Universidad Técnica Federico Santa María
Departamento de Informática
Introducción a redes neuronales artificiales
Primer Semestre 2017

Certamen I-2017

Profesor: Ricardo Ñanculef

Rolando Casanueva	201204505-3	rolando.casanueva.12@sansano.usm.cl

El enunciado puede ser encontrado en el GitHub rCasanueva.

Desarrollo

1. **Explique el concepto *overfitting*, cuando es común su aparición y mencione dos técnicas para mitigar este problema.**

   El overfitting, tal como se traduce (*sobre ajuste*), es un problema de entrenamiento donde un algoritmo es entrenado más de la cuenta con unos datos conocidos. Esto conlleva a que la red conoce muy bien como resolver los ejemplos dado que esta basada en ellos pero al momento de predecir, no da en los resultados. Como se entiende, una red debe ser capaz de predecir nuevos resultados mediante el conocimiento de un conjunto de entrenamientos, pero sobre entrenar dicha red puede generar que el algoritmo analice características muy especificas.

   Este fenómeno suele suceder en gran parte cuando se tienen entrenamientos muy extensos o cuando los datos de entrenamientos son *raros*, entiendase por raro, casos especiales apartados de la función objetivo. Un ejemplo puede ser un modelo que otorgue resultados en base a una fecha histórica, este modelo podrá dar solución a los eventuales datos de entrenamiento, sin embargo no podrá rendir en escenarios de predicción porque una fecha *nueva* no habrá sido nunca visto por el modelo por lo que no sabrá como actuar o analizarla.

   Existen varias técnicas para mitigar el sobre-ajuste, de las cuales las que tenemos más internalizadas y son más frecuentes consisten en:

   - Data Split
     Una de las técnicas más utilizadas, aún cuando no se esté esperando un *overfitting* es la división del dataset. En conjunto de datos donde los ejemplos son *raros* y escasos, la mejor opción es utilizar una semilla aleatorea para subdividir el conjunto de datos en 3 categorias: **Conjunto de entrenamiento**, **Conjunto de validación** y **Conjunto de testing**. De esta forma, sin importar la extrañeza del conjunto de datos, habrá un entrenamiento con elementos que difieran de la validación y del testeo.
   - Regularizadores
     Basicamente un regularizador es una técnica estadística que ajusta la función objetivo para actuar como optimizador. Los regularizadores mejoran el rendimiento dado que evita el sobre-ajuste al generar un conducto para los datos entre capas. Como se define en el capítulo 7 *"any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error"*, es decir, toda modificación que le hagamos a la función objetivo con intensión de reducir los errores de generalización que en parte generan este sobre-ajuste.

2. **Explique como eligiría la cantidad de neuronas y la taza de aprendizaje de un MLP estandar. Considere que tiene un conjunto de n datos y la tasa de aprendizaje es fija. ¿Cambia la decisión si se considara un conjunto de n datos muy grande en comparativa con un n muy pequeño?**

Cuando hablamos de la cantidad de neuronas en el perceptron, hablamos de la cantidad de nodos que tendrá la capa. Existe una gran variedad de literatura que habla de metodologías y eurísticas para generar un número verídico, sin embargo es claro que hasta el momento no existe una regla/teoremo/axioma respecto a esto. Particularmente, me gusta pensar que una red neuronal más alla de converger en un resultado con exactitud, me agrada imaginarlo como que cada capa de la red acota la información o al menos la mantiene, es por esto que una capa escondida, para mi, debe ser algo así como un escalafon entre la capa anterior y la siguiente.

$$N_h = (N_{input} + N_{output}) * \frac{2}{3} \tag{1}$$

Con respecto a la taza de aprendizaje, dado que se debe considerar fija, tendremos que ser muy rigurosos al respecto. Una taza muy grande nos puede llevar a un overfitting mientras que una taza muy pequeña puede generar el caso contrario. Generalmente considero que las tasas del orden de magnitud de 10^{-2} son una buena elección. Todas estas decisiones están evidentemente sujetas a la comprobación empírica, por lo que en su eventualidad podría decidirse cambiar estos parámentros.

Ahora bien, considerando la pregunta final, sobre el conjunto de datos. Esto no afecta en mayor medida un rendimiento de la red en su funcionalidad, por lo que el número de neuronas se mantendría intacto independiente de la cantidad de ejemplos, mas no así la taza de aprendizaje. Como mencioné, una taza del orden de magnitud de 10^{-2} es un valor estable y seguro, pero si sabemos que existe un conjunto muy pequeño de datos, yo subiría la taza para converger más rápido y para el caso contrario haría lo mismo, aunque siempre con cautela y no disminuyendo más de 2 ordenes de magnitud.

3. **Dos motivos del entrenar repetidamente una red manteniendo parametros, genera un resultado diferente**

Si bien una red tiene parametros que pueden generar la sensación de equidad tal como *batch size, número de capas, número de neuronas, funciones de pérdida, activación ... etc*, existen unos valores que pueden ser los mismo pero que al presentarlos de una manera distinta generarán un resultado completamente diferente.

- Conjunto de Entrenamiento
  Algo que no se tiene en cuenta siempre es que el conjunto de entrenamiento se presenta de una manera aleatorea, por este motivo es que el entrenamiento nunca tomará el mismo curso aún cuando se repita el proceso. La unica forma de mantener esto invariable es generar un entorno aleatoreo mediante una semilla para luego utilizar el mismo entorno, sin embargo esto no suele suceder, y basta que 1 elemento se presente en otro orden para que el resultado final varíe. Recordar que el entrenamiento conlleva una modificación de pesos los cuales, si no se presentan los datos en el mismo orden, variará la solución.
- Inicialización
  Otro tema muy importante que va de la mano con el punto anterior, son los pesos iniciales de la red. Como se dijo en el otro punto, estos tipos de parametros son aleatoreos y como tales al repetir un entrenamiento, la inicialización generará una nueva configuración.

Ya sea considerando uno o ambos, basta para entender que estos rasgos aleatoreos son los que producen una variación a la hora de repetir un entrenamiento.

4. **¿Es correcto utilizar una función de perdida *binary cross entropy* en un problema de clasificación con Redes Feed-Forward?**

En particular, las redes feed-forward tienen distintos comportamientos según el tipo de clasificación que se desea hacer. Si la pregunta fuese un verdadero o falso absoluto, la respuesta sería que si es correcto. Si separamos en 2 tipos de clasificaciones podemos explayarnos un poco al respecto, considerando un problema *2-way classification* lo más correcto es enfrentarlo con una pérdida *cross entropy*, las cuales pueden ser **binary cross**

entropy o **categorical cross entropy**, donde la primera tiende a ser la más utilizada en conjunto con una función de activación sigmoidal. Para problemas de multiple clasificación binaria, se usan individualmente las herramientas recien descritas. Por otra parte en un problema *multi-way classification* se utiliza una activación no lineal, comúnmente *softmax*, y como tal va acompañada de una función de pérdidad distinta *(ej. negative log-likelihood loss)*.

5. **Problema del 'desvanecimiento'. ¿Por qué en redes recurrentes? ¿Depende de la función de activación?**

   El desvanecimiento del gradiente decendente es un problema que supone una disminución de relevancia o disminución de aporte a través de las capas. En redes profundas la caracteristicas fundamentales de las primeras capas pueden no tener relevancia en las últimas lo que genera que a medida que el gradiente atraviesa la red por las capas con backpropagation su aporte converge a nulo. Las redes recurrentes tienen asociado este problema debido a su constante paso por las capas, estas recursiones o bucles generan que la convergencia del gradiente sea más intensa y finalmente su aporte se *desvanece*.

   No, no depende de la función de activación. El *gradiente descendente* funciona como un optimizador y como tal está ligado a la función de pérdida, dado que este actua al momento de realizar backpropagation y no en el momento de generar un forwardpropagation. Un ejemplo es la tarea 1, donde para el ejercicio dos, no importaba si utilizabamos una función cuadrática o sigmoidal de todas formas generaríamos este optimizador *SGD*.

6. **¿Por qué se afirma que Dropout actúa como Bagging? ¿Por qué Dropout resulta menos costoso?**

   En términos generales, el Bagging es un proceso de entrenamiento parcializado, es decir no actua realmente sobre todo el conjunto sino que divide el conjunto completo en subconjuntos y procede a entrenarlos como modelos independientes. Al funcionar de esta manera no todos estos 'submodelos' presentan todos los datos. El dropout simula el bagging ocultando ciertas caracteristicas generando una especie de entrenamiento compartido, aún cuando se entrena al conjunto completo. Sin embargo, como el dropout no genera realmente subconjuntos, y en consecuencia no genera submodelos, resulta mucho más barato a la hora de entrenar porque solo es 1 modelo el que debe ser entrenado.

7. **Etiquetar imagenes de caras humanas, K=10 emociones, Red Convolucional de capas L ¿3, dataset de 32 imagenes para cada emoción.**

   a. **Estrategia de expansion de imagenes**
   Afortunadamente hemos conocidos herramientas para trabajar imagenes a lo largo del curso, dado que el dataset es bastante pequeño yo aplicaría dos criterios de expansión.

   - Rotación
     Algo impresindible es notar que las imagenes serán vistas como matrices y el solo hecho de transformar estas matrices mediantes traslaciones y rotaciones, aun cuando en el despliegue parezca la misma imagen, para la red será una completamente diferente.
   - Filtros
     Otra técnica que conocemos es aplicación de filtros a las imagenes, de tal forma que sus tonalidades, su enfoque y otras caracteristicas sean modificadas, pero que a su vez siga siendo reconocible el patrón que buscamos.

   Así utilizando estas dos tecnicas se puede llegar a expandir el dataset de una manera bastante veridica.

   b. **¿Tiene sentido hacer pre-entrenamiento no supervisado?**
   En general un entrenamiento no supervisado puede ayudar a la red a conocer caracteristicas de las imagenes y generar un suerte de conciencia sobre el conjunto de datos. Particularmente todos los articulos que he leido sobre ANN señalan que esta práctica se ha dejado de lado porque puede generar que la red busque caracteristicas que no sean relevantes al no tener una supervisización en la adjudicación de pesos.

Ahora bien, la pregunta es si tiene sentido, si asumimos que el dataset es tan pequeño, y que no hemos realizado una expansión, si podría tener sentido.

c. **¿Tiene sentido hacer pre-entrenamiento con imagenes distintas?**
A diferencia de la suposicion anterior, donde las imagenes tiene una relación con lo que luego se utilizará para el entrenamiento, utilizar imagenes distintas como autos, aviones y otros, no generará ningún beneficio al objetivo que es categorizar imágenes según emociones. Por ende, no tiene sentido.

8. -

9. **Diferencia entre algoritmo CD *Contrastive Divergence* y PCD *Persistent CD* para entrenar RBM's. Explicar que es CD_k**

Para poder responder esta pregunta presentaré las ventajas y desventajas de cada algoritmo para presentar cuando es más beneficioso cada uso. Sin profundizar mucho al momento de realizar una aproximación **CD** explora las regiones cercanas a los ejemplos de entrenamiento, mientras que **PCD** explora en profundidad el dominio de entrada.

PCD es mucho mejor a la hora de representar el *log-likelihood* de la información entregada por lo que es mejor a la hora de realizar muestreos desde el modelo, si bien CD converge de una manera más acelerada es porque tiene una variación de parametros mucho menor, por lo mencionado anteriormente.

CD es mejor que PCD a la hora de extraer razgos y características. Se dice que cuando se ha minimizado el error de reconstrucción, CD entrega casi puros *updates* de valor 0, por lo que el aprendizaje concluye rápidamente, mientras que PCD al seguir aprendiendo, se desliga de mucha información especifica que al momento de solicitar una dependencia por sobre el input con respecto a una caracteristica, PCD tiene problemas para realizar esta relación.

En general, PCD resulta ser mucho más beneficioso aunque si se desease crear una red muy profunda, sería aconsejable utilizar CD para niveles inferiores (donde se utiliza mucho la extracción de caracteristicas) y PCD para niveles superiores.

CD_k: Como CD no espera a que las cadenas converjan, las muestras se obtienen tras *k-pasos* del *Gibbs Sampling*.

10. **Red convolucional alimentada con RGB 32x32. Ilustrar las transformaciones, determinar número de parametros. Sin implementación de padding.**

La red presenta una estructura donde los parametros de la imagen se ven reducidos a la mitad en las capas de pooling. El paso de la estructura está representado en la siguiente tabla. *Nota: Se comprobó con Keras.*

```
model = Sequential()
model.add(Convolution2D(16, (4, 4), input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Convolution2D(32, (4, 4)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dense(256))
model.add(Dense(10))
model.summary()
```

Capa	Output	Parametros
CONV	32x32x16	784
POOLING	16x16x16	0
CONV	16x16x32	8224
POOLING	8x8x32	0
HIDDEN	8x8x256	8448
OUT	8x8x10	2570

El número total de parametros es *20,026*

11. **PostGrado**

12. **¿Que tipo de arquitectura usar para una red de reconocimiento de lenguaje de señas a través de video?**

    Como se nos dice que los datos serán entregados en secuencias de frames, básicamente el problema se reduce a clasificar una imagen dependiendo de la posición y caracteristicas de las manos. Si embargo esta tarea no es fácil dadas las complejidades que conllevan las personas reales, digase, no todos presentan el mismo gesto de la misma forma ni en una misma posición. En el fondo, esta problemática ya se nos ha presentado pero en un formato distinto, como reconocer números o clasificar fotos por su contenido.

    Analizando los datos entregados, solo se puede pensar en una categorización de multiples clases. Gracias a la tarea 2 podemos decir que no debiese ser una red con una gran cantidad de capas. Como los datos de input son complejos se requiere o prefiere una activación sencilla de calcular y a su vez una activación para categorizaciones multiples. La funcion de perdida tambien debe ser del estilo categórico.

    Finalmente la arquitectura que se concluye es la siguiente, una red convolucional, la cual posee dos capas convolucionales con activación $ReLU$ y cada una con un pooling de filtros, dos capas escondidas, una con activación $ReLU$ mientras que la otra con *softmax* cumpliendo el rol de capa de salida. Finalizar la arquitectura con una función de pérdida *categorical crossentropy*.

13. **PostGrado**

14. **PostGrado**

15. -

16. -

17. **LSTM usando Dropout, ¿Debe ser la mascará compartida/fija en el tiempo?**

    Como LSTM se utiliza en redes recurrentes, debemos notar que genera bucles de información donde el dropout es utilizado para evitar el *overfitting*, sin embargo mantener fija la máscara reducirá la efectividad del dropout con el transcurso del entrenamiento, esto se debe a que generaría una conexión establecida entre neuronas la cual sería invariante y de realizar una secuencia repetida el dropout al disminuir los razgos evitará un correcto entrenamiento.