Inteligencia Artificial

CC-421

César Lara Avila

Universidad Nacional de Ingeniería

(actualización: 2021-01-17)

Bienvenidos

Conceptos fundamentales de Aprendizaje Profundo

- Entrenamiento de un modelo
- Regularización
- Selección de hiperparámetros
- Disponibilidad de datos y calidad
- Discusión

Entrenamiento de un modelo

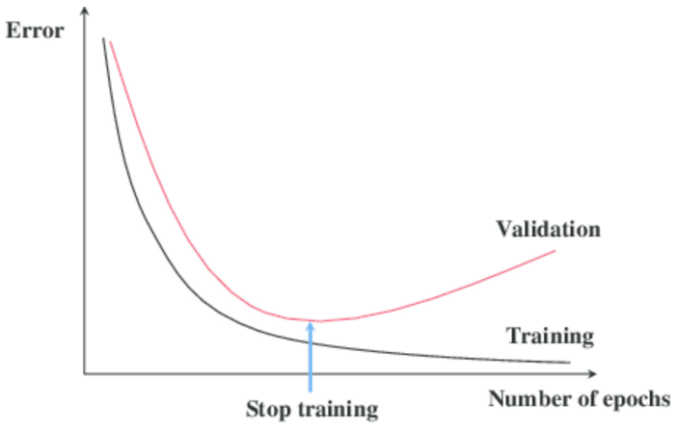
- El aprendizaje profundo es más propenso al sobreajuste. Con muchos parámetros libres, puede ser relativamente fácil encontrar una ruta para lograr $E = 0$. Se ha demostrado que muchas arquitecturas estándar de aprendizaje profundo pueden entrenarse con un etiquetado aleatorio de los datos de entrenamiento y lograr $E = 0$.
- El aprendizaje profundo se basa en encontrar una solución a un problema de optimización no convexo que es NP-completo para una función general no convexa. En la práctica, verás que calcular el mínimo global para una red profunda bien regularizada es irrelevante porque los mínimos locales suelen ser más o menos similares y se acercan al mínimo global a medida que aumenta la complejidad del modelo.
- Sin embargo, en una red mal regularizada, los mínimos locales pueden producir una gran pérdida, lo que no es deseable.

El mejor modelo es aquel que logra la brecha más pequeña entre la pérdida de entrenamiento y la pérdida de validación, sin embargo, seleccionar la configuración de arquitectura correcta y la técnica de entrenamiento puede ser agotador y complicado.

Early Stopping

- Una de las formas más prácticas en las que podemos evitar que un modelo se sobreajuste es el *early stopping*. Esto depende de la suposición: A medida que disminuye el error de validación, el error de prueba también debería disminuir.

- Al entrenar, calculamos el error de validación en distintos puntos (generalmente al final de cada época) y mantenemos el modelo con el error de validación más bajo, como se muestra en la siguiente figura,



La simplicidad del early stopping la convierte en la forma de regularización más utilizada en el aprendizaje profundo.

Desaparición y explosión del gradiente

- Cuando se entrena redes neuronales con muchas capas, con propagación hacia atrás, surge el problema de la desaparición/explosión de gradientes. Durante la propagación atrás, estamos multiplicando el gradiente por la salida de cada capa sucesiva. Esto significa que el gradiente puede hacerse más y más grande si $\nabla E > 1$ o $\nabla E < 1$ y más pequeño si el gradiente es $-1 < \nabla E < 0$, ya que se multiplica por cada capa sucesiva.

- En la práctica, esto significa que, en el caso de la desaparición de gradiente, muy poco del error se propaga a las capas anteriores de la red, lo que hace que el aprendizaje sea muy lento o inexistente.

- Para los gradientes explosivos, esto hace que los pesos se desborden, lo que impide el aprendizaje. Cuanto más profunda se vuelve una red neuronal, mayor es el problema.

- En el caso de la explosión de gradientes, una solución simple y práctica es *recortar (clip)* los gradientes estableciendo un máximo para los valores de gradiente en cada paso de propagación hacia atrás para controlar el crecimiento de los pesos.

Inicialización de Xavier y He

La inicialización de parámetros es un aspecto crítico del entrenamiento de una red neuronal profunda.

Para que el entrenamiento fluya correctamente durante la dirección hacia adelante y la propagación hacia atrás, los autores, Xavier Glorot y Yoshua Bengio indican que necesitamos que la varianza de las salidas de cada capa sea igual a sus entradas.

- **Inicializador normal de Xavier**:

$$var(w_{ij}) = \frac{2}{n_{entrada} + n_{salida}}$$

- **Inicializador normal de He**

$$var(w_{ij}) = \frac{2}{n_{entrada}}$$

donde $n_{entrada}$ y n_{salida} son los números de conexiones de entrada y salida para la capa cuyos pesos se están inicializando.

Descenso de gradiente por lote y mini-lotes

- El descenso de gradiente por lotes es una variante del descenso de gradiente que evalúa el error en todo el conjunto de datos antes de actualizar el modelo acumulando el error después de cada ejemplo.
- Este enfoque rara vez se usa en la práctica con el aprendizaje profundo.
- Un compromiso adecuado entre estas dos estrategias es el descenso de gradiente de mini-lotes. El descenso del gradiente de mini-lotes divide el conjunto de datos en lotes, y el modelo acumula el error en un mini-lote antes de realizar una actualización. Este enfoque ofrece una variedad de ventajas, que incluyen:
    - Reducción de ruido en cada actualización del modelo debido a la acumulación de gradientes de múltiples ejemplos de entrenamiento.
    - Mayor eficiencia que SGD
    - Entrenamiento más rápido al aprovechar las ventajas de las operaciones matriciales para reducir el tiempo de IO.

- Una desventaja del descenso del gradiente de mini-lotes es la adición del tamaño de mini-lotes como hiperparámetro.
- El tamaño de mini-lote, a menudo llamado tamaño de *lote* por conveniencia, generalmente se establece en función de las limitaciones de hardware del modelo para no exceder la memoria de la CPU o GPU.
- Los tamaños de lote son son típicamente potencias de 2, debido a implementaciones de hardware comunes. En general, es deseable lograr un equilibrio con un tamaño de lote pequeño que produzca una convergencia más rápida y un tamaño de lote más grande que converja más lentamente pero con estimaciones más precisas.

Se recomienda revisar las curvas de aprendizaje de algunos tamaños de lotes diferentes para decidir el mejor tamaño.

Regularización

Prácticamente, el control del error de generalización se logra mediante la creación de un modelo grande que se regulariza adecuadamente. La regularización puede tomar muchas formas.

Algunos métodos se centran en reducir la capacidad de los modelos penalizando los parámetros anormales en la función objetivo agregando un término de regularización.

$$E(\mathbf{W}; \hat{\mathbf{y}}, \mathbf{y}) = E(\hat{\mathbf{y}}, \mathbf{y}) + \Omega(\mathbf{W})$$

donde $\mathbf{W}$ es el peso de la red.

Algunos enfoques se centran en limitar la información proporcionada a la red (por ejemplo, dropout) o normalizar la salida de las capas (normalización por lotes), mientras que otros pueden hacer cambios en los datos directamente.

Regularización L_2 : disminución de peso

La disminución de peso agrega un término de regularización a la función de error que empuja los pesos hacia el origen, penalizando las variaciones de peso altas.

La disminución de peso introduce un escalar α que penaliza los pesos que se alejan del origen.

Esto funciona como un Gaussiano de media cero en el objetivo de entrenamiento, lo que limita la libertad de la red para aprender grandes pesos que podrían estar asociados con el sobreajuste.

La configuración de este parámetro se vuelve importante porque si el modelo es demasiado limitado, es posible que no pueda aprender.

La regularización de L_2 se define como:

$$\Omega(\mathbf{W}) = \frac{\alpha}{2}\mathbf{W}^T\mathbf{W}$$

La función de pérdida se puede describir como:

$$E(\mathbf{W}; \hat{\mathbf{y}}, \mathbf{y}) = \frac{\alpha}{2}\mathbf{W}^T\mathbf{W} + E(\hat{\mathbf{y}}, \mathbf{y})$$

Con el gradiente siendo:

$$\nabla_{\mathbf{W}} E(\mathbf{W}; \hat{\mathbf{y}}, \mathbf{y}) = \alpha\mathbf{W} + \nabla_{\mathbf{W}} E(\hat{\mathbf{y}}, \mathbf{y})$$

Y la actualización de parámetros se convierte en:

$$\mathbf{W} = \mathbf{W} - \epsilon(\alpha\mathbf{W} + \nabla_{\mathbf{W}} E(\hat{\mathbf{y}}, \mathbf{y}))$$

donde ϵ es la tasa de aprendizaje.

Regularización L_1

Un método de regularización menos común es la regularización L_1. Esta técnica también funciona como una penalización de peso.

El regularizador es una suma de los valores absolutos de los pesos:

$$\Omega(\mathbf{W}) = \alpha \sum |w_i|$$

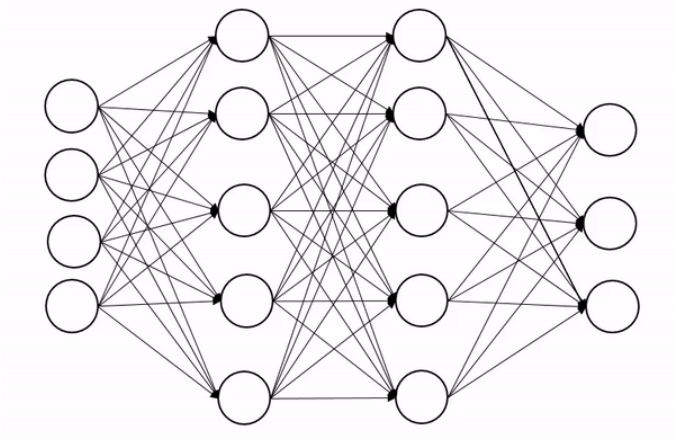
A medida que avanza el entrenamiento, muchos de los pesos se convertirán en cero, introduciendo la sparsity en los pesos del modelo.

Esto se usa a menudo en la selección de características, pero no siempre es una cualidad deseable con las redes neuronales.

Dropout

El dropout ha sido un método simple y altamente efectivo para reducir el sobreajuste de las redes neuronales. Se deriva de la idea de que las redes neuronales pueden tener conexiones muy frágiles desde la entrada hasta la salida. Estas conexiones aprendidas pueden funcionar para los datos de entrenamiento, pero no se generalizan a los datos de prueba.

El dropout tiene como objetivo corregir esta tendencia al *desconectar* aleatoriamente las conexiones en el proceso de entrenamiento de la red neuronal para que una predicción no pueda depender de ninguna neurona durante el entrenamiento.



Aplicar el dropout a una red implica aplicar una máscara aleatoria muestreada de una distribución de Bernoulli con una probabilidad de p.

Esta matriz de máscara se aplica por elementos (multiplicación por 0) durante la operación de avance. Durante el paso de propagación hacia atrás, los gradientes para cada parámetro y los parámetros que enmascaran el gradiente se establecen en 0 y otros gradientes se amplían en $1/1 - p$.

Agregar droput a los modelos de PyTorch es muy sencillo con la clase *torch.nn.Dropout*, que toma la tasa de dropout, la probabilidad de que una neurona se desactive, como parámetro.

```
self.dropout = nn.Dropout(0.25)
```

Podemos aplicar el dropout después de cualquier capa que no sea de salida.

Aprendizaje multitarea

Debido a que el aprendizaje profundo se logra a través del cálculo y el descenso basados en gradientes, podemos optimizar simultáneamente para una variedad de funciones de optimización. Esto permite que nuestra representación subyacente aprenda una representación general que puede realizar múltiples tareas.

El aprendizaje multitarea se ha convertido en un enfoque ampliamente utilizado recientemente. La adición de tareas auxiliares puede ayudar a mejorar la señal del gradiente a los parámetros aprendidos que conducen a una mejor calidad en la tarea general.

Lectura: An Overview of Multi-Task Learning in Deep Neural Networks de Sebastian Ruder.

Parámetros compartidos

En algunas tareas, las entradas son lo suficientemente similares como para que no sea deseable aprender un conjunto diferente de parámetros para cada tarea, sino más bien compartir los aprendizajes en varios lugares.

Esto se puede lograr compartiendo un conjunto de pesos en diferentes entradas.

El uso compartido de parámetros no solo es útil como un regularizador, sino que también proporciona múltiples beneficios de entrenamiento, como memoria reducida (una copia de un conjunto de pesos) y un número reducido de parámetros único de modelos.

Un enfoque que aprovecha el intercambio de parámetros es una red neuronal convolucional.

Normalización por lotes

Durante el proceso de entrenamiento, puede haber mucha variación en los ejemplos de entrenamiento que conducen a la introducción de ruido en el proceso de entrenamiento.

La normalización reduce la cantidad que los pesos necesitan cambiar para adaptarse a un ejemplo específico, manteniendo la misma propiedad de distribución. Con el aprendizaje profundo, tenemos múltiples capas de cálculo con valores ocultos que se pasan a las capas posteriores.

Es probable que la salida de cada una de estas capas sea una entrada no normalizada y es probable que la distribución cambie con frecuencia durante el proceso de entrenamiento. Este proceso se conoce comúnmente como *cambio de covariables interno*.

La normalización por lotes tiene como objetivo reducir este proceso en una red mediante la normalización de las salidas de las capas intermedias durante el entrenamiento. Esto acelera el proceso de entrenamiento y permite mayores tasas de aprendizaje sin correr el riesgo de divergencia.

La normalización por lotes logra esto al normalizar la salida de la capa oculta anterior por la media y la varianza del lote (mini-lote).

Sin embargo, esta normalización afectaría la fase de inferencia y por lo tanto, la normalización por lotes captura un promedio de la media y la varianza y los fija en el tiempo de inferencia.

Para un mini-lote de entrada $\beta = \{x_{1:m}\}$, aprendemos los parámetros γ y β a través de:

$$\mu_\beta = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_\beta^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2$$

$$\hat{x}_i = \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}}$$

$$y_i = \gamma\hat{x}_i + \beta.$$

En Pytorch, la implementación es: `bn = nn.BatchNorm2d()`.

Gradiente clipping

Una técnica popular para resolver el problema de la explosión de gradientes es simplemente recortar los gradientes durante la propagación hacia atrás para que nunca excedan algún umbral (se usa principalmente para redes neuronales recurrentes).

El objetivo del recorte de gradientes es cambiar la escala de los gradientes para que su norma tenga un valor particular.

Con el recorte del gradiente, se introduce un umbral de gradiente predeterminado y las normas de gradiente que exceden este umbral se reducen para que coincidan con la norma.

Selección de hiperparámetros

La mayoría de las técnicas de aprendizaje y los métodos de regularización tienen alguna forma de configuración de parámetros de entrenamiento asociados a ellos.

La tasa de aprendizaje, el momentum, la probabilidad del dropout y la disminución de pesos, por ejemplo, deben seleccionarse para cada modelo.

Seleccionar la mejor combinación de estos hiperparámetros puede ser una tarea difícil.

Ajuste manual

Se recomienda el ajuste manual de hiperparámetros cuando se aplica un modelo existente a un nuevo conjunto de datos o un nuevo modelo a un conjunto de datos existente.

La selección manual ayuda a proporcionar intuición sobre la red. Esto puede ser útil para comprender si un conjunto particular de parámetros hará que la red se sobreajuste o no.

Por experiencia se recomienda controlar la norma de los gradientes y la rapidez con que converge o diverge la pérdida de un modelo. En general, la tasa de aprendizaje es el hiperparámetro más importante, que tiene el mayor impacto en la capacidad efectiva de la red.

La selección de la tasa de aprendizaje adecuada para un modelo permitirá una buena convergencia y el early stopping evitará que el modelo se sobreajuste al conjunto de entrenamiento.

Ajuste automatizado

La selección automática de hiperparámetros es un método mucho más rápido y robusto para optimizar una configuración de entrenamiento. La búsqueda grid es la técnica más común y directa.

La búsqueda aleatoria de hiperparámetros a veces es más robusta a los matices del entrenamiento, ya que algunas combinaciones de hiperparámetros pueden tener un efecto acumulativo. Similar a una búsqueda grid, la búsqueda aleatoria muestra aleatoriamente valores en el rango de la búsqueda grid en lugar de muestras espaciadas uniformemente.

Por lo general, la mayoría de los modelos explorados con búsqueda grid y búsqueda aleatoria están sujetos a combinaciones deficientes.

Esto puede aliviarse hasta cierto punto estableciendo límites apropiados para la búsqueda obtenida de la exploración manual, sin embargo, idealmente, el rendimiento del modelo puede usarse para determinar el siguiente conjunto de parámetros.

Se han introducido varios procedimientos de selección de hiperparámetros condicionados y Bayesianos para lograr esto.

Disponibilidad de datos y calidad

La regularización es la técnica más común para evitar el sobreajuste, pero también se puede lograr aumentando la cantidad de datos. Los datos son el componente más importante de cualquier modelo de aprendizaje automático.

Ajustar los hiperparámetros de la red neuronal suele ser el mejor paso apropiado para mejorar el error de generalización. Si todavía existe una brecha de rendimiento entre el error de entrenamiento y generalización, puede ser necesario aumentar la cantidad de datos (o la calidad en algunos casos).

Las redes neuronales pueden ser robustas ante cierta cantidad de ruido en un conjunto de datos y durante el proceso de entrenamiento, los efectos de los valores atípicos generalmente disminuyen. Sin embargo, los datos erróneos pueden causar muchos problemas.

El bajo rendimiento del modelo en aplicaciones del mundo real puede ser causado por etiquetas consistentemente incorrectas o datos insuficientes. Esto generalmente se manifiesta en una de las dos formas: sobreajuste o mala convergencia.

Aumento de datos

Una de las formas más fáciles de mejorar el rendimiento del modelo es introducir más datos de entrenamiento. Esta técnica puede ser particularmente beneficiosa para reducir el sobreajuste a anomalías específicas en un conjunto de datos.

En el caso de las imágenes, podemos imaginar una rotación y el giro horizontal como la creación de un par diferente $(\mathbf{X}, y)$ sin tener que volver a etiquetar ningún dato. Sin embargo, este no sería el caso de los números escritos a mano, donde un giro horizontal podría corromper la interpretación de la etiqueta.

Al incorporar el aumento de datos, debes asegurarte de tener en cuenta las restricciones del ejemplo y la relación objetivo.

Bagging

Esta técnica se basa en la idea de que podemos reducir la capacidad de los modelos de sobreajustar entrenando múltiples modelos en diferentes partes del conjunto de entrenamiento.

La técnica de bagging toma muestras del conjunto de datos original (con reemplazo), creando conjuntos de sub entrenamiento en los que se entrenan los modelos. Los modelos deben aprender diferentes características, ya que están aprendiendo diferentes partes de los datos, lo que lleva a un error de generalización más bajo después de combinar los resultados de cada modelo.

Esta estrategia tiende a usarse con menos frecuencia en la práctica debido al tiempo de cálculo de los modelos de aprendizaje profundo, los grandes requisitos de datos de los modelos profundos y la introducción de otros métodos de regularización (como dropout).

Entrenamiento Adversarial

Los ejemplos adversarios son ejemplos diseñados para hacer que un clasificador clasifique mal el ejemplo.

El espacio de parámetros libre de las redes neuronales significa que podemos encontrar ejemplos de entrada específicos que pueden aprovechar el conjunto específico de parámetros entrenados dentro de un modelo.

Debido a las propiedades de los ejemplos adversarios, podemos utilizar las técnicas utilizadas para crear ejemplos adversarios para producir datos de entrenamiento para la red, así como mejorar la solidez de la red al proporcionar ejemplos de entrenamiento que se enfocan en las áreas de incertidumbre en el espacio de parámetros.

Discusión

En términos generales, generalmente hay cuatro pilares en tensión al configurar y entrenar redes neuronales:

- Disponibilidad de datos (y calidad)
- Velocidad de cálculo
- Requisitos de memoria
- Calidad

En la práctica, generalmente es una buena idea establecer el objetivo final y trabajar hacia atrás para descubrir los límites de cada una de las restricciones.

En términos generales, la etapa inicial de la selección del modelo garantiza que el modelo tenga la capacidad de aprender de manera confiable.

En la práctica, generalmente no es necesario (ni factible) comenzar desde cero para cada nuevo tipo de modelo o tarea.

Computación y restricciones de memoria

Si bien numerosos avances hicieron posible el aprendizaje profundo, uno de los contribuyentes más importantes al reciente crecimiento en la adopción es, sin duda, las mejoras de hardware, particularmente las arquitecturas informáticas especializadas (GPU).

Las velocidades de procesamiento logradas con las GPU han sido uno de los factores más importantes que contribuyen a la popularidad y practicidad del aprendizaje profundo.

Las ventajas de velocidad a través de optimizaciones matriciales y la capacidad de procesamiento por lotes hacen que los problemas de aprendizaje profundo sean ideales para las arquitecturas de GPU.

Grandes conjuntos de datos y arquitecturas de aprendizaje profundo han llevado a mejoras de calidad significativas, sin embargo, el costo computacional de los modelos de aprendizaje profundo suele ser mayor que otros métodos de aprendizaje automático, que deben considerarse en entornos de recursos limitados.

Los requisitos del modelo también afectan la cantidad de optimización de hiperparámetros que se puede hacer. Es poco probable que se pueda realizar una búsqueda grid completa para modelos que tardan días o semanas en entrenarse.

El mismo razonamiento se aplica a cuestiones de memoria, con modelos más grandes que requieren más espacio. Sin embargo, se están introduciendo muchas técnicas de cuantificación para reducir el tamaño de los modelos, como cuantificar parámetros o usar valores de parámetros hash.

Fin!