Pipelines de modelado de aprendizaje automático en producción

En el tercer curso de Ingeniería de Aprendizaje Automático para la Especialización en Producción, construirá modelos para diferentes entornos de servicio; implementará herramientas y técnicas para gestionar eficazmente sus recursos de modelado y atender mejor las solicitudes de inferencia offline y online; y utilizará herramientas de análisis y métricas de rendimiento para abordar la equidad del modelo, los problemas de explicabilidad y mitigar los cuellos de botella.

Comprender los conceptos de aprendizaje automático y aprendizaje profundo es esencial, pero si quieres desarrollar una carrera profesional eficaz en IA, también necesitas capacidades de ingeniería de producción. La ingeniería de aprendizaje automático para producción combina los conceptos básicos del aprendizaje automático con los conocimientos funcionales del desarrollo de software moderno y las funciones de ingeniería para ayudarle a desarrollar habilidades listas para la producción.

Semana 1: Búsqueda de arquitectura neuronal

Contenido

Semana 1: Búsqueda de arquitectura neuronal 1

Búsqueda de arquitectura neuronal 2

Demostración de Keras Autotuner 4

Introducción a AutoML 9

Comprender los espacios de búsqueda 12

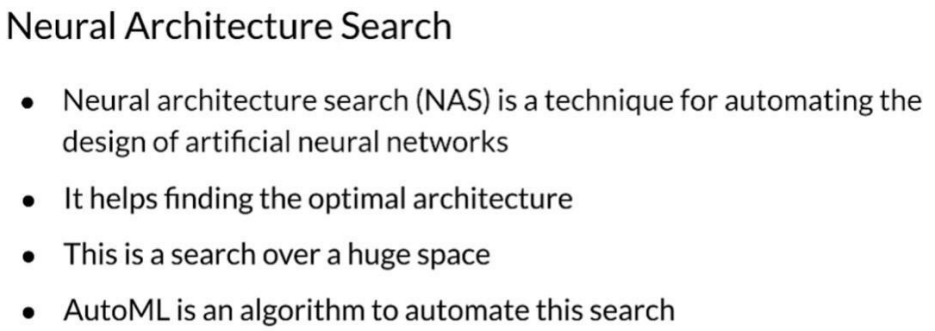
Estrategias de búsqueda 13

Medición de la eficacia del AutoML 16

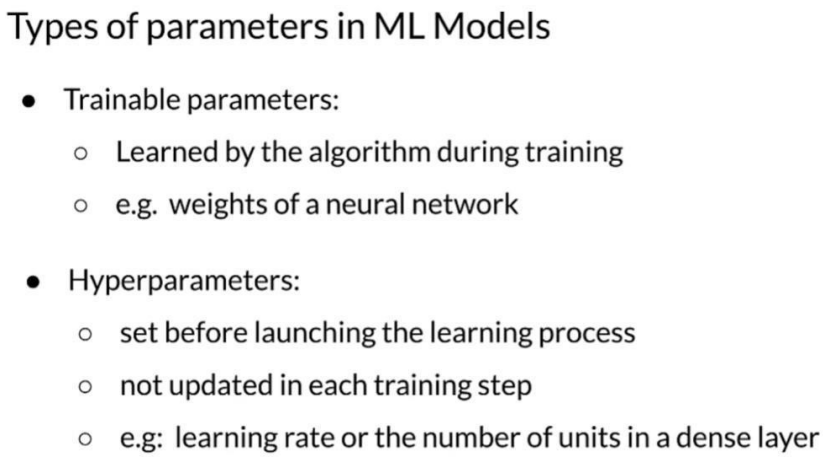
AutoML en la nube 19

Referencias 25

Búsqueda de arquitectura neuronal



- Empezaremos con la búsqueda de arquitectura neuronal.
- Empezaremos con una discusión de algo con lo que puede que estés más familiarizado, el ajuste de hiperparámetros. Como verán, hay similitudes entre el ajuste de hiperparámetros y la búsqueda de arquitectura neuronal.
- La búsqueda de arquitecturas neuronales, o NAS, ocupa un lugar destacado en los recientes desarrollos de ML.
- En esencia, es una técnica para **automatizar el diseño de redes neuronales**.
- Los modelos hallados por NAS suelen igualar o superar a las arquitecturas diseñadas a mano para muchos tipos de problemas. El objetivo de NAS es encontrar la arquitectura óptima.
- Tenga en cuenta que las redes neuronales modernas abarcan un espacio de parámetros enorme.
- Automatizar la búsqueda con herramientas como AutoML tiene mucho sentido.



- Antes de sumergirnos a fondo en AutoML, vamos a entender el problema que resuelve analizando uno de los procesos más tediosos del modelado ML que se ha hecho ingenuamente, que es el ajuste de hiperparámetros.
- En los modelos de aprendizaje automático, hay dos tipos de parámetros. Están los parámetros del modelo. Son parámetros que el modelo debe aprender utilizando el conjunto de entrenamiento.
- Son parámetros ajustados o entrenados de nuestros modelos. Por lo general, eso significa pesos y sesgos.
- Luego tenemos los hiperparámetros. Se trata de parámetros ajustables que deben sintonizarse para crear un modelo con un rendimiento óptimo, pero a diferencia de los parámetros del modelo, los hiperparámetros no se optimizan automáticamente durante el proceso de entrenamiento.
- Deben establecerse antes de que comience el entrenamiento del modelo y afectan al modo en que éste se entrena.

Manual hyperparameter tuning is not scalable

- Hyperparameters can be numerous even for small models
- e.g shallow DNN:
  - Architecture choices
  - activation functions
  - Weight initialization strategy
  - Optimization hyperparameters such as learning rate, stop condition
- Tuning them manually can be a real brain teaser
- Tuning helps with model performance

- El ajuste de los hiperparámetros puede tener un gran impacto en el rendimiento del modelo y, por desgracia, el número de hiperparámetros puede ser considerable incluso para modelos pequeños.
- Por ejemplo, en una DNN superficial, se pueden elegir hiperparámetros para las opciones de arquitectura, para las funciones de activación, para la estrategia de inicialización de pesos y los hiperparámetros de optimización, entre otros.
- Realizar el ajuste manual de los hiperparámetros puede ser un verdadero rompecabezas, lo cual es una buena forma de decirlo, ya que vas a necesitar hacer un seguimiento de lo que has probado y lanzar diferentes experimentos, por lo que un proceso manual como ese es tedioso.
- No obstante, cuando se hace correctamente, el ajuste de los hiperparámetros ayuda a mejorar el rendimiento del modelo de forma significativa.

Automating hyperparameter tuning with Keras Tuner

- Automation is key: open source resources to the rescue
- Keras Tuner:
  - Hyperparameter tuning with Tensorflow 2.0.
  - Many methods available

- Se han creado varias bibliotecas de código abierto que utilizan diversos enfoques para el ajuste de hiperparámetros. El equipo de Keras ha publicado una de las mejores, Keras Tuner, que es una biblioteca para realizar fácilmente el ajuste de hiperparámetros con TensorFlow 2.0.
- Ofrece diversos métodos de ajuste, como la búsqueda aleatoria, la hiperbanda y la optimización bayesiana. Veamos un ejemplo concreto en el siguiente vídeo.

Demostración de Keras Autotuner

Setting up libraries and dataset

```
import tensorflow as tf
from tensorflow import keras
mnist = tf.keras.datasets.mnist


(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Deep learning “Hello world!”

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

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

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Model performance

```
Epoch 1/5
1875/1875 - 10s 5ms/step - loss: 0.3603 - accuracy: 0.8939
Epoch 2/5
1875/1875 - 10s 5ms/step - loss: 0.1001 - accuracy: 0.9695
Epoch 3/5
1875/1875 - 10s 5ms/step - loss: 0.0717 - accuracy: 0.9781
Epoch 4/5
1875/1875 - 10s 5ms/step - loss: 0.0515 - accuracy: 0.9841
Epoch 5/5
1875/1875 - 10s 5ms/step - loss: 0.0432 - accuracy: 0.9866
```

Parameters rational: if any

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

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

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Is this architecture optimal?

- Do the model need more or less hidden units to perform well?
- How does model size affect the convergence speed?
- Is there any trade off between convergence speed, model size and accuracy?
- Search automation is the natural path to take
- Keras tuner built in search functionality.

- Puede entrenar un modelo en sólo unos segundos, con una precisión del 97% en este conjunto de datos.
- Como puedes ver, cuando ejecutamos esto, acabamos con una precisión de 0,9866 y cada epoch tarda poco más de 10 segundos. Pero, ¿podemos hacerlo mejor?
- La primera pregunta que me suelen hacer después de mostrar esto es: ¿de dónde salen estos números? ¿Por qué esta arquitectura tiene 512 neuronas? ¿Por qué no otro número? ¿Y por qué el dropout utiliza 0,2, por qué no 0,5 o 0,1?
- A menudo, la respuesta es que utilicé un montón de ensayo y error para llegar a estos números. O, como suele ocurrir, me limité a copiar el código de un ejemplo que funcionaba y no pensé realmente en ello.
- Pensemos si se trata de una elección de atributos o no. Las preguntas naturales son: ¿el modelo funcionará mejor con más unidades ocultas o con menos? ¿Aprenderá más rápido con menos unidades? ¿Puede una arquitectura más pequeña hacerlo sin perder precisión?
- Puedes experimentar cambiándolo, volviéndolo a ejecutar, cambiándolo de nuevo y volviéndolo a ejecutar, etcétera. Pero eso es mucho trabajo. ¿Y si pudieras automatizar esto? Entonces Keras tuner al rescate.

Automated search with Keras tuner

```
# First, install Keras Tuner
!pip install -q -U keras-tuner

# Import Keras Tuner after it has been installed
import kerastuner as kt
```

Building model with iterative search

```
def model_builder(hp):
  model = keras.Sequential()
  model.add(keras.layers.Flatten(input_shape=(28, 28)))

  hp_units = hp.Int('units', min_value=16, max_value=512, step=16)
  model.add(keras.layers.Dense(units=hp_units, activation='relu'))
  model.add(tf.keras.layers.Dropout(0.2))
  model.add(keras.layers.Dense(10))

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

- En lugar de codificar 512 neuronas en la capa oculta, utilizaremos este código. Observe que el número de unidades en la primera capa densa se establece en **hp.Int**.
- Esto será sintonizado por el sintonizador Keras. Así que esto establece un valor entero que comienza en 16 y va hasta 512 en pasos de 16. Más adelante verás cómo el sintonizador Keras ejecutará el modelo varias veces, recopilando métricas cada vez y optimizando la mejor.
- En este caso, el valor es lo que impulsa el número de veces que el sintonizador Keras hace lo suyo.

Search strategy

```
tuner = kt.Hyperband(model_builder,
                     objective='val_accuracy',
                     max_epochs=10,
                     factor=3,
                     directory='my_dir',
                     project_name='intro_to_kt')
```

- A continuación, definirás tu estrategia de búsqueda. Keras tuner soporta múltiples estrategias y tú defines cuál quieres usar así.
- En este caso, estoy utilizando la estrategia **Hyperband**.
- También admite la búsqueda aleatoria y las estrategias de optimización bayesiana y Sklearn.
- Los parámetros que elija variarán en función de su estrategia. Pero lo importante a tener en cuenta es el objetivo. En este caso, nuestro objetivo es val_accuracy, por lo que queremos maximizar la precisión de la validación.

Callback configuration

```
stop_early =
    tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                     patience=5)

tuner.search(x_train,
             y_train,
             epochs=50,
             validation_split=0.2,
             callbacks=[stop_early])
```

- La búsqueda puede tardar un poco en completarse y consumir muchos recursos informáticos. Pero puedes configurar una llamada de retorno que detenga la búsqueda cuando se cumplan las condiciones.
- Así, por ejemplo, aquí estoy monitoreando la pérdida de validación y la **paciencia** se establece en cinco, lo que significa que no cambia significativamente, o si no cambia significativamente en cinco épocas, a continuación, dejar de buscar en esta iteración.
- Se establece la devolución de llamada como parámetro de búsqueda. El resto del parámetro especifica cómo buscar, como los datos y la etiqueta, el número de épocas para entrenar y la división de validación.

Search output

```
Trial 24 Complete [00h 00m 22s]
val_accuracy: 0.3265833258628845
Best val_accuracy So Far: 0.5167499780654907
Total elapsed time: 00h 05m 05s
Search: Running Trial #25
Hyperparameter    |Value           |Best Value So Far
units             |192             |48
tuner/epochs      |10              |2
tuner/initial_e...|4               |0
tuner/bracket     |1               |2
tuner/round       |1               |0
tuner/trial_id    |a2edc917bda476c...|None
```

- Mientras busca, verás los resultados de cada prueba. Sólo estamos buscando en un parámetro, las unidades en la capa oculta densa.
- Como se puede ver a medida que se actualiza, puede realizar un seguimiento del mejor valor hasta el momento. Que en este momento tenemos en la diapositiva es 48, y en este caso en realidad terminó de esa manera también cuando se completó.

Back to your model

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(48, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

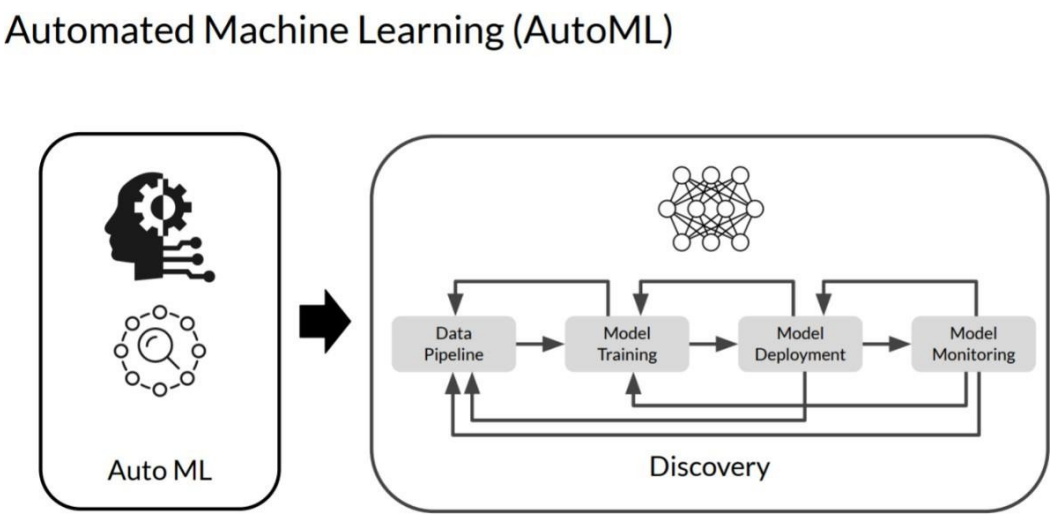
- Así que puedo volver y probar mi arquitectura de nuevo y utilizar los resultados del sintonizador Keras para establecer el número de unidades manualmente, esta vez con 48 neuronas en la capa, que es el número que obtuvimos del sintonizador Keras.

Training output

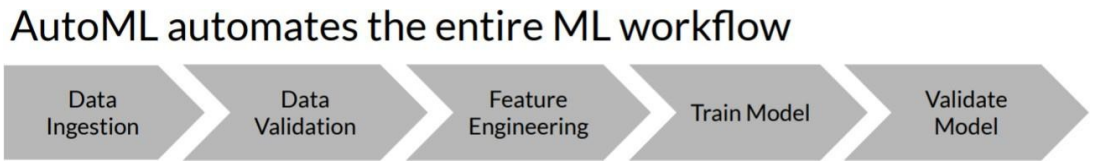
```
Epoch 1/5
1875/1875 - 3s 1ms/step - loss: 0.6427 - accuracy: 0.8090
Epoch 2/5
1875/1875 - 3s 1ms/step - loss: 0.2330 - accuracy: 0.9324
Epoch 3/5
1875/1875 - 3s 1ms/step - loss: 0.1835 - accuracy: 0.9448
Epoch 4/5
1875/1875 - 3s 1ms/step - loss: 0.1565 - accuracy: 0.9515
Epoch 5/5
1875/1875 - 3s 1ms/step - loss: 0.1393 - accuracy: 0.9564
```

- Y cuando esté reentrenado, verás los resultados.
- Sólo han pasado cinco épocas y el valor no es tan bueno como antes, pero nuestras épocas son **tres veces más rápidas** que antes (3 segundos frente a los 10 segundos por época anteriores).
- Así que tal vez pueda entrenar durante más tiempo para obtener mejores resultados sabiendo que, como mínimo, he optimizado parte de mi arquitectura.

Introducción a AutoML

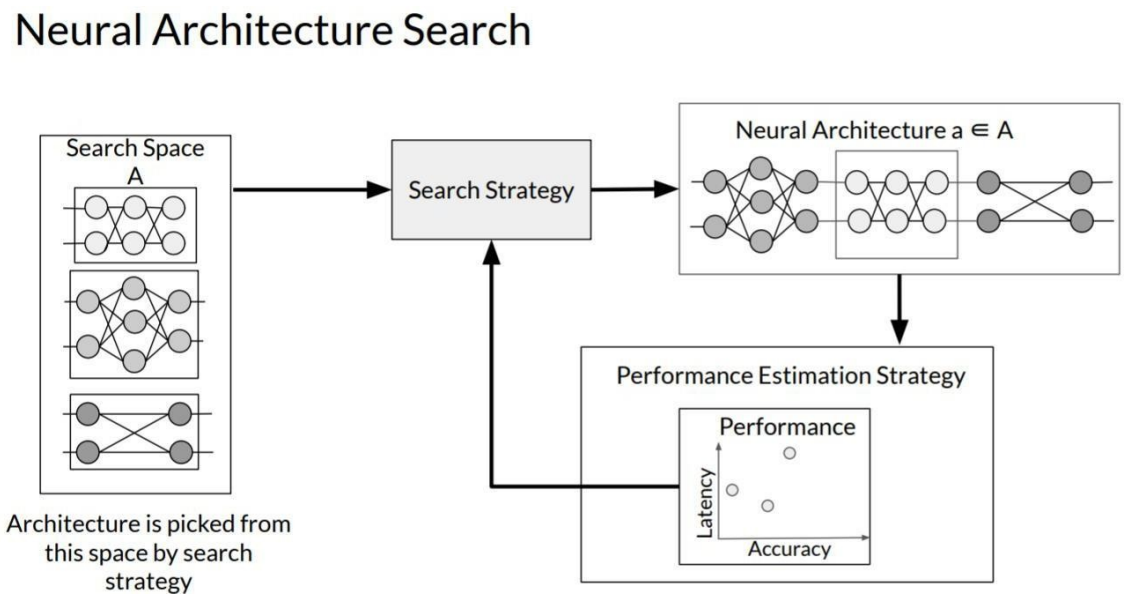


- Ahora que ya hemos visto el ajuste de hiperparámetros, echemos un vistazo al AutoML real.
- En esta lección, hablaremos de AutoML, un conjunto de herramientas muy versátiles para automatizar el proceso de aprendizaje automático de principio a fin.
- Encontrar el modelo adecuado es una pieza importante del rompecabezas.
- La búsqueda de arquitecturas neuronales es un proceso que ayuda a encontrar modelos relevantes. Aprenderá sobre los espacios de búsqueda y las estrategias que son clave tanto para el ajuste de hiperparámetros como para AutoML.
- También hablaremos de herramientas para cuantificar la estimación del rendimiento de los modelos explorados en su búsqueda.
- Por último, conocerá las diferentes ofertas de AutoML disponibles en la nube por parte de los principales proveedores de servicios.
- Automated Machine Learning o AutoML tiene como objetivo permitir a los desarrolladores con muy poca experiencia en aprendizaje automático, hacer uso de modelos y técnicas de aprendizaje automático.
- Trata de automatizar el proceso de aprendizaje automático de principio a fin, con el fin de producir soluciones sencillas, una creación más rápida de esas soluciones y modelos que a veces superan incluso a los modelos ajustados a mano.
- AutoML aplica técnicas de aprendizaje automático y búsqueda al proceso de creación de modelos y cadenas de aprendizaje automático.
- Abarca todo el proceso, desde el conjunto de datos brutos hasta el modelo de aprendizaje automático desplegable.



- En el aprendizaje automático tradicional, escribimos código para todas las fases del proceso.
- Empezamos ingiriendo y depurando los datos brutos y, a continuación, realizamos la selección y la ingeniería de características.
- Seleccionamos una arquitectura de modelo para una tarea, entrenamos o modelamos y realizamos el ajuste de hiperparámetros, quizá manualmente o utilizando un sintonizador como Keras tuner y, a continuación, validamos el rendimiento de nuestros modelos.
- El ML requiere mucha programación manual y un conjunto de conocimientos muy especializados.

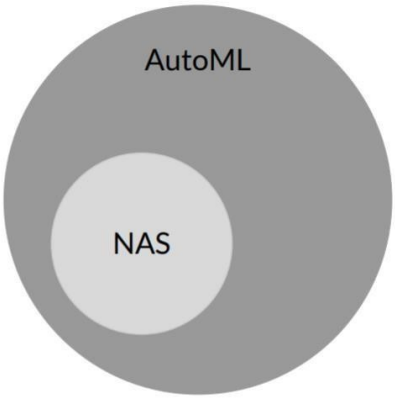
- Auto ML pretende automatizar todo el flujo de trabajo de ML. Si podemos proporcionar al sistema AutoML datos sin procesar y nuestros requisitos de validación del modelo, éste pasa por todas las fases del flujo de trabajo de ML y realiza el proceso iterativo de desarrollo de ML de forma sistemática hasta que el modelo está entrenado.



- La búsqueda de arquitectura neuronal o NAS es el núcleo de AutoML.
- **La búsqueda de arquitecturas neuronales** consta de **tres partes principales**: un espacio de búsqueda, una estrategia de búsqueda y u n a estrategia de estimación del rendimiento.
- El espacio de búsqueda define la gama de arquitecturas que pueden representarse.
- Para reducir el tamaño del problema de búsqueda, tenemos que limitar el espacio de búsqueda a las arquitecturas que mejor se adaptan al problema que intentamos modelar. Esto ayuda a reducir el espacio de búsqueda, pero también significa que se introducirá un **sesgo humano**, que podría impedir que la Búsqueda de Arquitectura Neuronal encuentre bloques arquitectónicos que vayan más allá del conocimiento humano actual.
- La estrategia de búsqueda define cómo exploramos el espacio de búsqueda. Queremos explorar el espacio de búsqueda con rapidez, pero esto puede conducir a una convergencia prematura en una región subóptima del espacio de búsqueda.
- El objetivo de la búsqueda de arquitectura neuronal es encontrar una arquitectura que funcione bien con nuestros datos.
- La estrategia de estimación del rendimiento ayuda a medir y comparar el rendimiento de varias arquitecturas.
- Una estrategia de búsqueda selecciona una arquitectura de un espacio de búsqueda predefinido de arquitecturas. La arquitectura seleccionada se pasa a una estrategia de estimación del rendimiento, que devuelve su rendimiento estimado a la estrategia de búsqueda.
- El espacio de búsqueda, la estrategia de búsqueda y la estrategia de estimación del rendimiento son los componentes clave de la búsqueda de arquitecturas neuronales y trataremos cada uno de ellos por separado.

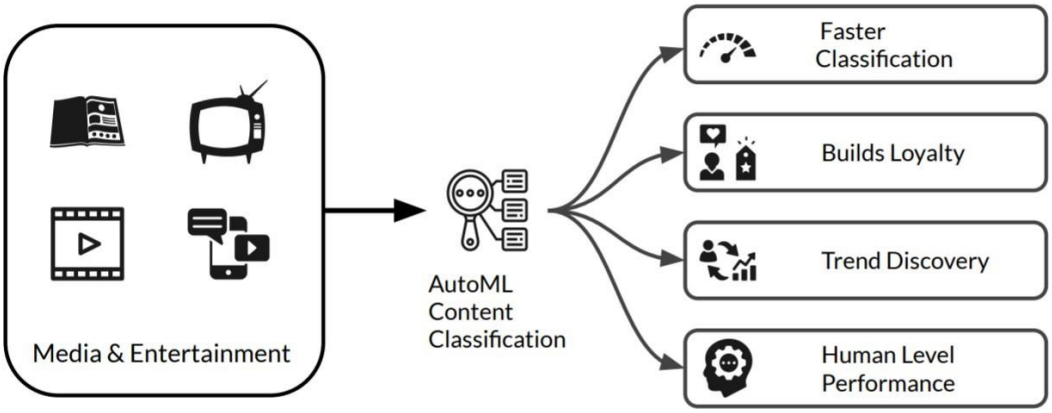
Neural Architecture Search

- **AutoML** automates the development of ML models
- **AutoML** is not specific to a particular type of model.
- Neural Architecture Search (**NAS**) is a subfield of AutoML
- NAS is a technique for automating the design of artificial neural networks (ANN).



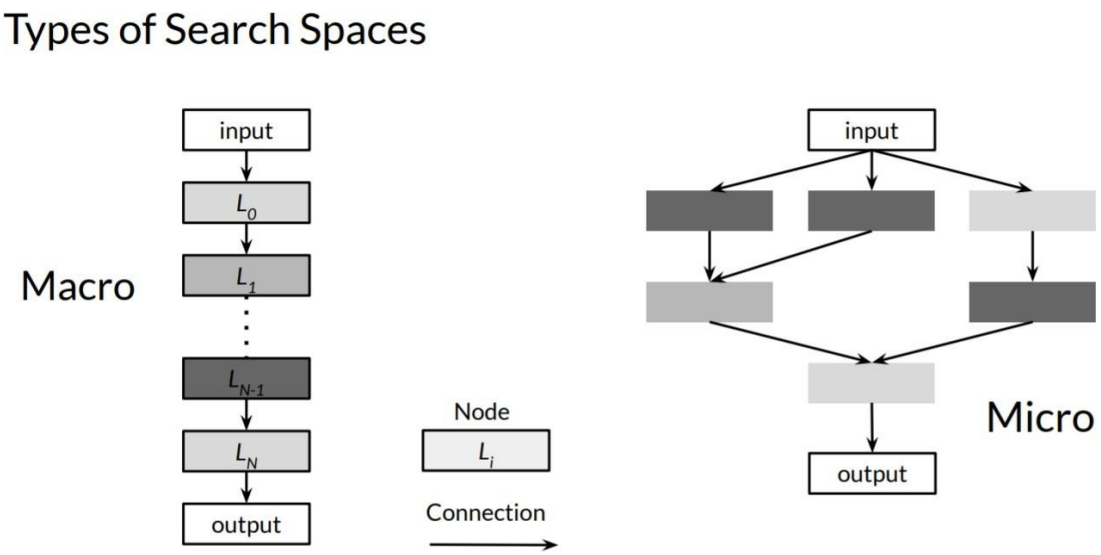
- AutoML facilita la construcción de modelos de forma automatizada. Además, AutoML no se limita a una familia o subconjunto específico de modelos.
- **La búsqueda de arquitectura neuronal es un subcampo de AutoML. Se** centra específicamente en la parte de selección de modelos del flujo de trabajo de AutoML y en el diseño de redes neuronales. Se ha utilizado para diseñar arquitecturas que igualan o superan a los modelos diseñados a mano.
- **Es importante señalar que NAS NO equivale a AutoML.** NAS es el proceso de automatización de la ingeniería de arquitectura

Real-World example: Meredith Digital

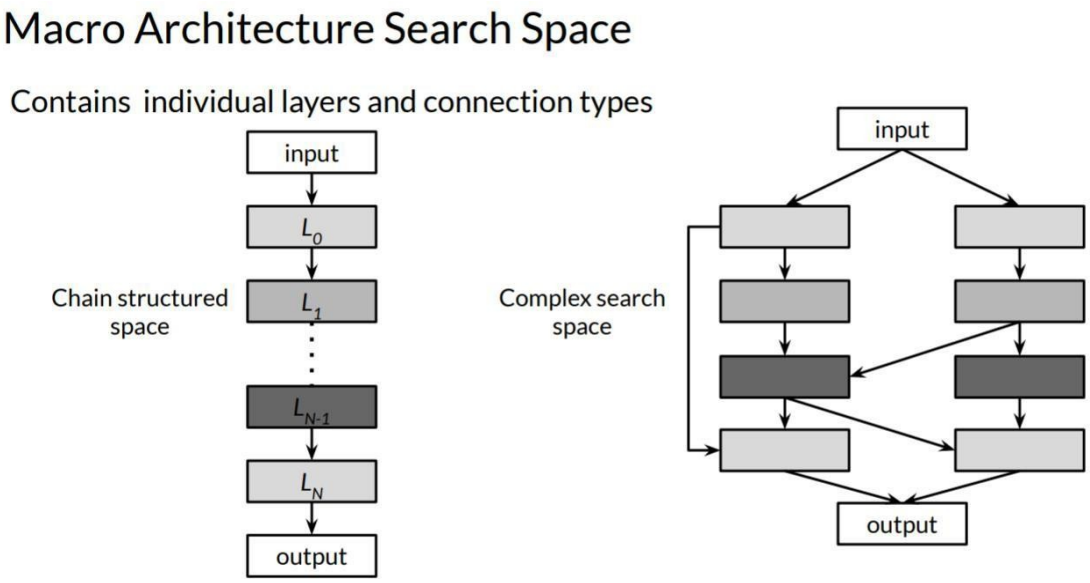


- Veamos un uso real de AutoML. Meredith Digital es una empresa editorial especializada en múltiples formatos de medios de comunicación y entretenimiento. Meredith Digital utiliza AutoML para entrenar modelos, en su mayoría **basados en lenguaje natural para automatizar la clasificación de contenidos.**
- AutoML acelera el proceso de clasificación reduciendo el proceso de meses a sólo unos días.
- También ayuda proporcionando recomendaciones perspicaces y prácticas para ayudar a fidelizar a los clientes.
- Ayuda a identificar las nuevas tendencias de los usuarios y los intereses de los clientes, a fin de adaptar los contenidos para servir mejor a los clientes.
- Para comprobar su eficacia, realizaron una prueba en la que compararon AutoML con sus modelos generados manualmente y los resultados fueron bastante sorprendentes.
- Las **herramientas de lenguaje natural AutoML de Google Cloud** proporcionaron una clasificación de contenidos comparable al rendimiento humano.

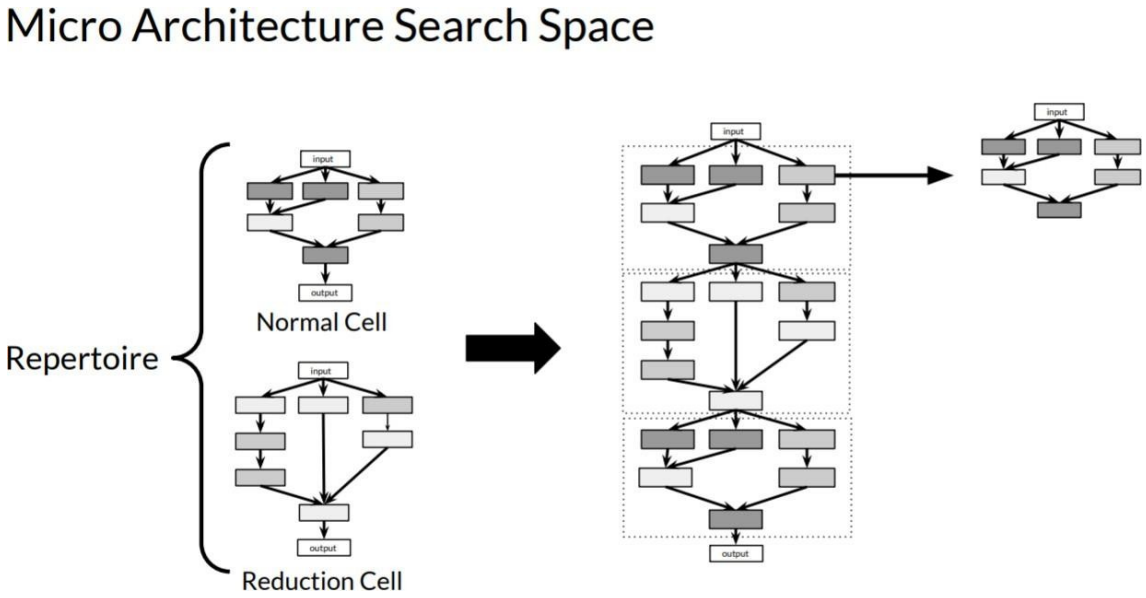
Comprender los espacios de búsqueda



- Hay dos tipos principales de espacios de búsqueda, macro y micro. Y en realidad sus nombres están un poco al revés, pero así es como se llaman.
- En primer lugar, definamos qué entendemos por nodo.
- **Un nodo es una capa de una red neuronal, como una capa de convolución o de agrupación.**
- En esta ilustración, cada color representa un tipo de capa diferente. Una flecha de la capa LI a LJ indica que la capa LJ recibe la salida de LI como entrada.

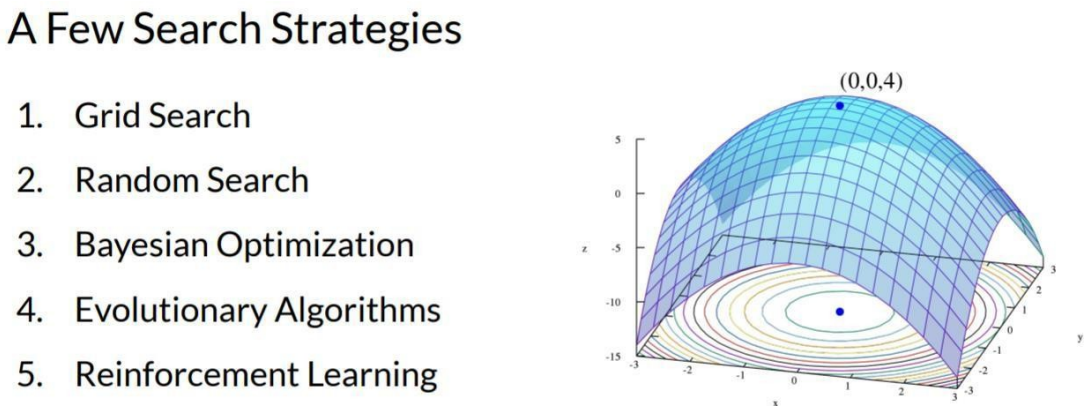


- Un macroespacio de búsqueda contiene las capas individuales y los tipos de conexión de una red neuronal.
- La búsqueda de arquitecturas neuronales busca dentro de ese espacio el mejor modelo, construyendo el modelo capa a capa. Como se muestra aquí, una red puede construirse de forma muy sencilla **apilando** capas individuales, lo que se conoce como un espacio estructurado en cadena, o con múltiples ramas y conexiones de salto en un espacio complejo.



- En cambio, en un microespacio de búsqueda, la búsqueda de arquitectura neuronal construye una red neuronal a partir de células en las que
  cada célula es una red más pequeña.
- Aquí hay dos tipos diferentes de células, una célula normal en la parte superior y una célula de reducción en la parte inferior. Las células se apilan para producir la red final.
- **Este enfoque ha demostrado tener importantes ventajas de rendimiento en comparación con un enfoque macro**.
- Aquí se muestra una arquitectura construida apilando las celdas secuencialmente. Obsérvese que las celdas también pueden combinarse de forma más compleja, como en los espacios multirama, simplemente sustituyendo capas por celdas.

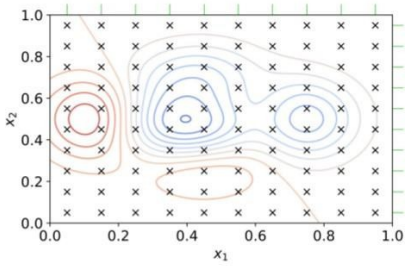
Estrategias de búsqueda



- ¿Cómo decide la búsqueda de arquitectura neuronal qué opciones del espacio de búsqueda probar a continuación? Necesita una **estrategia de búsqueda**
- La búsqueda de arquitecturas neuronales busca en la base de búsqueda la arquitectura que produce el mejor rendimiento. Para esa búsqueda se pueden utilizar distintos enfoques.
- Entre ellos figuran la búsqueda en cuadrícula, la búsqueda aleatoria, la optimización bayesiana, los algoritmos evolutivos y el aprendizaje por refuerzo.

Grid Search and Random Search

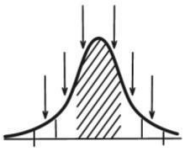
- Grid Search
  - Exhaustive search approach on fixed grid values
- Random Search
- Both suited for smaller search spaces.
- Both quickly fail with growing size of search space.

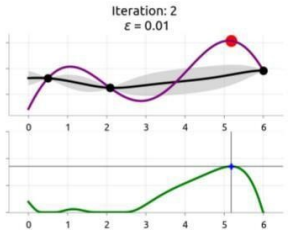


- En la **búsqueda cuadriculada** se busca todo, lo que significa que se cubren todas las opciones que tenemos en la base de búsqueda. En la búsqueda **aleatoria**, seleccionas la siguiente opción al azar dentro del espacio de búsqueda.
- **Ambos funcionan razonablemente bien en bases de búsqueda pequeñas, pero también fallan con bastante rapidez** cuando el espacio de búsqueda crece más allá de un cierto tamaño, lo cual es demasiado habitual.

Bayesian Optimization

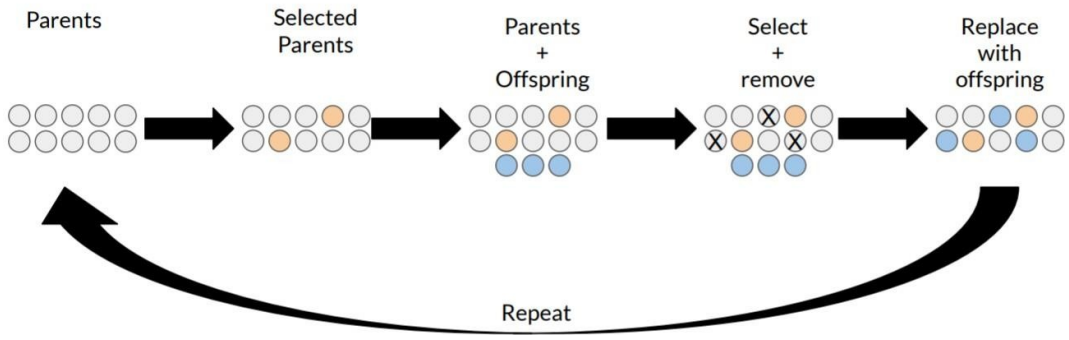
- Assumes that a *specific probability distribution*, is underlying the performance.
- Tested architectures constrain the probability distribution and guide the selection of the next option.
- In this way, promising architectures can be stochastically determined and tested.





- La optimización bayesiana es un poco más sofisticada. Asume que una distribución de probabilidad específica, que suele ser una **distribución gaussiana**, subyace al rendimiento de las arquitecturas de los modelos.
- Así que se utilizan observaciones de arquitecturas probadas para restringir la distribución de probabilidades y guiar la selección de la siguiente opción.
- Esto nos permite **construir una arquitectura estocástica** basada en los resultados de las pruebas y la distribución restringida.

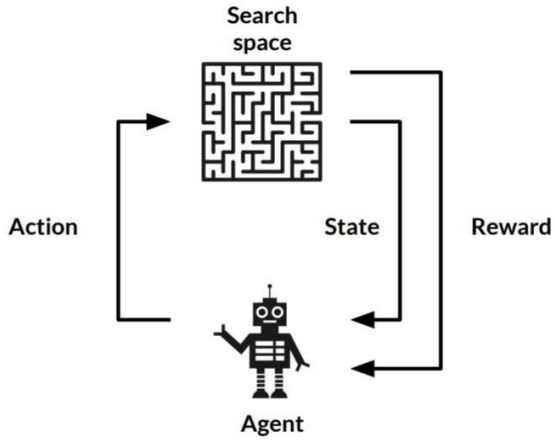
Evolutionary Methods



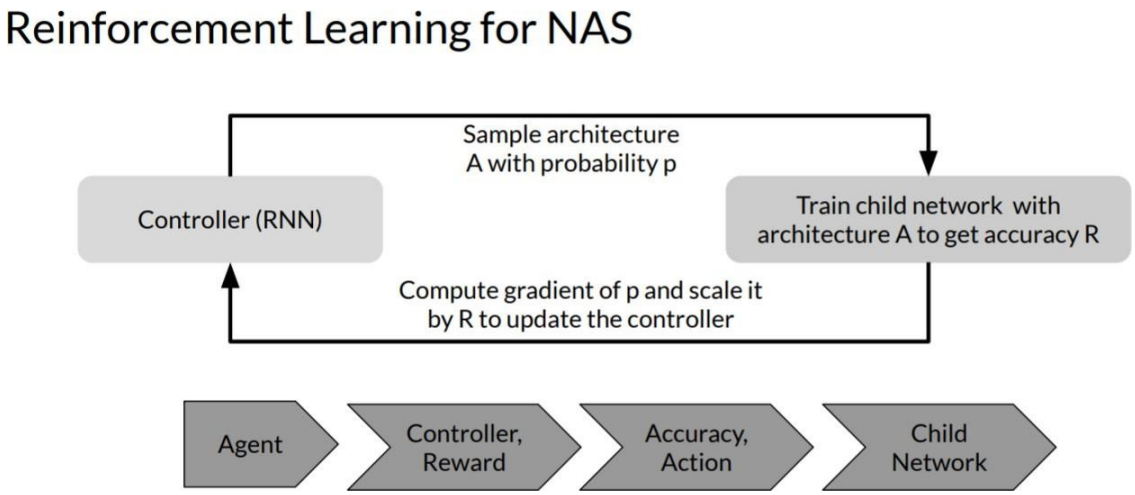
- La búsqueda por arquitectura neuronal también puede utilizar un **método evolutivo** para buscar, y así es como funciona.
- En primer lugar, se genera aleatoriamente una población inicial de n arquitecturas de modelos diferentes. El rendimiento de cada individuo, es decir, de cada arquitectura, se evalúa según lo definido por la estrategia de estimación del rendimiento
- A continuación, los X con mejores resultados son **seleccionados como padres de una nueva generación**.
- Esta nueva generación de arquitecturas podrían ser copias de los respectivos progenitores con alteraciones o mutaciones aleatorias inducidas. O pueden surgir de combinaciones de los progenitores, se evalúa el rendimiento de la descendencia. De nuevo utilizando la estrategia de estimación del rendimiento.
- La lista de posibles mutaciones puede incluir operaciones como añadir o eliminar una capa, añadir o eliminar una conexión, cambiar el tamaño de una capa o cambiar otro hiperparámetro.
- Se seleccionan Y arquitecturas para eliminarlas de la población. Puede tratarse de los Y individuos con peores resultados, de los Y individuos más viejos de la población o de una selección de individuos basada en una combinación de estos parámetros.
- La descendencia sustituye a las arquitecturas eliminadas y el proceso se reinicia con esta nueva población.

Reinforcement Learning

- Agents goal is to maximize a reward
- The available options are selected from the search space
- The performance estimation strategy determines the reward

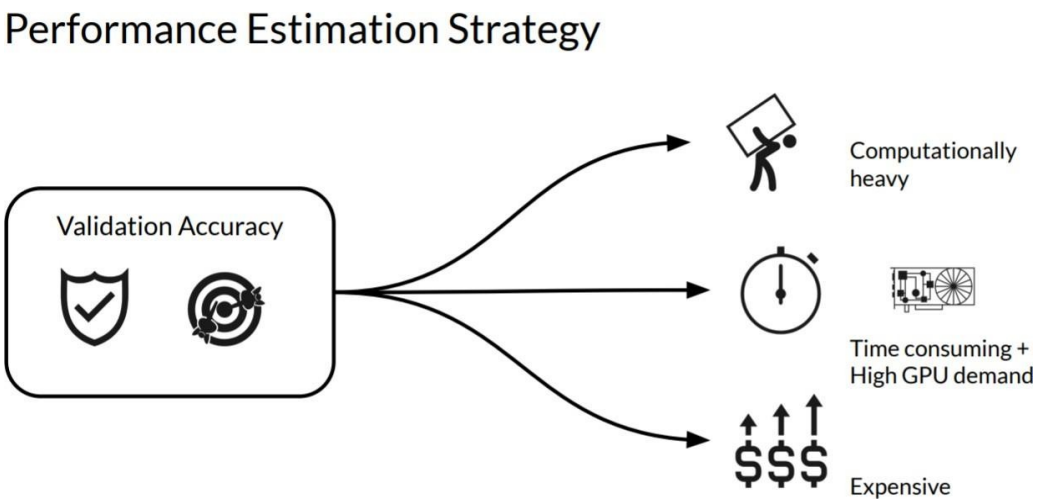


- En el **aprendizaje por refuerzo,** los agentes realizan acciones en un entorno, tratando de maximizar una recompensa.
- Después de cada acción, se actualiza el estado del agente y del entorno y se emite una recompensa basada en una métrica de rendimiento; a continuación, se evalúa el rango de posibles acciones siguientes
- En este caso, el entorno es nuestro espacio de búsqueda y la función de recompensa es nuestra estrategia de estimación del rendimiento.



- Una red neuronal también puede especificarse mediante una cadena de longitud variable en la que los elementos de la cadena especifican las capas individuales de la red.
- Eso nos permite utilizar una red neuronal recurrente o RNN para generar esa cadena, como podríamos hacer para un modelo de PNL.
- La RNN que genera la cadena se denomina **controlador**
- Después de entrenar la red denominada red hija con datos reales, podemos medir la precisión en el conjunto de validación
- En este caso, la precisión determina la recompensa del aprendizaje por refuerzo.
- Basándonos en la precisión, podemos calcular el **gradiente de la política** para actualizar el controlador RNN.
- En la siguiente iteración, el controlador habrá aprendido a dar mayores probabilidades a las arquitecturas que den lugar a mayores precisiones durante el entrenamiento.
- Así es como el controlador aprenderá a mejorar su búsqueda con el tiempo,
- Por ejemplo, en el conjunto de datos CIFAR-10, este método, partiendo de cero, puede diseñar una nueva arquitectura de red que rivaliza con la mejor arquitectura diseñada por humanos en términos de precisión del conjunto de pruebas.

Medición de la eficacia del AutoML



- La búsqueda de arquitecturas neuronales se basa en poder medir la precisión o eficacia de las distintas arquitecturas que prueba. Esto requiere una estrategia de estimación del rendimiento.

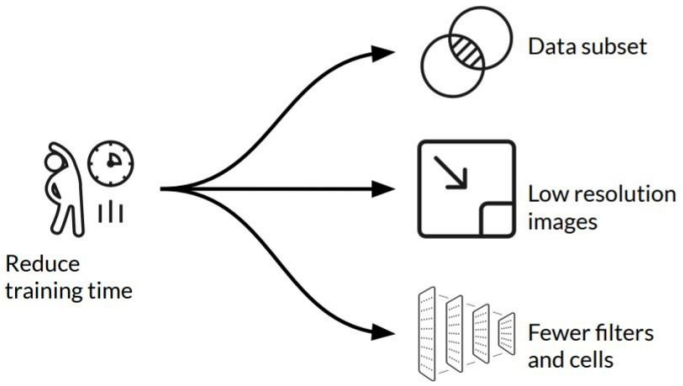
- Las estrategias de búsqueda en la búsqueda de arquitecturas neuronales necesitan estimar el rendimiento de las arquitecturas generadas para poder generar arquitecturas que tengan un mejor rendimiento.
- Analicemos algunas de las estrategias de estimación del rendimiento que se utilizan en la búsqueda de arquitecturas neuronales.
- El enfoque más sencillo consiste en medir la **precisión de validación** de cada arquitectura que se genera, como vimos con el enfoque del aprendizaje por refuerzo.
- Esto se convierte en una **pesadez computacional**, especialmente para grandes espacios de búsqueda y redes complejas. Y como resultado, puede llevar varios días de GPU encontrar las mejores arquitecturas utilizando este enfoque, lo que lo hace caro y lento. También hace que la búsqueda de arquitecturas neuronales resulte **poco práctica** para muchos casos de uso.

Strategies to Reduce the Cost

1. Lower fidelity estimates
2. Learning Curve Extrapolation
3. Weight Inheritance/ Network Morphisms

- ¿Hay alguna forma de reducir la estimación de los costes de rendimiento? Se han propuesto varias estrategias, como las estimaciones de menor fidelidad, la extrapolación de la curva de aprendizaje y la herencia de pesos o el morfismo de red. Analicemos cada una de ellas.

Lower Fidelity Estimates

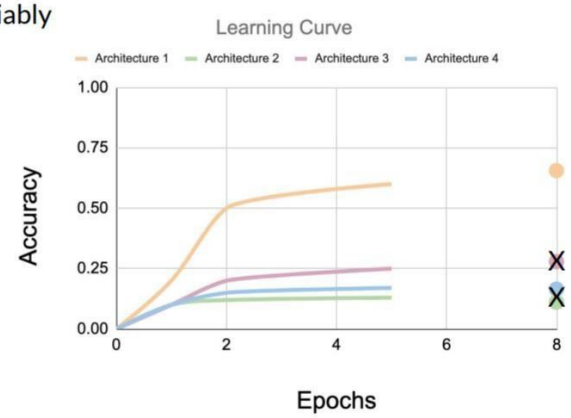


- Reduce cost but underestimates performance
- Works if **relative ranking** of architectures does not change due to lower fidelity estimates
- Recent research shows this is not the case

- Las estimaciones de menor fidelidad o precisión intentan reducir el tiempo de entrenamiento replanteando el problema para que sea más fácil de resolver entrenando sobre un **subconjunto de datos o utilizando imágenes de menor resolución, por** ejemplo, o usando **menos filtros por capa y menos celdas**.
- Reduce mucho el coste computacional, pero acaba subestimando el rendimiento. Eso está bien si puedes asegurarte de que la clasificación relativa de las arquitecturas no cambia debido a sus estimaciones de menor fidelidad.
- Pero, por desgracia, investigaciones recientes han demostrado que no es así. ¿Qué más podemos probar?

Learning Curve Extrapolation

- Requires predicting the learning curve reliably
- Extrapolates based on initial learning.
- Removes poor performers



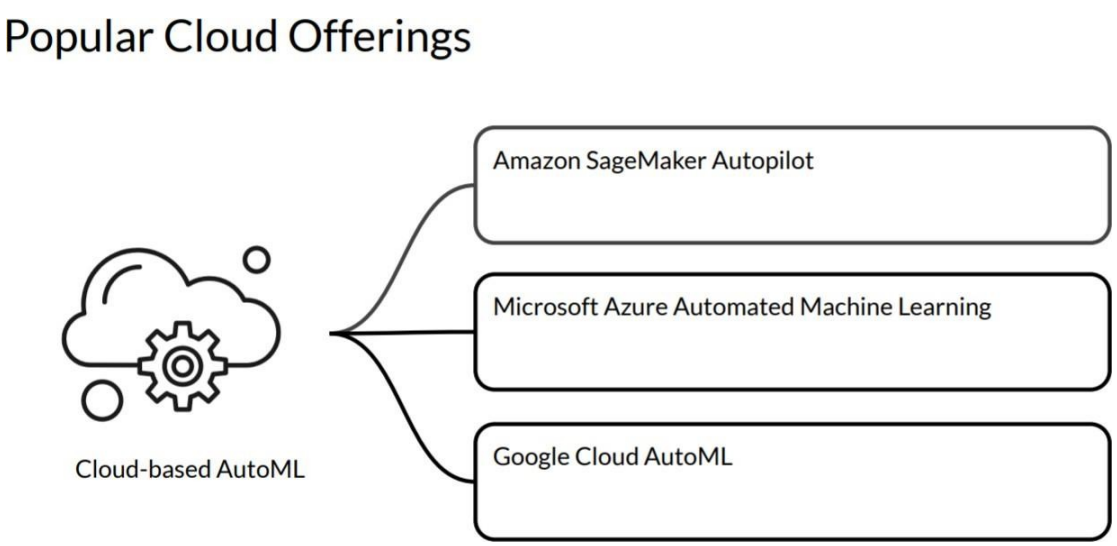
- La extrapolación de la curva **de** aprendizaje se basa en el supuesto de que se dispone de mecanismos para **predecir la curva de aprendizaje de forma fiable,** por lo que la extrapolación es una opción sensible y válida.
- A la derecha se muestran curvas de aprendizaje para distintas arquitecturas. Basándose en unas pocas iteraciones y en los conocimientos disponibles, el método extrapola las curvas de aprendizaje iniciales y da por terminadas todas las arquitecturas que obtienen malos resultados.
- El algoritmo de búsqueda progresiva de arquitecturas neuronales, que es uno de los enfoques para la búsqueda de arquitecturas neuronales, utiliza un método similar entrenando un **modelo sustituto** y utilizándolo para predecir el rendimiento utilizando propiedades arquitectónicas.

Weight Inheritance/Network Morphisms

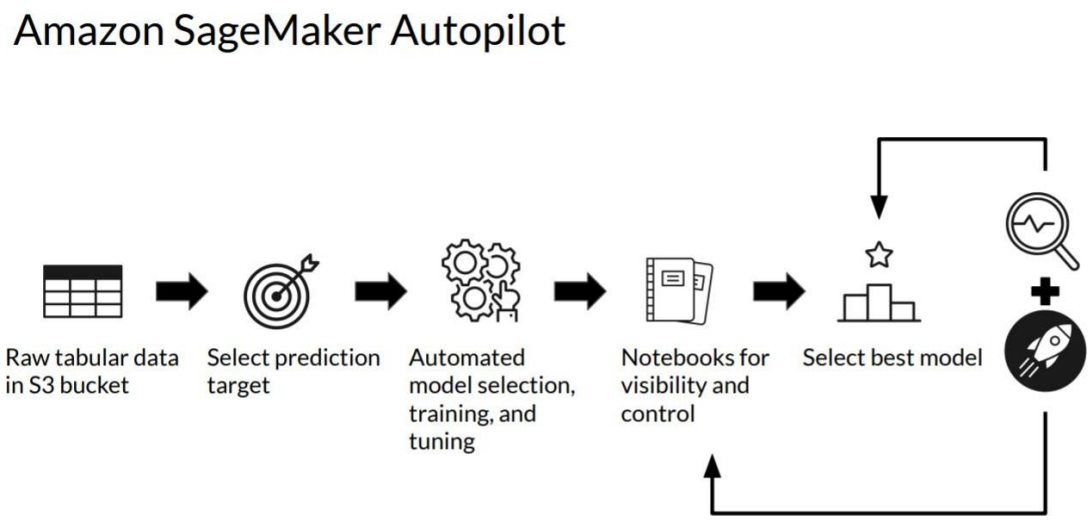
- Initialize weights of new architectures based on previously trained architectures
  - Similar to transfer learning
- Uses **Network Morphism**
- Underlying function unchanged
  - New network inherits knowledge from parent network.
  - Computational speed up: only a few days of GPU usage
  - Network size not inherently bounded

- Otro método para acelerar la búsqueda de arquitecturas inicializa los pesos de las nuevas arquitecturas basándose en los pesos de otras arquitecturas que han sido entrenadas anteriormente, de **forma similar a como funciona el aprendizaje por transferencia**.
- Una forma de conseguirlo se denomina morfismo de red.
- El morfismo de red modifica la arquitectura sin cambiar la función subyacente.
- Esto es ventajoso porque la red hereda conocimientos de la red matriz, lo que da lugar a métodos cuyo diseño y evaluación sólo requieren unos pocos días de GPU.
- Esto permite aumentar sucesivamente la capacidad de la red y conservar un alto rendimiento **sin necesidad de entrenarla desde cero.**
- Una ventaja de este enfoque es que permite bases de búsqueda que no tienen un límite superior inherente al tamaño de la arquitectura.

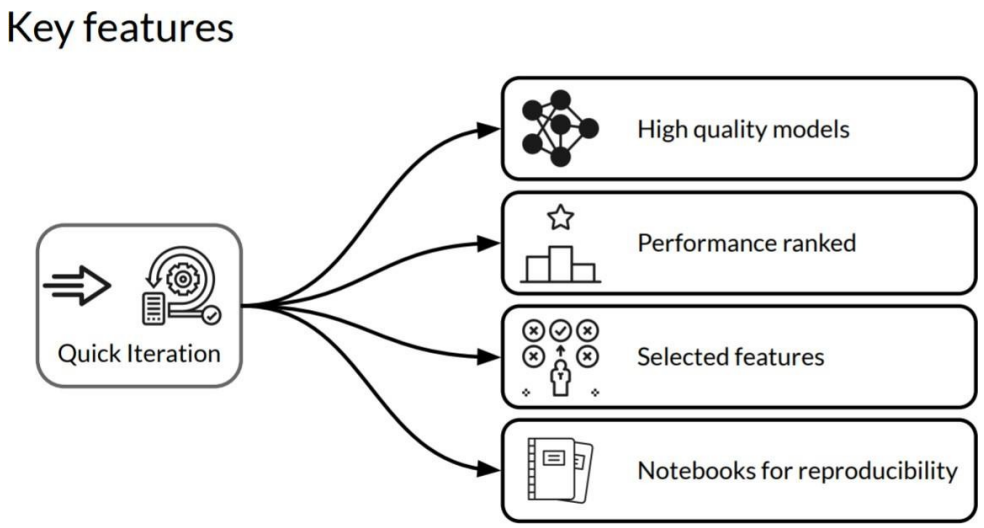
AutoML en la nube



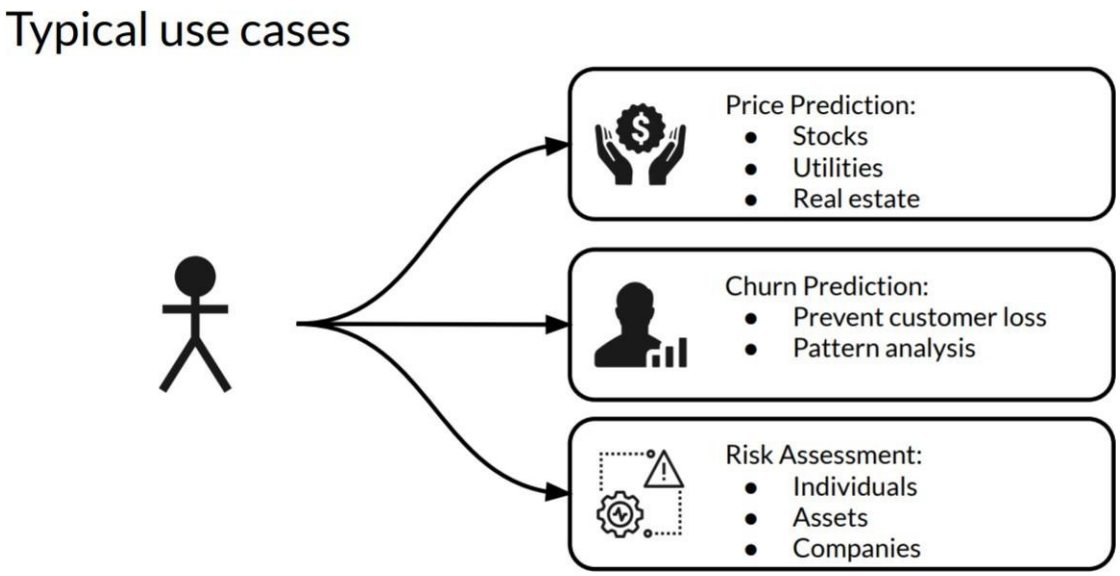
- Una de las formas más sencillas de utilizar AutoML es mediante un servicio en la nube.
- Revisaremos tres ofertas populares: Amazon SageMaker Autopilot, Microsoft Azure Automated Machine Learning y Google Cloud AutoML.
- En los ejercicios, entrenarás un modelo utilizando Google Cloud AutoML.



- Empecemos con el piloto automático de Amazon SageMaker.
- Amazon SageMaker Autopilot entrena y ajusta automáticamente los mejores modelos de aprendizaje automático para clasificación o regresión en función de sus datos, al tiempo que le permite mantener el control y la visibilidad totales.
- A partir de sus datos brutos, usted selecciona una etiqueta u objetivo. A continuación, Autopilot busca modelos candidatos para que usted los revise y elija.
- Todos estos pasos están **documentados** en cuadernos ejecutables que le ofrecen un control y una reproducibilidad totales del proceso.
- Incluye una tabla de candidatos a modelo que le ayudará a elegir el que mejor se adapte a sus necesidades.
- A continuación, puede desplegar el modelo en producción o iterar sobre las soluciones recomendadas para mejorar aún más la calidad del modelo.

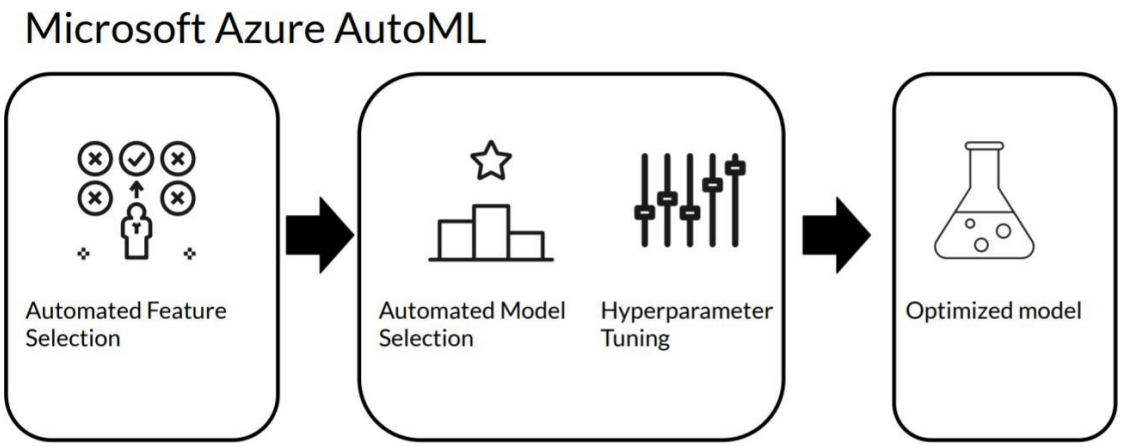


- El piloto automático está optimizado para una iteración rápida. Genera modelos de alta calidad con rapidez.
- Tras el conjunto inicial de iteraciones, el piloto automático crea una tabla clasificatoria de modelos ordenados por rendimiento. Puede ver qué características de su conjunto de datos fueron seleccionadas por cada modelo. A continuación, puede implementar un modelo en producción.
- El proceso de generación de los modelos es completamente transparente. Autopilot permite crear un cuaderno SageMaker a partir de cualquier modelo creado por él.
- A continuación, puede consultar el Cuaderno para profundizar en los detalles de la aplicación de los modelos.
- En caso necesario, puede perfeccionar el modelo y volver a crearlo desde el Cuaderno en cualquier momento.

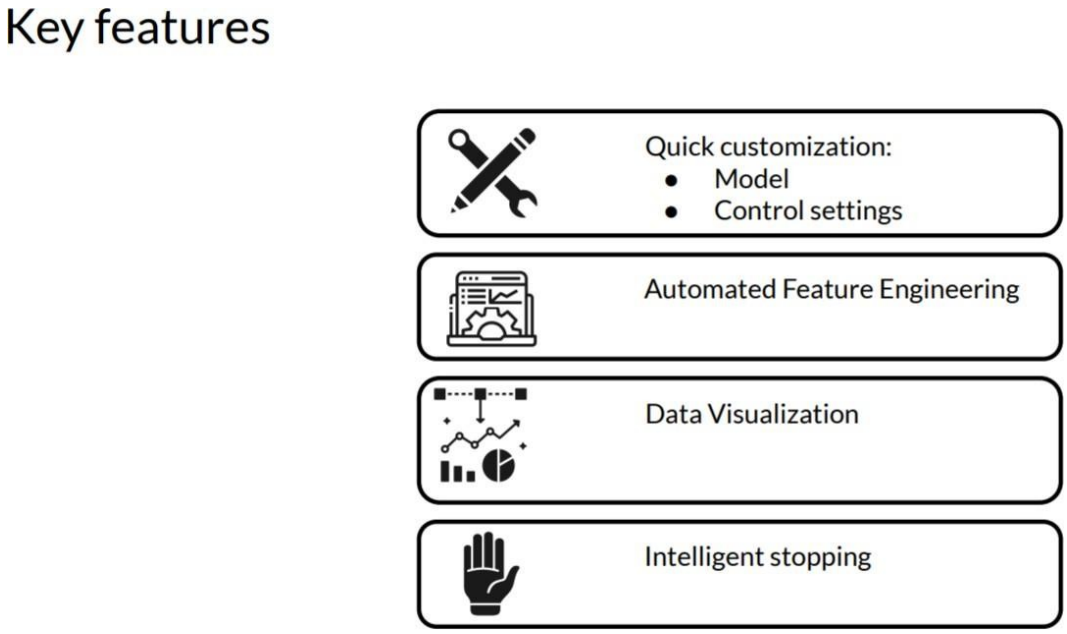


- Amazon sugiere que el piloto automático puede aplicarse a distintos casos de uso.
- Por ejemplo, puede utilizarse para predecir precios futuros y ayudarle a tomar decisiones de inversión acertadas basándose en sus datos históricos, como la demanda, las tendencias estacionales y el precio de otras materias primas.
- Las predicciones de precios son especialmente útiles en el sector de los servicios financieros, para predecir el precio de las acciones, o en el inmobiliario, para predecir los precios de los inmuebles, o en el de la energía y los servicios públicos, para predecir los precios de los recursos naturales.
- La predicción del churn puede ser útil para predecir la pérdida de clientes o churn. Las empresas siempre buscan formas de eliminar la pérdida de clientes.
- La predicción de bajas funciona aprendiendo patrones en los datos existentes e identificando patrones en los nuevos conjuntos de datos, de modo que el modelo pueda predecir qué clientes tienen más probabilidades de darse de baja.
- Otra aplicación es la evaluación de riesgos. La evaluación de riesgos requiere identificar y analizar posibles sucesos que puedan afectar negativamente a las personas, los activos y la empresa.

- Los modelos de evaluación de riesgos se entrenan utilizando sus conjuntos de datos existentes para que pueda optimizar las predicciones de los modelos para su empresa.

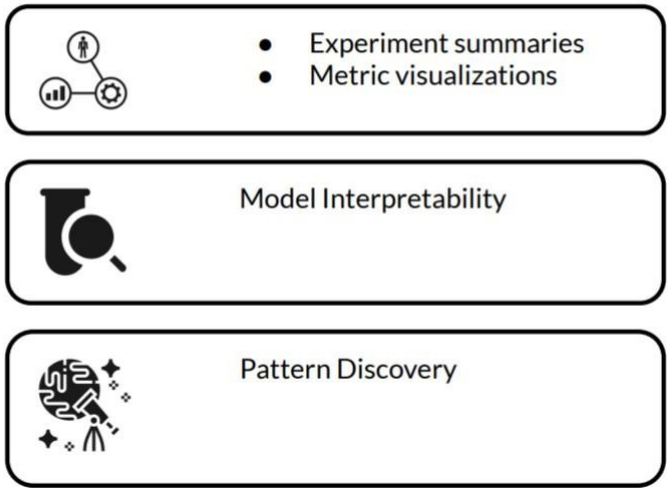


- Microsoft Azure Automated Machine Learning tiene como objetivo capacitar a los científicos de datos profesionales y no profesionales para crear modelos de aprendizaje automático rápidamente.
- Automatiza la larga e iterativa tarea del desarrollo de modelos, así que básicamente hace AutoML.
- Comienza con la selección automática de características, seguida de la selección del modelo y el ajuste de hiperparámetros en el modelo seleccionado para generar el modelo más optimizado para la tarea en cuestión.



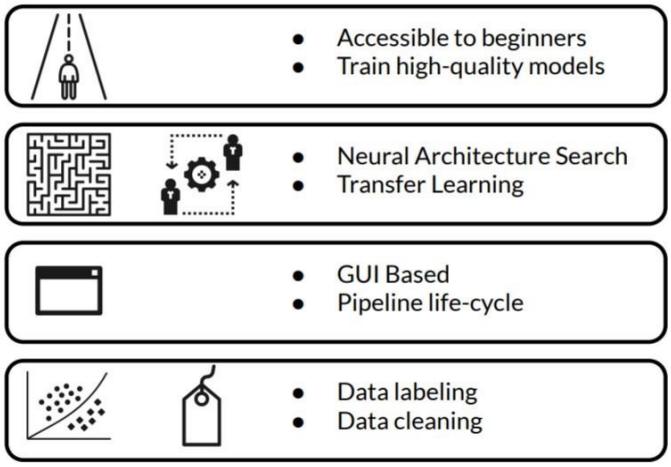
- Puede crear sus modelos utilizando una interfaz de usuario sin código o utilizando cuadernos de código inicial.
- Puede personalizar rápidamente sus modelos, aplicar ajustes de control a las iteraciones, umbrales, validaciones, algoritmos bloqueados y otros criterios experimentales.
- También proporciona herramientas para automatizar completamente el proceso de ingeniería de características.
- También puede visualizar y perfilar fácilmente sus datos para detectar tendencias, así como descubrir errores comunes e incoherencias en sus datos.
- Esto le ayudará a comprender mejor las acciones recomendadas y a aplicarlas automáticamente.
- También proporciona una parada inteligente para ahorrar tiempo de cálculo y priorizar la métrica principal y el submuestreo para agilizar las ejecuciones de los experimentos y acelerar los resultados.

Key features



- También incorpora resúmenes de las ejecuciones de los experimentos y visualizaciones detalladas de las métricas para ayudarle a comprender sus modelos y comparar su rendimiento.
- La interpretabilidad del modelo ayuda a evaluar el ajuste del modelo a las características brutas y de ingeniería y proporciona información sobre la importancia de las características.
- Puede descubrir patrones, realizar análisis hipotéticos y desarrollar una comprensión más profunda de los modelos para respaldar la transparencia y la confianza en su empresa.

Google Cloud AutoML



- Google Cloud AutoML es un conjunto de productos de aprendizaje automático que permite a los desarrolladores con conocimientos limitados de aprendizaje automático entrenar modelos de alta calidad específicos para sus necesidades empresariales.
- Se basa en las tecnologías de búsqueda de Google, de última generación, de aprendizaje por transferencia y de arquitectura neuronal.
- Cloud AutoML aprovecha más de 10 años de investigación de Google para ayudar a sus modelos de aprendizaje automático a lograr un rendimiento más rápido y predicciones más precisas.
- Puede utilizar Cloud AutoML, una sencilla interfaz gráfica de usuario para entrenar, evaluar, mejorar y desplegar modelos basados en sus datos.
- Está a sólo unos minutos de tener su propio modelo de aprendizaje automático personalizado.
- El servicio de etiquetado humano de Google también puede poner a un equipo de personas a trabajar anotando o limpiando tus etiquetas para asegurarse de que tus modelos se están entrenando con datos de alta calidad.

Cloud AutoML Products

Sight	Auto ML Vision Derive insights from images in the cloud or at the edge.	Auto ML Video Intelligence Enable powerful content discovery and engaging video experiences.
Language	AutoML Natural Language Reveal the structure and meaning of text through machine learning.	Auto ML Translation Dynamically detect and translate between languages.
Structured Data	AutoML Tables Automatically build and deploy state-of-the-art machine learning models on structured data.	

- Dado que los distintos problemas y los distintos datos deben tratarse de forma diferente, Cloud AutoML no es una sola cosa.
- Es un conjunto de productos diferentes, cada uno enfocado a casos de uso y tipos de datos concretos.
- Por ejemplo, para datos de imagen, existe AutoML Vision y para datos de vídeo, AutoML Video Intelligence, para lenguaje natural, AutoML natural language y para traducción, AutoML Translation.
- Por último, para datos estructurados en general, está AutoML Tables.

AutoML Vision Products

Auto ML Vision Classification	AutoML Vision Edge Image Classification
AutoML Vision Object Detection	AutoML Vision Edge Object Detection

Steps to Classify Images using AutoML Vision

- Algunas de ellas se desglosan aún más. En el caso de los datos de imágenes, por ejemplo, existen la visión y la clasificación y la detección de objetos por visión.

- También existen versiones Edge de ambas, centradas en la optimización para la ejecución de inferencia en el borde en aplicaciones móviles o dispositivos IoT.

AutoML Video Intelligence Products

AutoML Video Intelligence Classification

Enables you to train machine learning models, to classify shots and segments on your videos according to your own defined labels.

AutoML Video Object detection

Enables you to train machine learning models to detect and track multiple objects, in shots and segments.

- En cuanto al vídeo, existe la clasificación Video Intelligence y la detección de objetos de vídeo. De nuevo, centradas en estos casos de uso específicos.

So what's in the secret sauce?

How do these Cloud offerings perform AutoML?

- We don't know (or can't say) and they're not about to tell us
- The underlying algorithms will be similar to what we've learned
- The algorithms will evolve with the state of the art

- ¿Cómo funcionan estos tres servicios en la nube?
- Bueno, nadie sabe exactamente o no puede decir si trabaja para un proveedor determinado.
- Sin embargo, es seguro asumir que los algoritmos en juego serán similares a los que hemos discutido en lecciones anteriores.
- Además, tenga en cuenta que la ingeniería de ML para la producción es un campo que cambia rápidamente y que cada pocos meses surgen nuevas tecnologías y avances.
- Los proveedores de AutoML suelen incorporar y explotar al máximo estos nuevos avances.
- Observe también que cada una de estas ofertas de AutoML realiza las mismas operaciones que usted realiza o debería realizar al entrenar su modelo, incluida la selección de características y la ingeniería de características, así como la limpieza de sus etiquetas. Los diseñadores de AutoML comprenden la importancia de estas actividades.

Referencias

- Búsqueda de arquitectura neuronal
- Optimización bayesiana
- Búsqueda de arquitectura neuronal con aprendizaje por refuerzo
- Búsqueda progresiva de arquitecturas neuronales
- Morfismo de la red
- Piloto automático Amazon SageMaker
- Aprendizaje automático de Microsoft Azure
- Google Cloud AutoML