Aspectos prácticos de entrenamiento de redes neuronales profundas (y en general modelos de ML)

Generalización: ¿qué tan bien se adaptará mi modelo a datos nuevos?

Hasta ahora cuando entrenamos una red lo que tratamos de hacer es minimizar el error de entrenamiento (training error). Esto no se diferencia de (simple) optimización.

- Error de generalización (o test error): Valor esperado del error para un dato que la red no ha visto durante el entrenamiento.

Esta es una estimación de qué tan bien/mal lo hará la red en el mundo real.

Objetivo central en Machine Learning:

***** No solo queremos minimizar el error de entrenamiento***
si no también minimizar el error de generalización **

Paréntesis
Conceptos de estadística asociados:
- Sesgo (bias): ¿qué tan lejos estamos de los parámetros reales? mucho sesgo == muy lejos de los parámetros, nuestro algoritmo/parámetros está incorrectamente sesgado (hacia un modelo incorrecto)
- Varianza (variance): ¿qué tanto cambiaría mi estimación de los parámetros si cambio los datos de entrenamiento? mucha varianza == puedo no estar aprendiendo los parámetros correctos por un sobre ajuste a los datos ==> relacionado al error de generalización

Lo que queremos es estimación de los parámetros que sea poco sesgada (se acerque a la real) y tenga poca varianza (no dependa tanto de los datos que usamos para entrenar).

Nota práctica:
- Hasta ahora hemos usado la función de pérdida como error. Esto tiene sentido para que el aprendizaje funcione bien y lo hicimos pensando en tener una motivación estadística (estimador de máxima verosimilitud + cross entropy).
- En la prácitca para evaluar qué tan bien/mal lo hace un modelo, puede que esa función no signifique mucho. En ese caso debemos elegir la función de error de nuestra aplicación. Le llamaremos **métrica**
- Esta elección es muy importante en la práctica y debemos hacerla temprano en el ciclo de construcción de una red neuronal. Cambiarla en el camino puede significar pérdida de tiempo valioso!
- Debemos intentar siempre que sea sólo una (para guiar el progreso)
- Típicamente consideramos el porcentaje de acierto/falla: cantidad de ejemplos clasificados correcta/incorrectamente dividido por la cantidad total de ejemplos
- Adevertencia: es razonable elegir esta métrica cuando nuestro problema de clasificación está bien balanceado (aproximadamente la misma cantidad de ejemplos/datos-reales para cada posible clase), si no, esta puede carecer de sentido! Ir a curso de ML o DM para discutir detalles adicionales
- Otras métricas simples:

- cantidad de aciertos dentro de las top k clases: cantidad de veces que la clase correcta se encontraba dentro de las top k con mayor probabilidad de mi modelo.
- precisión (precision en inglés) por clase: para cada clase C considerar el porcentaje de ejemplos clasificados como C que efectivamente eran de la clase C
- cobertura (recall) por clase: para cada clase C considerar la cantidad de ejemplos de la clase C que fueron efectivamente clasificados como C por el modelo
- F1: pondera precisión y cobertura
- promedio de precisión/cobertura para todas las clases: promedio ponderado de la métrica para las clases, o promedio plano (macro average) de los valores de la métrica para las clases.

==========

¿Cómo aseguramos (o intentamos asegurar) buena generalización?
- consiguiéndonos un conjunto de datos que sólo usaremos para estimar el error de generalización (NO LOS USAREMOS PARA ENTRENAR DE NINGUNA FORMA).

==========

Train/Test set y Capacidad

- cuando tenemos un conjunto de datos (etiquetados) para construir un modelo, lo que hacemos es dejar cierto porcentaje de datos de lado para estimar el error de generalización. A este conjunto le llamamos "test". Todo el resto será nuestro conjunto para entrenar o "train".

- train error: error cometido en el conjunto de entrenamiento
- test error: error cometido en el conjunot de test ==> estimación del error de generalización

- Importante: intentar asegurar que los dos conjuntos (train y test) vienen aproximadamente de la misma distribución de probabilidad!!!!

- Note que: si eligiéramos los parámetros al azar, los valores esperados de error de train y error de test deberían ser iguales

- Pero los parámetros se eligen usando el train set ==> el entrenamiento disminuirá el train error (lo que está muy bien! es lo que queremos) ==> el error de train será menor o igual al error de test

Nuestro objetivo general será (siempre):

1. disminuir el train error tanto como podamos
2. asegurar que la diferencia entre el train error y test error sea tan chica como podamos

- si 1 es grande ==> tenemos **underfitting** ==> nuestra red no es suficientemente expresiva para representar la distribución de probabilidad que genera los datos ==> baja capacidad (cantidad de funciones que puede aprender)

- si 2 es grande ==> tenemos **overfitting** ==> nuestra red no generaliza a nuevos datos ==> no está aprendiendo la real distribución de probabilidad aunque esta vez no es por falta de capacidad (probablemente por lo contrario!)

[DIBUJO DE TRAIN/GENERALIZATION ERROR SEGUN CAPACIDAD]

OJO: overfitting no significa que el train error sea pequeño, si no que el generalization error es grade.

- Un modelo de ML (en general) funcionará bien cuando su capacidad sea la correcta para el problema en cuestión.

==========

Hiperparámetros y Dev (Val) set

Dos puntos importantes:

a) ¿Cómo podemos mejorar el error de generalización si no podemos usar el test set mientras entrenamos?
b) Para disminuir el train error generalmente debemos modificar la capacidad del modelo (aumentar la capacidad), ¿cómo elegimos esto evitando overfitting?

- Hiperparámetro: cualquier medida/parámetro que gobierne la capacidad del modelo o de la forma de aprender el modelo. En nuestro caso de redes neuronales:
  - cantidad de capas escondidas
  - tamaño de cada capa
  - funciones de activación
  - algoritmo de optimización (descenso de gradiento u otro)
  - tasas de aprendizaje (en el caso del descenso de gradiente)
  - tamaño del paquete que usamos para cada paso del desceso del gradiente
  - cantidad de pasos de actualización por descenso de gradiente
  - métodos y parámetros de regularización (pronto lo veremos!)
  - incluso cantidad de ejemplos usados en el entrenamiento (aunque este no es un parámetro trivial de modificar!)

- Para encontrar buenos hiperparámetros y a la vez buscar un modelo con bajo error de generalización, se usa otro conjunto de datos, que llamaremos Development set o Validation set (dev set)

- ¿Por qué no buscamos los hiperparámetros en el train set? ==> esto nos llevaría a encontrar hiperparámetros que siempre favorecieran una capacidad mayor para disminuir el train error ==> no mejoraría el error de generalización

- Típicamente el dev set es un porcentaje del train set y lo usamos para seleccionar la capacidad correcta que baje train y generalization error

==========

En resumen:

* Train set: se usa para adaptar los parámetros de la red y disminuir el train error
* Dev set: se usa para adaptar los hiperparámetros de la red (capacidad, regularización) y estimar el generalization error durante el entrenaiento (evitar overfitting)
* Test set: se usa para estimar el generalization error después de que el entrenamiento ya acabó

Buenas prácticas para elegir Train/Dev/Test:

- clásicamente (no muchos datos <= 10.000):
  60%/20%/20% o 70%/15%/15%
- cuándo los datos son muchos (>= 1.000.000):
  90%/5%/5% o incluso 98%/1%/1%
- a veces las particiones vienen hechas en el data set (investigación o competencias)
- a veces puedo decidir desde antes el tamaño del test set, por ejemplo 10.000, y puedo usar todo lo que quiera como train (ir a buscar más datos)

IMPORTANTE: asegurar siempre que (al menos) el dev set y el test set vienen de la misma distribución de probabilidad. A veces se puede vivir con un train set que no tiene exactamente la misma distribución de probabilidad del test set (si es que eso me permite aumentar el conjunto de entrenamiento).

Receta básica para resolver un problema con DL-ML

++++++++++
P.0) Entrenar la red
P.1) Si hay mucho train error => underfitting (alto bias):
  - aumentar capacidad
  - volver a (P.0)
P.2) Si hay bajo train pero gran diferencia entre train y dev error => overfitting (alta varianza):
  - cambiar hiperparámetros (principalmente los que no aumentan capacidad)
  - volver a (P.0)
P.3) Si no, listo! :-)
++++++++++

- Cómo puedo aumentar la capacidad?

* más capas escondidas
* más nueronas por capas
* cambiar la arquitectura
* entrenar por más iteraciones (<- le doy más tiempo para buscar)

- Qué cambio sin aumentar la capacidad?
* buscar más datos para entrenar
* regularizar más (ya lo veremos!)
* cambiar la arquitectura (por una más simple)

- Cómo saber qué tan aceptable es el train error? Cómo saber cuánto puedo mejorar
* Comparar con el error cometido por personas (Human Level Performance)
* Human level performance se usa como aproximación para el error mínimo

==========

Algunos tips de desarrollo:

- Las anteriores estrategias tienen sentido solo si el modelo no tiene bugs
- Un par de tips para encontrar bugs:
0) Chequear que todas las dimensiones de cada capa están correctas
1) Si las derivadas se calcularon a mano: programar chequeo numérico de gradiente
2) Entrenar la red con (muy) pocos ejemplos: cualquier modelo sin bugs debiera ser capaz de aprenderse los ejemplos con 0% de error!