rojos, el error de reconstrucción vendrá dado por la longitud de la línea roja que las conecta. - Observa cómo cambia la longitud de las líneas rojas mientras la línea gira. ¿Puedes ver dónde está la longitud total? llega a un mínimo?

## Applying PCA on Iris



- Aquí aplicamos el algoritmo PCA con dos componentes principales en el conjunto de datos IRIS y visualizamos los resultados.

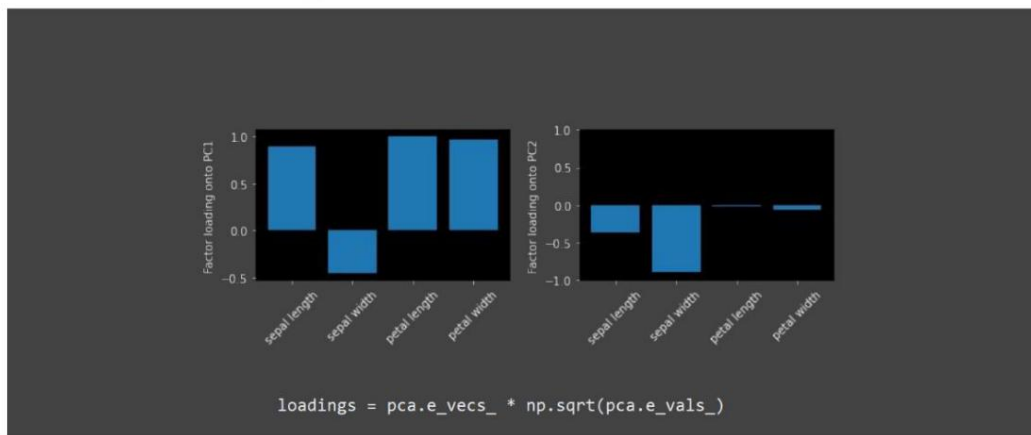
## Plot the explained variance



- Ahora, suponiendo que haya aplicado PCA nuevamente usando cuatro componentes en lugar de dos, visualicemos cuánto la varianza se ha explicado usando estos cuatro componentes. Si observa
  - la varianza relativa, es posible que pierda algo de información, pero si los valores propios son pequeños, no se pierde mucha información.
  - Los componentes principales son de naturaleza ortogonal como hemos visto antes y esto significa que no están correlacionados. Además, se clasifican en orden de su varianza explicada. El primer componente principal explica la mayor variación en el conjunto de datos, y el segundo explica la segunda mayor variación y así sucesivamente. Por lo tanto, puede reducir la dimensionalidad limitando la cantidad de componentes principales que se deben mantener, en función de la varianza acumulada explicada.
- Por ejemplo, puede decidir mantener solo tantos componentes principales como sean necesarios para alcanzar una varianza explicada acumulada del 90 por ciento.



## PCA factor loadings



- Las cargas factoriales son los valores no estandarizados de los vectores propios. - Podemos interpretar las cargas como covarianzas o correlaciones.

## 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)
```

- Scikitlearn tiene una implementación de PCA que incluye métodos de ajuste y transformación, al igual que la operación de escalador estándar, así como un método de ajuste y transformación que combina ajuste y transformación.
- El método de ajuste aprende a desplazar y rotar las muestras, pero en realidad no las cambia. - El método de transformación, por otro lado, aplica la transformación aprendida por el ajuste. En particular, el método de transferencia se puede aplicar a nuevas muestras no vistas.
- Antes de aplicar PCA, usemos el escalador estándar en las características.

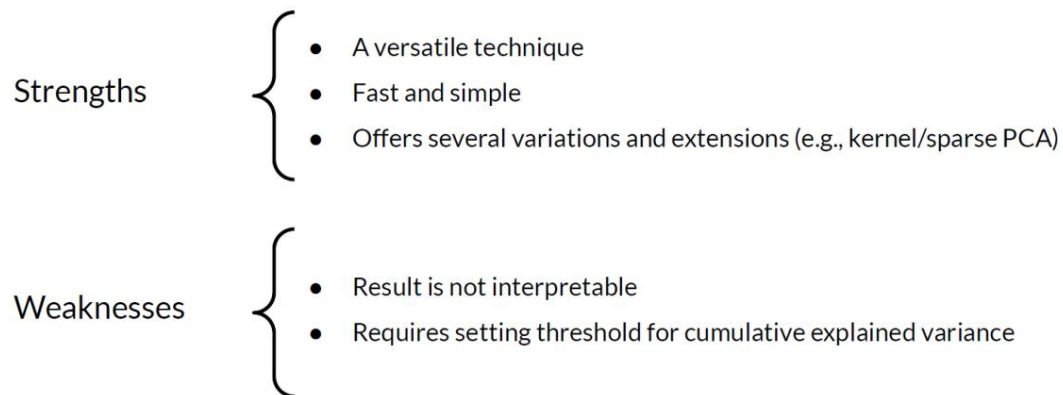
## PCA in scikit-learn

```
# 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)
```

- Este código crea una instancia de PCA que retendrá el 99 por ciento de la varianza ajustada a los datos y aplicará la transformación lo aprendido.

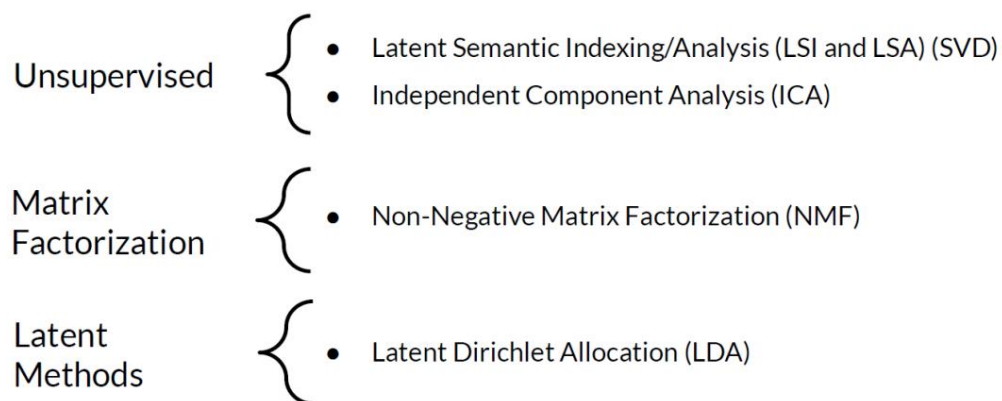
## When to use PCA?



- Resumiendo todo eso. PCA es una técnica útil que funciona bien en la práctica.
- Es rápido y sencillo de implementar, lo que significa que puede probar fácilmente algoritmos con y sin PCA para comparar el rendimiento. Además, PCA ofrece varias variaciones y extensiones. Por ejemplo, kernel PCA o sparse PCA, etc., para abordar obstáculos específicos.
- Sin embargo, los componentes principales resultantes no son interpretables, lo que puede ser un factor decisivo en algunos escenarios donde la interpretabilidad es importante.
- Además, aún debe establecer o ajustar manualmente un umbral para la variación explicada acumulada. - Aparte de esto, PCA es especialmente útil cuando se estudian visualmente grupos de observaciones en dimensiones altas. Esto podría ser cuando todavía está explorando los datos. - Por ejemplo, es posible que tenga razones para creer que los datos tienen un rango inherentemente bajo, lo que significa que hay son muchos atributos, pero solo unos pocos atributos que en su mayoría determinan el resto a través de una asociación lineal.

## Otras Técnicas

### More dimensionality reduction algorithms



- Además de las técnicas que ya hemos analizado, existen varios enfoques algorítmicos más para realizar la reducción de la dimensionalidad. Veamos algunos de esos ahora
- Algunas técnicas se centran en tipos particulares de problemas. Por ejemplo, siguiendo con los enfoques no supervisados, existen técnicas como la descomposición en valor único o SVD, y el análisis de componentes independientes o ICA. En las técnicas de factorización de matrices, puede utilizar la factorización de matrices no negativa (NMF). Y finalmente, Latent Dirichlet Allocation o LDA es uno de los métodos de reducción de dimensionalidad latente más populares.

## Singular value decomposition (SVD)

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

- Analicemos la descomposición de valor único o SVD.
- Las matrices pueden verse como transformaciones lineales en el espacio. PCA, que discutimos anteriormente se basa en eigen descomposición, que solo se puede hacer para matrices cuadradas
- Por supuesto, no siempre tienes matrices cuadradas. En TFIDF, por ejemplo, una alta frecuencia de términos puede En realidad no será fructífera, en algunos casos, las palabras raras aportan más. En
- general, la importancia de las palabras aumenta si también aumenta el número de ocurrencias de estas palabras dentro del mismo documento.
- Por otro lado, se disminuirá la importancia de las palabras que aparecen con frecuencia en el corpus. - Los desafíos que la matriz resultante es muy escasa y no cuadrada. - Para descomponer este tipo de matrices, que no se pueden descomponer con eigendecomposition, podemos usar técnicas como la descomposición en valores singulares o SVD.
- SVD descompone nuestro conjunto de datos original en sus componentes, lo que resulta en una reducción de la dimensionalidad
- Se utiliza para eliminar características redundantes del conjunto de datos.

## 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

- El análisis de componentes independientes, o ICA, es otro algoritmo y se basa en la teoría de la información. Es también una de las técnicas de reducción de dimensionalidad más utilizadas.
- La diferencia significativa de PCA e ICA es que PCA busca factores no correlacionados, mientras que ICA busca factores independientes. Si
- dos factores no están correlacionados, significa que no existe una relación lineal entre ellos. Si son independientes,
- significa que no dependen de otras variables.
- Por ejemplo, la edad de una persona es independiente de lo que come o de la cantidad de televisión que ve. relojes

## How does ICA work?

- Assume there exists independent signals:

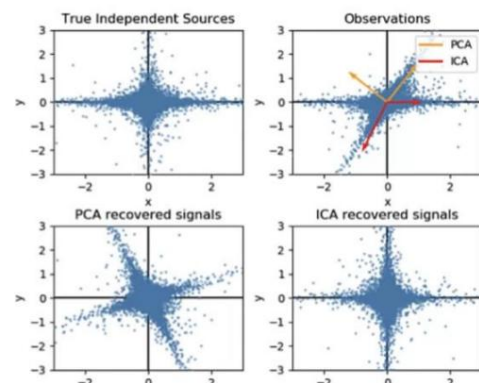
$$S = [s_1(t), s_2(t), \dots, s_N(t)]$$

- 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  $Y(t)$

- El análisis de componentes independientes separa una señal multivariante en componentes aditivos que son máximamente independiente. A menudo, ICA no se usa para reducir la dimensionalidad sino para separar señales superpuestas. Dado que el modelo no incluye un término de ruido, para que el modelo sea correcto, se debe aplicar un blanqueamiento. Esto se puede hacer de varias maneras, incluido el uso de una de las variantes de PCA.
- ICA asume además que existen señales independientes,  $S$ , y una combinación lineal de señales,  $Y$ .
- El objetivo de ICA es recuperar las señales originales,  $S$ , de  $Y$  - ICA asume que las variables dadas son mezclas lineales de algunas variables latentes desconocidas.
- También supone que estas variables latentes son mutuamente independientes. En otras palabras, no dependen de otras variables y, por lo tanto, se denominan componentes independientes de los datos observados. Comparemos PCA e ICA visualmente para comprender mejor en qué se diferencian.

## Comparing PCA and ICA

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



- Ambas son transformaciones estadísticas, es decir, PCA utiliza información extraída de estadísticas de segundo orden, mientras que ICA sube a estadísticas de orden superior.
- Ambos se utilizan en varios campos, como la separación ciega de fuentes, la extracción de características y también en neurociencia. - ICA es un algoritmo que encuentra direcciones en el espacio de características correspondientes a proyecciones que son altamente no gaussianas.
- A diferencia de PCA, estas direcciones no necesitan ser ortogonales en el espacio de características original, pero son ortogonales en el espacio de características blanqueadas, en el que todas las direcciones corresponden a la misma varianza.
- PCA, por otro lado, encuentra direcciones ortogonales en el espacio de características sin procesar que corresponden a direcciones teniendo en cuenta la varianza máxima.
- Veamos una simulación de dos fuentes independientes usando un proceso altamente no gaussiano a la izquierda (Imagen superior izquierda)

- A continuación, apliquemos un esquema de mezcla para crear observaciones, en este espacio de observación sin procesar, las direcciones identificadas por PCA están representadas por vectores naranjas. (imagen superior derecha)
- Luego, representemos la señal en el espacio PCA después del blanqueamiento por la varianza correspondiente al PCA vectores (Imagen inferior izquierda)
- Ejecutar ICA corresponde a encontrar una rotación en este espacio para identificar las direcciones que son más no gaussianas. (Imagen inferior derecha)
- En la figura inferior derecha, puede ver que PCA elimina las correlaciones pero no la dependencia de orden superior.
- Por el contrario, ICA elimina las correlaciones junto con la dependencia de orden superior. - Cuando se trata de la importancia de los componentes, PCA considera que algunos de ellos son más importantes que otros. ICA, por otro lado, considera que todos los componentes son igualmente importantes.

## Non-negative Matrix Factorization (NMF)

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

genfaces - PCA using randomized SVD - Train time 0.0



Non-negative components - NMF - Train time 0.1s



- Ahora, hablemos de una técnica de reducción de dimensionalidad llamada factorización matricial no negativa o NMF. - NMF expresa muestras como una combinación de partes interpretables. - Por ejemplo, representa documentos como combinaciones de temas e imágenes en términos de

patrones visuales que ocurren.

- NMF, como PCA, es una técnica de reducción de dimensionalidad
- Sin embargo, a diferencia de PCA, los modelos NMF son interpretables. Esto
- significa que los modelos NMF son más fáciles de entender y mucho más fáciles de explicar a los demás. - NMF no se puede aplicar a todos los conjuntos de datos; sin embargo, requiere que las características de la muestra no sean negativas, por lo que los valores deben ser mayores o iguales a cero. Se ha observado que, cuando se restringe cuidadosamente, NMF puede producir una
- representación basada en partes del conjunto de datos, lo que da como resultado modelos interpretables. Este ejemplo muestra 16 componentes dispersos encontrados por NMF a partir de las imágenes en el conjunto de [datos de caras de Olivetti](#) a la derecha, en
- comparación con las caras propias de PCA a la izquierda.

## Optimización de modelos: móvil, IoT y casos de uso similares

### Trends in adoption of smart devices



- La optimización del modelo es otra área de enfoque en la que puede optimizar aún más el rendimiento y los recursos.  
requisitos
- El objetivo es crear modelos que sean lo más eficientes y precisos posible y lograr la mayor  
rendimiento al menor costo. Veamos algunas técnicas avanzadas para eso ahora. Comencemos analizando  
algunos de los problemas relacionados con las aplicaciones móviles, IoT e integradas. - El aprendizaje automático se  
está convirtiendo cada vez más en parte de más y más dispositivos y productos. Esto incluye el rápido crecimiento de las aplicaciones  
móviles y de IoT, incluidos los dispositivos que se encuentran en todas partes, desde campos de agricultores hasta vías de tren.
- Las empresas están utilizando los datos que generan estos dispositivos para entrenar modelos de aprendizaje automático para mejorar sus  
procesos comerciales, productos y servicios.
- Incluso los anunciantes digitales gastan más en dispositivos móviles que en  
computadoras de escritorio. - Ya hay miles de millones de dispositivos informáticos móviles y de borde, y ese número seguirá creciendo  
rápidamente en la próxima década.
- McKinsey predice que para 2025, el impacto económico general de IoT y la tecnología móvil podría alcanzar billones de dólares, superando  
a muchos sectores como la automatización del trabajo del conocimiento o la tecnología en la nube. - A medida que estos dispositivos  
se vuelvan cada vez más ubicuos y potentes, muchas de las tareas de aprendizaje automático, que usted cree que requieren meses de  
tiempo de cómputo de alta potencia, se volverán parte de dispositivos cada vez más comunes.

### Factors driving this trend

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

- Ahora veamos algunas de las razones por las que esas tendencias ocurren en primer lugar. Tradicionalmente, puede pensar en implementar  
modelos de aprendizaje automático en la nube. Esto requiere un servidor para ejecutar la inferencia y devolver los resultados. Pero con  
el avance de la investigación de aprendizaje automático para aplicaciones en dispositivos de menor potencia, este procesamiento  
puede descargarse en un dispositivo y ejecutarse localmente. Esto permite más oportunidades para incluir el aprendizaje automático  
como parte de la funcionalidad principal de un dispositivo.



- Además, los costos de hardware para estos dispositivos continúan cayendo, lo que permite puntos de precio más bajos y volúmenes más altos.
- Un aspecto clave del aprendizaje automático en el dispositivo es que, en la mayoría de los casos, garantiza un mayor cumplimiento de la privacidad . regulaciones al mantener los datos del usuario en el dispositivo.

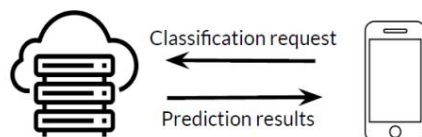
## Online ML inference

- 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?

- ¿Cómo debe implementar sus modelos para que puedan usarse para generar valor real?
- Si los aloja en un servidor, el dispositivo debe estar conectado para que pueda realizar una solicitud de red.
- Otra opción es incrustar el modelo en un dispositivo móvil directamente.
- En tu caso, ¿puedes confiar siempre en que el dispositivo tenga conexión a la red?
- ¿También es su modelo lo suficientemente pequeño y rápido para realizar inferencias en el dispositivo? Además, los dispositivos móviles ofrecen capacidades de procesamiento limitadas, lo que puede afectar los tipos de modelos que puede insertar en ellos.
- Además, ¿el dispositivo tiene todo el acceso que necesita a los datos que necesita? O necesita cosas
- ¿Te gustan los datos históricos que solo están disponibles en un servidor?

## 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

- Suponga que está tratando de crear una aplicación que aplica diferentes efectos a las fotos.
- En un escenario donde los modelos están alojados en un servidor, la aplicación primero debe enviar la foto al servidor, y luego el servidor alimenta la imagen a través de un modelo para aplicar el efecto deseado, y unos segundos más tarde, envía la imagen modificada. a la aplicación
- El uso de un servidor para la inferencia tiene la ventaja de que mantiene la aplicación móvil simple. - El servidor encapsula toda la complejidad del modelo. Esto significa que puede actualizar el modelo o agregar nuevas características en cualquier momento que desee.
- Para implementar el modelo mejorado, actualice el modelo en el servidor. Eso significa que probablemente no tenga que actualizar la aplicación en sí, a menos que necesite cambiar la solicitud que envía la aplicación.
- Un gran inconveniente es que la inferencia oportuna es un requisito importante en este entorno.



# Mobile inference

## On-device Inference



### Pro



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

### Cons

- Less capacity
- Tight resource constraints

- En caso de inferencia en un dispositivo, carga el modelo entrenado en la aplicación. Dado
- que el modelo se ejecuta en la aplicación, no necesita enviar una solicitud a través de Internet y esperar una respuesta. En cambio, la
- predicción ocurre rápido y no necesita una conexión de red.
- Hay una demanda creciente de servicios sofisticados habilitados para IA, como reconocimiento de imagen y voz,
- procesamiento de lenguaje natural, búsqueda visual y recomendaciones personalizadas. Al mismo
- tiempo, los conjuntos de datos están creciendo. Las redes son cada vez más complejas. La privacidad se está convirtiendo cada vez
- más en un problema y los requisitos de latencia se están ajustando para satisfacer las expectativas de los usuarios.
- Todas estas tendencias influyen en la elección de dónde generar predicciones a partir de sus modelos entrenados, lo que a su vez
- afecta la arquitectura y la complejidad de los modelos que entrena.

## 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 *	✓	✓	✓		✓	✓

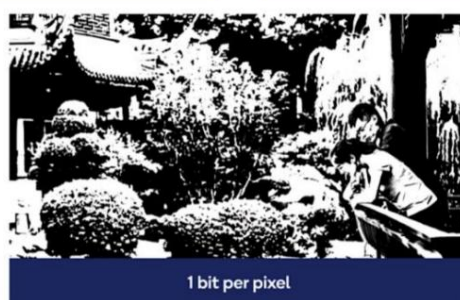
\* Also supported in TFX

- Ahora echemos un vistazo a algunas de las opciones disponibles en la actualidad para implementar modelos en aplicaciones móviles. - ML Kit para Firebase ofrece API listas para usar. También puede implementar sus propios modelos de TensorFlow Lite si no encuentra una API base que cubra su caso de uso. Se dirige a plataformas móviles y utiliza TensorFlow Lite, la API de Google Cloud
- Vision y la API de redes neuronales de Android para proporcionar aprendizaje automático en el dispositivo, como reconocimiento facial, escaneo de códigos de barras y detección de objetos, entre otros. - ML Kit le brinda API en el dispositivo y en la nube, lo que significa que también puede usar las API cuando no hay red
- conexión.
- Las API basadas en la nube utilizan Google Cloud Platform. - Con ML Kit, puede
- cargar modelos a través de Firebase Console y dejar que el servicio se encargue del alojamiento.
- y servir los modelos a los usuarios de sus aplicaciones.

- Otra ventaja es que dado que ML Kit está disponible a través de Firebase, es posible aprovechar la plataforma Firebase más amplia.
- Con Core ML, puede construir su modelo o usar un modelo previamente entrenado. Para usar su modelo, primero debe crear un modelo utilizando marcos de trabajo de terceros. - Luego convierte su modelo al formato de modelo Core ML. Los marcos admitidos incluyen Scikit-learn, Keras, Caffe y XGBoost. También hay algunos modelos preentrenados listos para usar.
- TensorFlow Lite fue desarrollado por Google y tiene API para muchos lenguajes de programación, incluido Java, C++, Python, Swift y ObjectiveC. Está optimizado para aplicaciones en dispositivos y proporciona un intérprete ajustado para el aprendizaje automático en dispositivos.
- Los modelos personalizados se convierten al formato TensorFlow Lite y su tamaño se optimiza para aumentar la eficiencia. - TensorFlow Lite también admite la optimización de modelos para IoT y aplicaciones integradas, para dispositivos con poco como 20k de memoria.
- Los modelos TensorFlow Lite se pueden entrenar y evaluar en canalizaciones TFX, lo cual es importante cuando el modelo se convertirá en parte de una producción, producto o servicio.

## Beneficios y proceso de cuantificación

### Quantization



- Ahora analicemos algunas técnicas para optimizar sus modelos. Especialmente para escenarios de implementación, como dispositivos móviles e IoT, donde las capacidades del dispositivo son extremadamente limitadas en comparación con la ejecución en un servidor o en la nube.
- Sin embargo, antes de hacerlo, aclaremos que estas técnicas pueden beneficiar a cualquier modelo sin importar dónde se encuentre implementadas, ya que reducen los recursos informáticos necesarios para servir el modelo.
- La cuantificación implica transformar un modelo en una representación equivalente que utiliza parámetros y cálculos con menor precisión. Esto mejora el rendimiento y la eficiencia de la ejecución del modelo, pero a menudo puede resultar en un modelo más bajo .
- Usemos una analogía para entender esto mejor. Piensa en una imagen. Como sabrá, una imagen es una cuadrícula de píxeles donde cada píxel tiene una cierta cantidad de bits.

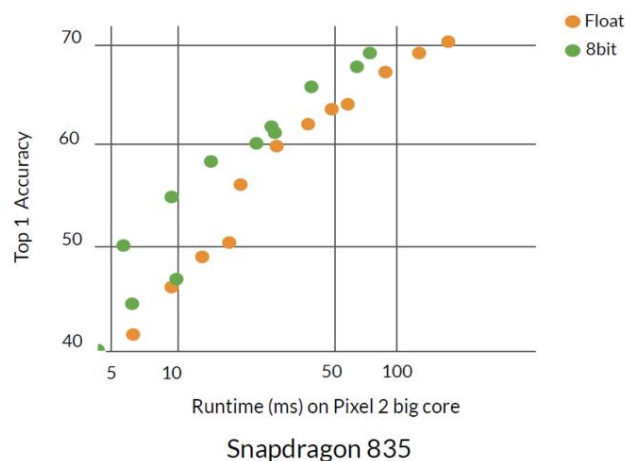
- Ahora, si intenta reducir el espectro de color continuo de la vida real a colores discretos, estamos cuantificando o aproximando la imagen. En esta
- animación, puede ver que una imagen en blanco y negro se puede representar con un bit por píxel, mientras que una imagen típica en color tiene 24 bits por píxel.
- La cuantificación, en esencia, disminuye o reduce el número de bits necesarios para representar la información. - Sin embargo, puede notar que a medida que reduce la cantidad de píxeles más allá de cierto punto, dependiendo de la imagen, puede ser más difícil reconocer cuál es esa imagen.

## 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

- Los modelos de redes neuronales pueden ocupar mucho de este espacio. Por ejemplo, AlexNet requiere alrededor de 200 megabytes de espacio en disco.
- Casi todo ese tamaño se ocupa con los pesos de las conexiones neuronales, ya que a menudo hay muchos millones de estos en un solo modelo.
- Debido a que todos son números de coma flotante ligeramente diferentes, la compresión simple no funciona. comprimirlos bien a menos que hagamos modelos menos densos.
- La motivación más sencilla para la cuantificación es reducir el tamaño de los archivos. Para las aplicaciones móviles, especialmente, a menudo no es práctico almacenar un modelo de 200 megabytes en el teléfono solo para ejecutar una sola aplicación. Por lo tanto, es necesario comprimir modelos de mayor precisión.
- Otra razón para cuantificar es reducir los recursos computacionales que necesita para hacer inferencias cálculos ejecutándolos completamente con entradas y salidas de baja precisión. Esto es mucho más
- difícil ya que requiere cambios en todos los lugares donde realiza los cálculos, pero ofrece recompensas potenciales.
- Hacer esto lo ayudará a ejecutar sus modelos más rápido y usar menos energía, lo cual es especialmente importante en dispositivos móviles.
- También abre la puerta a una gran cantidad de sistemas integrados que no pueden ejecutar punto flotante de manera eficiente, lo que permite muchas aplicaciones en el mundo de IoT.
- Ahora, echemos un vistazo a lo que significa cuantización.

## MobileNets: Latency vs Accuracy trade-off



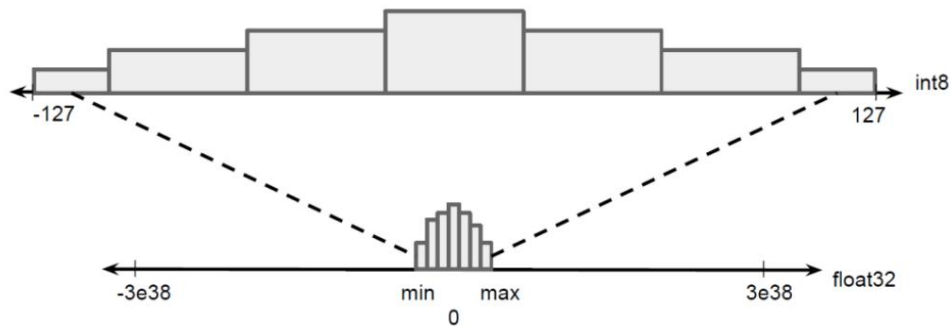
- MobileNets son una familia de arquitecturas que logran un compromiso avanzado entre la latencia en el dispositivo y la precisión de clasificación de ImageNet. - Una publicación reciente demostró cómo la cuantificación de números enteros podría mejorar aún más la compensación en ferretería común.
- Los autores del artículo compararon la arquitectura de MobileNet con multiplicadores de profundidad y resoluciones variables en ImageNet en tres tipos de núcleos de Qualcomm. Este gráfico es para el chip Snapdragon 835. - Puede ver que para cualquier nivel de precisión dado, el tiempo de latencia es menor para la versión de 8 bits del modelo.

## Benefits of quantization

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

- La aritmética con una profundidad de bits más baja es más rápida, suponiendo que el hardware la admita. - Aunque el cálculo de coma flotante ya no es más lento que el de enteros en las CPU modernas, las operaciones con coma flotante de 32 bits casi siempre serán más lentas que, por ejemplo, los enteros de ocho bits. - Pasando de 32 bits a ocho bits, generalmente obtenemos aceleraciones de 4x de reducción en la memoria.
- Los modelos de implementación más livianos significan que tienen menos espacio de almacenamiento y son más fáciles de compartir en anchos de banda más pequeños y más fáciles de actualizar.
- Las profundidades de bits más bajas también significan que podemos comprimir más datos en los mismos cachés y registros. Esto hace posible crear aplicaciones con mejores capacidades de almacenamiento en caché que reducen el uso de energía y se ejecutan más rápido. La aritmética de coma flotante es difícil, por lo que es posible que no siempre sea compatible con los microcontroladores y con algunos dispositivos integrados de potencia ultrabaja, como drones, relojes o dispositivos IoT. - El soporte de enteros, por otro lado, siempre está disponible.

## The quantization process



- Las redes neuronales consisten en nodos de activación, las conexiones entre los nodos, un parámetro de peso asociado con cada conexión y un término de sesgo. - Cuando se trata de cuantificar estas redes, son estos parámetros de peso y los cálculos del nodo de activación los que estamos tratando de cuantificar. - La cuantificación comprime un pequeño rango de valores de punto flotante en un número fijo de cubos de información como

se puede ver en este diagrama. Este

- proceso es de naturaleza con pérdidas, pero los pesos y las activaciones de una capa en particular a menudo tienden a estar en un rango pequeño, que se puede estimar de antemano. Esto significa que no necesitamos la capacidad de almacenar un rango en el mismo tipo de datos, lo que nos permite concentrar nuestros valiosos bits en un rango más pequeño. Decir tres negativos a tres positivos.
- Como puede imaginar, será crucial conocer con precisión este rango más pequeño. Si se hace correctamente, la cuantización provoca solo una pequeña pérdida de precisión, que normalmente no cambia la salida de forma significativa.

## What parts of the model are affected?

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

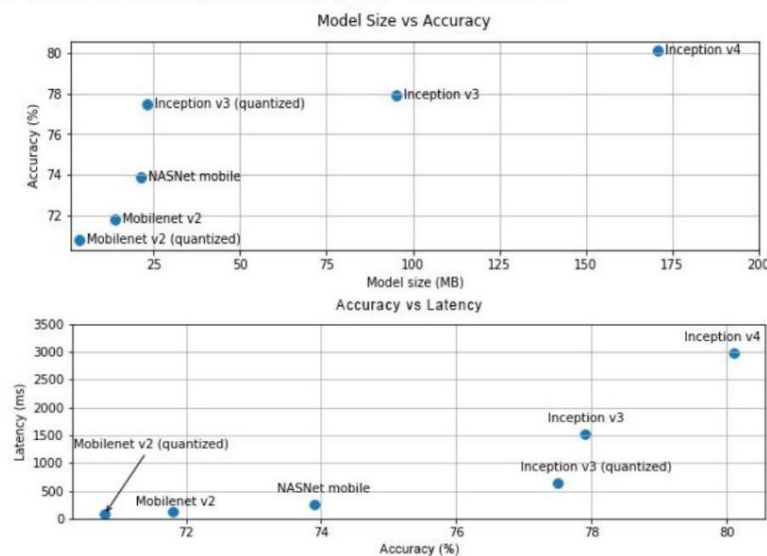
- Ahora, veamos qué partes de un modelo se ven afectadas después de aplicar la cuantización. - Uno de ellos podría ser parámetros estáticos como los pesos de las capas, y otros podrían ser dinámicos como activaciones dentro de las redes.
- También puede tener transformaciones como agregar, modificar o eliminar operaciones, fusionar diferentes operaciones, etc. En algunos casos, las transformaciones pueden necesitar datos adicionales. Verá cómo este es el caso en una de las técnicas de cuantificación
- donde se utilizan algunos datos no etiquetados para determinar los parámetros de escala.

## 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

- Las optimizaciones a menudo pueden dar lugar a cambios en la precisión del modelo, que deben tenerse en cuenta durante el proceso de desarrollo de aplicaciones.
- Los cambios de precisión dependen del modelo individual y de los datos que se están optimizando y son difíciles de predecir antes de tiempo
- En general, los modelos que están optimizados para el tamaño y la latencia perderán cierta precisión.
- Dependiendo de su aplicación, esto puede o no afectar la experiencia de su usuario.
- En casos raros, ciertos modelos pueden ganar algo de precisión como resultado del proceso de optimización. En términos de interpretabilidad, hay algunos efectos que pueden imponerse al modelo después de la cuantificación. Esto significa que es difícil evaluar si la transformación de una capa iba en la dirección correcta o incorrecta.

## Choose the best model for the task



- Los dispositivos móviles e integrados tienen recursos computacionales limitados, por lo que es importante mantener su aplicación eficiente en recursos.
- Según la tarea, deberá hacer un balance entre la precisión del modelo y la complejidad del modelo.
- Si su tarea requiere una alta precisión, es posible que necesite un modelo grande y complejo. - Para tareas que requieren menos precisión, es mejor usar un modelo más pequeño y menos complejo. - Porque no solo usan menos espacio en disco en la memoria, sino que también son generalmente más rápidos y consumen más energía eficiente.
- Por ejemplo, estos gráficos muestran las compensaciones de precisión y latencia para algunas clasificaciones de imágenes comunes. modelos
- Un ejemplo de modelos optimizados para dispositivos móviles son MobileNets, que están optimizados para aplicaciones de visión móvil. Una vez que haya seleccionado un modelo candidato adecuado para su tarea, es una buena práctica perfilar y comparar su modelo.

## Post-training quantization

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



- Puede realizar la cuantificación durante el entrenamiento o después de que el modelo haya sido entrenado.
- Veamos, primero, la cuantización posterior al entrenamiento. La cuantificación posterior al entrenamiento es una técnica de conversión que puede reducir el tamaño del modelo al mismo tiempo que mejora la latencia del acelerador de CPU y hardware con poca degradación en la precisión del modelo.
- Puede cuantificar un modelo TensorFlow ya entrenado cuando lo convierte al formato TensorFlow Lite usando el convertidor TensorFlow Lite.
- Es fácil de usar ya que está integrado directamente en el convertidor TFLite. - Lo que hace básicamente la cuantificación posterior al entrenamiento es convertir, o más precisamente, cuantificar los pesos de números de punto flotante a enteros de una manera eficiente.
- Al hacer esto, puede obtener una latencia hasta tres veces menor sin sufrir un gran impacto en la precisión. - Con la estrategia de optimización predeterminada, el convertidor hará todo lo posible para aplicar una cuantificación posterior al entrenamiento, tratando de optimizar el modelo tanto en tamaño como en latencia. Esto se recomienda, aunque siempre puede personalizar este comportamiento.

## 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

- Hay varias opciones de cuantificación posterior al entrenamiento para elegir. Este es un resumen
- de las opciones y los beneficios que brindan. Si está buscando un aumento de velocidad
- decente, como 23 veces más rápido, mientras que es dos veces más pequeño, puede considerar la cuantificación del rango dinámico. - Por otro lado, si desea expresar aún más el rendimiento de su modelo, entonces entero completo
- la cuantificación o la cuantificación float16 pueden dar como resultado un rendimiento más rápido.
- Float16 es especialmente útil cuando planea usar una GPU. - Con la cuantificación
- de rango dinámico, durante la inferencia, los pesos se convierten de ocho bits a punto flotante y las activaciones se calculan utilizando núcleos de punto flotante. Esta conversión se realiza una vez y se almacena en caché para reducir la latencia. Esta optimización proporciona latencias cercanas a la inferencia de punto totalmente fijo.
- 
-



## Post training quantization

```
import tensorflow as tf

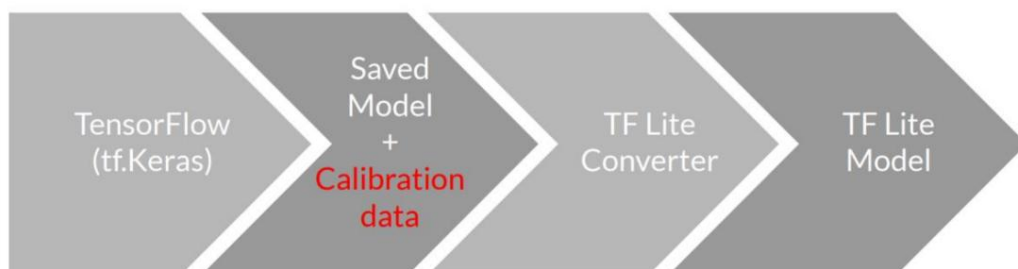
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)

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

tflite_quant_model = converter.convert()
```

- La cuantificación posterior al entrenamiento requiere solo dos líneas de código. Comencemos importando TensorFlow y definiendo un convertidor con TFLite.
- Luego configura el convertidor para optimizar el tamaño del modelo usando la opción `Optimize_for_size`. - Luego aplicas el convertidor a tu modelo. - Los otros modos de optimización disponibles incluyen `Optimize_for_latency`, que reduce la latencia de su modelo, mientras que el modo predeterminado básicamente intenta optimizarlo tanto para la velocidad como para el almacenamiento.
- Se pueden aplicar optimizaciones mejoradas proporcionando un conjunto de datos representativo.

## Post-training integer quantization



- Usando la cuantificación de rango dinámico, puede reducir el tamaño del modelo y/o la latencia, pero esto viene con un limitación ya que requiere que la inferencia se haga con números de coma flotante. Es posible que esto no siempre sea ideal, ya que algunos aceleradores de hardware solo admiten operaciones con números enteros, por ejemplo, Edge TPU.
- El kit de herramientas de optimización también es compatible con la cuantificación de enteros posterior al entrenamiento. Esto permite a los usuarios tomar un modelo de punto flotante ya entrenado y cuantificarlo completamente para usar solo un entero con signo de ocho bits, lo que permite que los aceleradores de hardware de punto fijo ejecuten estos modelos. - Al apuntar a mayores mejoras de CPU o aceleradores de punto fijo, esta suele ser una mejor opción.
- La cuantificación de enteros posterior al entrenamiento funciona mediante la recopilación de datos de calibración, lo que hace mediante la ejecución de inferencias en un pequeño conjunto de entradas para determinar los parámetros de escala correctos necesarios para convertir el modelo en un modelo cuantificado de enteros.

## 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



- La cuantificación posterior al entrenamiento puede resultar en una pérdida de precisión, particularmente para redes más pequeñas, pero a menudo es bastante insignificante.
- En el lado positivo, esto acelerará la ejecución de los cálculos más pesados mediante el uso de menor precisión y la los cálculos más sensibles con mayor precisión, por lo que generalmente resulta en poca o ninguna pérdida final de precisión.
- Los modelos preentrenados totalmente cuantificados también están disponibles para redes específicas en el modelo TensorFlow Lite repositorio. Es importante verificar la precisión del modelo cuantificado para verificar que cualquier degradación en la precisión esté dentro de los límites aceptables. TensorFlow Lite incluye una herramienta para evaluar la precisión del modelo. Alternativamente, si la pérdida de precisión es demasiado grande, considere usar un entrenamiento consciente de la cuantificación. Sin embargo, hacerlo requiere modificaciones durante el entrenamiento del modelo para agregar nodos de cuantificación falsos, mientras que las técnicas de cuantificación posteriores al entrenamiento son bastante simples.

### Entrenamiento consciente de la cuantización

## 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

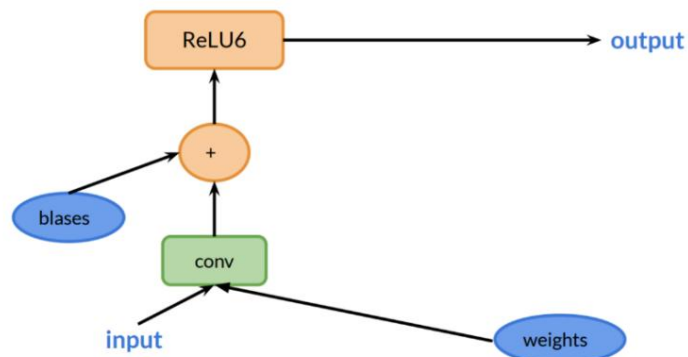
- Ahora echemos un vistazo a la formación consciente de la cuantificación. El enfoque más simple para cuantificar una red neuronal es primero entrenarlo con total precisión y luego simplemente cuantificar los pesos a un punto fijo. Esta es la cuantificación posterior al entrenamiento.
- Por el contrario, el entrenamiento consciente de la cuantificación aplica la cuantificación al modelo mientras se está entrenando. La idea central es que el entrenamiento consciente de la cuantización simula el cálculo del tiempo de inferencia de baja precisión en el paso adelante del proceso de formación.
- Al insertar nodos de cuantificación falsos, los efectos de redondeo de la cuantificación se asimilan en el avance pasar, como ocurriría normalmente en la inferencia real. El objetivo es afinar los pesos para ajustar la pérdida de precisión.
- Si se incluyen nodos de cuantificación falsos en el gráfico del modelo en los puntos donde se espera que ocurra la cuantificación, por ejemplo, convoluciones, entonces, en el paso hacia adelante, los valores flotantes se redondearán al número especificado de niveles para simular los efectos de la cuantificación. Esto introduce el error de cuantificación como ruido durante el entrenamiento y es parte de la pérdida total que el algoritmo de optimización intenta minimizar.
- Aquí, el modelo aprende parámetros que son más robustos a la cuantificación. A continuación, veamos el proceso de cuantificar un modelo durante el entrenamiento.

## Quantization-aware training (QAT)



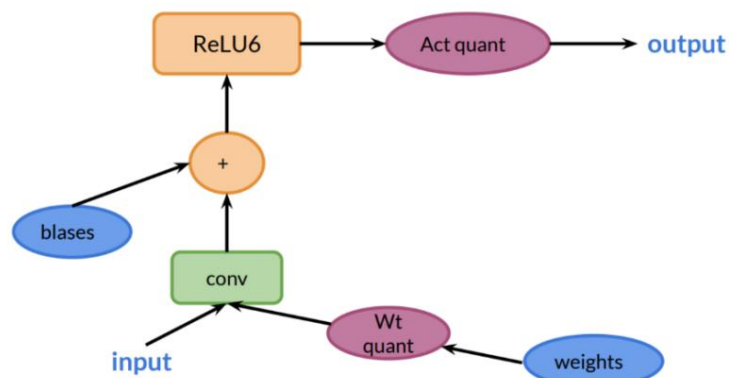
- En la capacitación consciente de la cuantificación, primero crea un modelo como lo haría normalmente y lo hace consciente de la cuantificación utilizando los kits de herramientas de optimización del modelo de TensorFlow, las API. Finalmente, entrena este modelo con las operaciones de emulación de cuantificación para obtener un modelo cuantificado solo de enteros.

### Adding the quantization emulation operations



- Veamos este gráfico simplificado que muestra las operaciones básicas en una red neuronal.

### Adding the quantization emulation operations



- El siguiente paso es agregar operaciones de emulación de cuantificación. Las operaciones de emulación de cuantificación deben colocarse en el gráfico de entrenamiento de manera que sean consistentes con la forma en que se calculará el gráfico cuantificado.
- Las operaciones de cuanto de peso y cuanto de activación introducen pérdidas en el avance del modelo para simular la pérdida de cuantificación real durante la inferencia.
- Tenga en cuenta que no hay operación cuantitativa entre convolución y Relu6. Esto se debe a que Relu6 se fusiona en TensorFlow Lite.

## 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(...)
```

- La API de entrenamiento consciente de la cuantificación facilita el entrenamiento con conocimiento de la cuantificación para un modelo completo o solo partes de ella. -

Luego, expórtela para su implementación con TensorFlow Lite. - Para que

todo el modelo sea consciente de la cuantización, aplicamos `tfmot.quantization.keras.quantize_model` al modelo.

## 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)
```

- La API también es bastante flexible y capaz de manejar casos de uso mucho más complicados. Por ejemplo, le permite controlar la cuantificación con precisión dentro de una capa, crear algoritmos de cuantificación personalizados y manejar cualquier capa personalizada que haya escrito.

- Puede cuantificar selectivamente capas de un modelo para explorar el equilibrio entre precisión, velocidad y modelo Talla.

- Por ejemplo, intente cuantificar las últimas capas en lugar de las primeras capas, y recuerde siempre evitar cuantificar las capas críticas como el mecanismo de atención en las arquitecturas de transformadores, por ejemplo.

## 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()
]))
```

- Si tiene activaciones u otras operaciones que aún no son compatibles con el marco compatible con la cuantificación, puede usar una configuración de cuantificación para resolver esto. - Por ejemplo, en este fragmento de código llama a una configuración personalizada, como `DefaultDenseQuantizeConfig()` para cuantificar una capa personalizada.

## 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)
```

- Finalmente, incluyamos su configuración personalizada en su alcance cuantificado antes de llamar a `quantize_apply`.

## 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

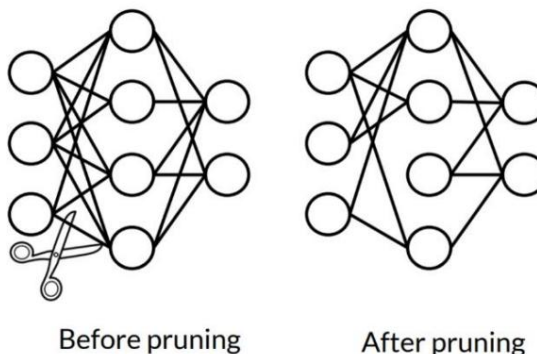
## Model Optimization Results

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

- Estos son algunos resultados que muestran la pérdida de precisión en algunos modelos.
- Esto debería darle una idea de qué esperar en sus propios modelos. Veamos la latencia
- de algunos modelos. Recuerde que los números más bajos son mejores en este caso. Finalmente, veamos el tamaño del
- modelo. Los números más bajos son mejores.

## Poda

### Connection pruning



- Otro método para aumentar la eficiencia de los modelos es eliminar partes del modelo que no contribuyeron sustancialmente para producir resultados precisos. Esto se conoce como poda.
- La optimización de los programas de aprendizaje automático puede tomar varias formas diferentes.

- Afortunadamente, las redes neuronales han demostrado ser resistentes a diversas transformaciones dirigidas a la partitura.

- Cuando consideras redes neuronales más extensas con más capas y nodos, reduciendo su almacenamiento y el costo computacional se vuelve crítico, especialmente para algunas aplicaciones en tiempo real.

- La compresión de modelos se puede utilizar para solucionar este problema. -

A medida que los modelos de aprendizaje automático se incorporaron a dispositivos integrados como teléfonos móviles, la compresión de redes neuronales creció en importancia.

- La poda en el aprendizaje profundo es un concepto de inspiración biológica que discutiremos a continuación.

- La poda tiene como objetivo reducir el número de parámetros y operaciones involucradas en la generación de una predicción mediante eliminación de conexiones de red.

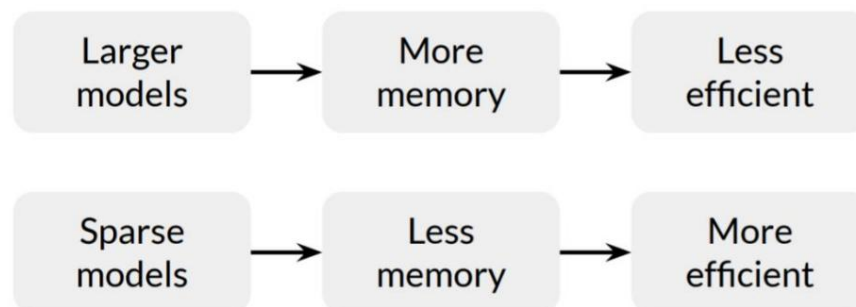
- Con la poda, puede reducir el recuento general de parámetros en la red. - Las redes generalmente

se ven como la de la izquierda. Aquí, cada neurona en una capa tiene una conexión con la capa anterior, pero esto significa que tenemos que multiplicar muchos flotadores juntos. Idealmente, solo conectaríamos cada neurona con algunas otras y ahorraríamos en hacer algunas

- de las multiplicaciones, si podemos encontrar una manera de hacerlo sin demasiada pérdida de precisión. Esa es la motivación detrás de la poda.

-

## Model sparsity



- La escasez de conexiones ha sido durante mucho tiempo un principio fundamental en la investigación en neurociencia, ya que es uno de los hallazgos más críticos sobre el neocórtex.

- Mires donde mires en el cerebro, la actividad de las neuronas siempre es escasa. - Pero las

arquitecturas de redes neuronales comunes tienen muchos parámetros que generalmente no son escasos. - Tomemos por ejemplo

ResNet50. Tiene casi 25 millones de conexiones. Esto significa que durante el entrenamiento necesitamos para ajustar 25 millones de pesos.

- Hacer eso es relativamente costoso, por decir lo menos, por lo que es necesario solucionarlo de alguna manera.

- La historia de la escasez en la red neuronal comienza con la poda, que es una forma de reducir el tamaño de la red neuronal mediante la compresión. - Reducir el número de parámetros tendría varios beneficios. - Una red dispersa no solo es más pequeña, sino que también es más rápida de entrenar y usar. - Donde el hardware es limitado, como en dispositivos integrados como teléfonos inteligentes, la velocidad y el tamaño pueden hacer o

romper un modelo.

- Además, los modelos más complejos son más propensos al sobreajuste.

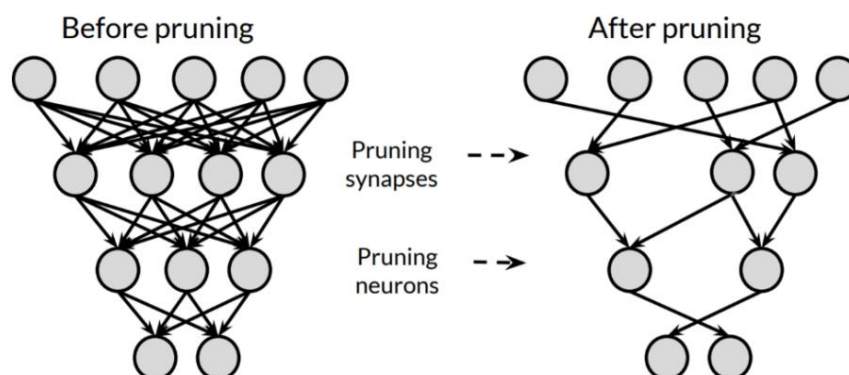
- En cierto sentido, restringir el espacio de búsqueda también puede actuar como regularizador.

- Sin embargo, aun cuando todo eso dicho, no es una tarea sencilla ya que reducir la capacidad del modelo también puede conducir a una pérdida de precisión.

- Como en muchas otras áreas, existe un delicado equilibrio entre complejidad y rendimiento. - Ahora echemos un vistazo más profundo a algunos de los desafíos y posibles soluciones.



## Origins of weight pruning



- Comencemos con un poco de historia. El primer artículo importante que aboga por la escasez y las redes neuronales data de 1990, escrito por Yann LeCun, John S. Denker y Sara A. Solla y recibió el título bastante provocativo de 'Daño cerebral óptimo'. En ese momento, la pospurificación de redes neuronales para comprimir modelos de trenes ya era un enfoque popular. - La poda se realizó principalmente utilizando la magnitud como una aproximación de la prominencia para determinar las menos útiles.

conexiones

- La intuición es que los pesos de menor magnitud tienen un efecto menor en la salida y, por lo tanto, son menos probable que tenga un impacto en el resultado del modelo si se prueba.
- Era un método de poda iterativo. El primer paso fue entrenar un modelo, luego se estimó la prominencia de cada peso, que se definió por el cambio en la función de pérdida al aplicar una perturbación a los nodos de la red.
- Cuanto menor sea el cambio, menor será el efecto del peso en el entrenamiento.
- Finalmente, eliminan los pesos con menor prominencia. Esto es equivalente a ponerlos a cero.
- Finalmente, este modelo podado fue reentrenado.
- Surge un desafío particular con este método cuando se vuelve a entrenar la red podada. Resultó que debido a su capacidad reducida, el reciclaje fue mucho más difícil.
- La solución a este problema llegó más tarde, junto con una idea llamada la hipótesis del billete de lotería.

## The Lottery Ticket Hypothesis

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

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

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

- La probabilidad de ganar el premio mayor de una lotería es muy baja. - Por ejemplo, si está jugando Powerball, tiene probabilidades de ganar exactamente uno en aproximadamente 3 millones. boleto. ¿Cuáles son sus posibilidades si compra n boletos? Si la probabilidad de ganar es de 1 sobre 3 millones, ¿qué pasa con las posibilidades de no ganar? Es el complemento de 1 menos p. Extiende esto al comprar n boletos y tenemos la probabilidad de 1 menos p elevado a n.
- De esto se deduce que la probabilidad de que al menos uno de ellos gane es simplemente el complemento nuevamente.

- ¿Qué tiene que ver esto con las redes neuronales?
- Antes del entrenamiento, los pesos de un modelo se inicializan aleatoriamente. ¿Puede suceder que haya una subred? de una red inicializada aleatoriamente que ganó la lotería de inicialización?

## 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

- Algunos investigadores se propusieron investigar el problema y responder la pregunta.
- En particular, Frankle y Carbin 2019 descubrieron que no era necesario ajustar los pesos después del entrenamiento para estas nuevas redes podadas. De hecho, demostraron que el mejor enfoque era restablecer los pesos a su valor original y luego volver a entrenarlos .
- Esto daría lugar a modelos con una precisión aún mayor en comparación con el modelo denso original y el enfoque de pospoda más ajuste fino propuesto por Han y sus colegas. Este descubrimiento los llevó a proponer una idea que al principio se consideró descabellada, pero ahora comúnmente aceptada, es decir, sobre redes densas parametrizadas que contienen varias subredes dispersas con diferentes rendimientos, y una de estas subredes es el boleto ganador, que supera a todos los demás.

## 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

- Sin embargo, hubo limitaciones significativas para este método. - Por un lado, no funciona bien para problemas y arquitecturas de mayor escala.
- En el artículo original, los autores afirmaron que para conjuntos de datos más complejos como ImageNet y arquitecturas más profundas como ResNet, el método no logra identificar a los ganadores de la lotería de inicialización. En general, lograr un buen equilibrio entre precisión y escasez es un problema difícil. Es un campo de investigación muy activo y el estado del arte sigue mejorando.

## Eliminate connections based on their magnitude

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

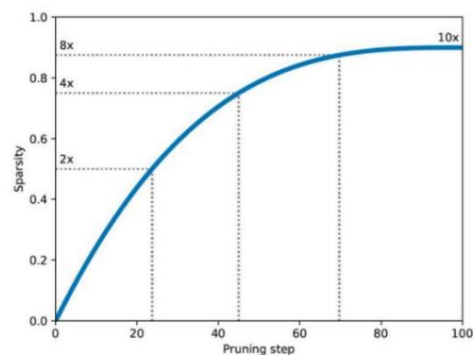
0	2	0	4
0	6	3	0
4	0	0	3
0	3	0	5

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)

- TensorFlow incluye una API de reducción de peso basada en Keras, que utiliza un sencillo pero ampliamente aplicable algoritmo diseñado para eliminar iterativamente las conexiones en función de su magnitud durante el entrenamiento.
- Fundamentalmente se especifica un objetivo final de escasez junto con un cronograma para realizar la poda.

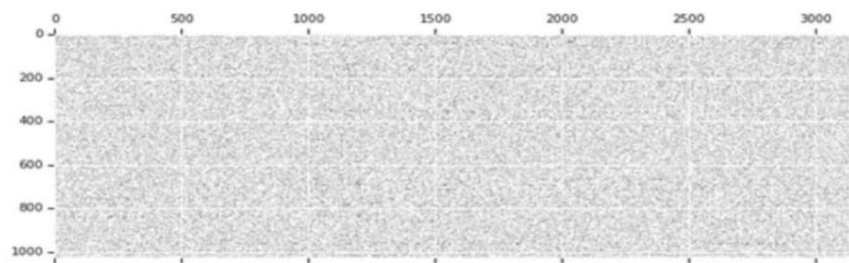
## Apply sparsity with a pruning routine



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%.

- En esta figura, puede ver que durante el entrenamiento, se programará la ejecución de una rutina de poda, eliminando los pesos con los valores de magnitud más bajos que estén más cerca de cero hasta que se alcance el objetivo de dispersión actual.
- Cada vez que se programa la ejecución de la rutina de poda, el objetivo de escasez actual se vuelve a calcular a partir del cero por ciento hasta que alcanza la escasez objetivo final al final del programa de poda aumentándolo gradualmente de acuerdo con una función de aumento suave. Al igual que un programa, la función de rampa se puede ajustar según sea necesario.
- Por ejemplo, en ciertos casos, puede ser conveniente programar el procedimiento de entrenamiento para que comience después de un cierto paso cuando se ha alcanzado algún nivel de convergencia.
- O finalice la poda antes del número total de pasos de entrenamiento en su programa de entrenamiento para ajustar aún más el sistema en el nivel de dispersión objetivo final.

## Sparsity increases with training



Black cells indicate where the non-zero weights exist

Animation of pruning applied to a tensor

- La dispersión aumenta a medida que avanza el entrenamiento. Necesitas saber cuándo parar.
- Eso significa que al final del procedimiento de formación, los tensores correspondientes a las capas de Keras podadas se contienen ceros donde los pesos se han reducido de acuerdo con el objetivo de escasez final para la capa.

## 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

- Un beneficio inmediato que puede obtener de la poda es la compresión del disco.
- Eso es porque los tensores dispersos son comprimibles. Por lo tanto, al aplicar una compresión de archivos simple al punto de control de TensorFlow podado o al modelo de TensorFlow Lite convertido, podemos reducir el tamaño del modelo para el almacenamiento o la transmisión.
- En algunos casos, incluso puede obtener mejoras de velocidad en la CPU y los aceleradores de aprendizaje automático que aprovechan las eficiencias de precisión de enteros.
- Además, a través de varios experimentos, encontramos que la poda de peso es compatible con la cuantificación, resultando en beneficios compuestos. En el próximo ejercicio, mostramos que podemos comprimir aún más el modelo recortado de dos megabytes a casi la mitad de un megabyte aplicando la cuantificación posterior al entrenamiento. En un futuro relativamente cercano, TensorFlow Lite agregará soporte de primera clase para representación escasa en computación, expandiendo así el beneficio de compresión a la memoria en tiempo de ejecución y desbloqueando mejoras de rendimiento.
- Los tensores dispersos le permiten omitir cálculos innecesarios que involucran los valores puestos a cero. - O dependiendo de cuándo estés viendo esto, es posible que ya esté incluido.

## Pruning with TF Model Optimization Toolkit



- Para usar la API de poda, primero crea un modelo de TensorFlow Keras. - Luego agregamos escasez a algunas de las capas en el modelo y lo volvemos a entrenar o lo entrenamos.
- Finalmente, también puede leer los beneficios de la cuantificación al convertir el modelo recortado a TFLite.

## 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(...)
  
```

- Apliquemos la poda a todo el modelo. En este ejemplo, comienza con una dispersión del 50 por ciento. 50 por ciento de ceros y pesos y terminan con un 80 por ciento de escasez.
- También puede podar solo una parte del modelo o capas específicas para mejorar la precisión del modelo.
- Más adelante, verá cómo crear modelos dispersos con la API del kit de herramientas de optimización de modelos TensorFlow para TensorFlow y TFLite.
- Luego combina la poda con la cuantificación posterior al entrenamiento para obtener beneficios adicionales.

## Results across different models & tasks

Model	Non-sparse Top-1 acc.	Sparse acc.	Sparsity	Model	Non-sparse BLEU	Sparse BLEU	Sparsity
Inception V3	78.1%	78.0%	50%	GNMT EN-DE	26.77	26.86	80%
		76.1%	75%			26.52	85%
		74.6%	87.5%			26.19	90%
Mobilenet V1 224	71.04%	70.84%	50%	GNMT DE-EN	29.47	29.50	80%
						29.24	85%
						28.81	90%

- Además, esta técnica de poda se puede aplicar con éxito a diferentes tipos de modelos en distintas tareas. - Desde redes neuronales basadas en convoluciones de procesamiento de imágenes hasta procesamiento de voz utilizando redes neuronales recurrentes. Esta tabla muestra algunos de estos resultados experimentales.

## Referencias

- [Análisis de componentes principales \(PCA\)](#) •
- [Análisis de componentes independientes \(ICA\)](#) •
- [Extensiones de PCA](#)
- [Cuantización](#) •
- [Cuantización posterior al entrenamiento](#)
- [Entrenamiento consciente de la](#)
- [cuantificación](#) • [Poda](#) • [La hipótesis del](#)
- [billete de lotería](#)