Clase 19: Redes neuronales (parte 3)

MA5204 Aprendizaje de Máquinas

Felipe Tobar

Department of Mathematical Engineering &
Center for Mathematical Modelling
Universidad de Chile

8 de junio de 2021

UNIVERSIDAD
DE CHILE

Algoritmos de Optimización - Minibatch

Los algoritmos de optimización para aprendizaje de máquinas típicamente actualizan los parámetros usando un valor esperado del costo, obtenido a través de la esperanza muestral de la función de costos:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x},y\sim\hat{p}_{\text{data}}} \nabla_{\boldsymbol{\theta}} \log p_{\text{modelo}}(y|\boldsymbol{x}).$$

Los métodos que utilizan todo el conjunto de datos de entrenamiento en cada iteración se conocen como **métodos de batch** o **determinísticos** y tienden a quedar atrapados en óptimos locales.

Los algoritmos que usan un solo dato a la vez se conocen como **métodos estocásticos** y son tremendamente ineficientes para una cantidad grande de datos.

Por lo tanto, la mayoría de los algoritmos usados pertenecen a una categoría intermedia, estos son los **métodos de minibatch o minibatch estocástico**, los cuales usan un subconjunto de tamaño reducido y calculan un promedio de gradientes para obtener el valor esperado.

Algoritmos de Optimización - Variantes del SGD

Promedio móvil exponencial: Esta modificación consiste en promediar el gradiente con su valor en la iteración anterior para incorporar *momentum* en la secuencia de actualizaciones de parámetros:

$$v_t = (1-\beta)\left(\frac{\partial J}{\partial \theta_t}\right) + \beta v_{t-1}; \quad v_0 = 0,$$

donde la actualización de parámetros viene dada por

$$\theta_{t+1} = \theta_t - \lambda v_t.$$

AdaGrad: Esta modificación es sobre la taza de aprendizaje con la finalidad de avanzar el descenso de gradiente a tazas independiente para cada parámetro. Definimos

$$G_t = \sum_{\tau=1}^{t} \frac{\partial J}{\partial \theta_t}^{\top} \frac{\partial J}{\partial \theta_t}$$

Con actualizaciones dadas por

$$\theta_{t+1} = \theta_t - \alpha \cdot \operatorname{diag}(G_t)^{-1} \frac{\partial J}{\partial \theta_t}.$$

Algoritmos de Optimización - Variantes del SGD

RMSprop: Esta modificación intenta ser una mejora a la anterior, Adagrad, y lo que hace es introducir promedios moviles exponenciales sobre el learning rate de la siguiente forma

$$v_t = \beta v_{t-1} + (1-\beta)\frac{\partial J}{\partial \theta_t} * \frac{\partial J}{\partial \theta_t}; \quad v_0 = 0$$

y actualiza

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} * \frac{\partial J}{\partial \theta_t}$$

Adam: Finalmente esta modificación es simplemente una combinación de las ideas de taza de aprendizaje adaptativo RMSprop y momentum (calculados para la media y la varianza del gradiente). Actualmente es una de las más utilizadas.

Observación

Hasta ahora hemos visto varias modificaciones del descenso de gradiente estocástico pero cabe destacar que no existen grandes resultados teóricos que las sustenten, es decir, corresponden a heurísticas.

Redes Neuronales Convolucionales

Las **redes neuronales convolucionales** (o **CNNs**) están diseñadas para procesar datos que tienen estructura que se repite espacialmente. Por ejemplo, una serie de tiempo puede tener patrones repetitivos en el tiempo o una imagen puede tener patrones que se repiten en distintas regiones (como bordes o círculos). Estas redes operan mediante convoluciones, definidas como:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da.$$

En el contexto de CNNs, el primer argumento a convolucionar, x, es la **entrada**, y el segundo argumento, w, se conoce como el **kernel de convolución**.Usualmente, el **kernel** es un arreglo muldimensional de parámetros entrenables de menor dimensionalidad que la entrada y que se desplaza a través de este según un valor denotado como *stride*, para extraer *características* de alto nivel (como bordes por ejemplo).

Redes Neuronales Convolucionales

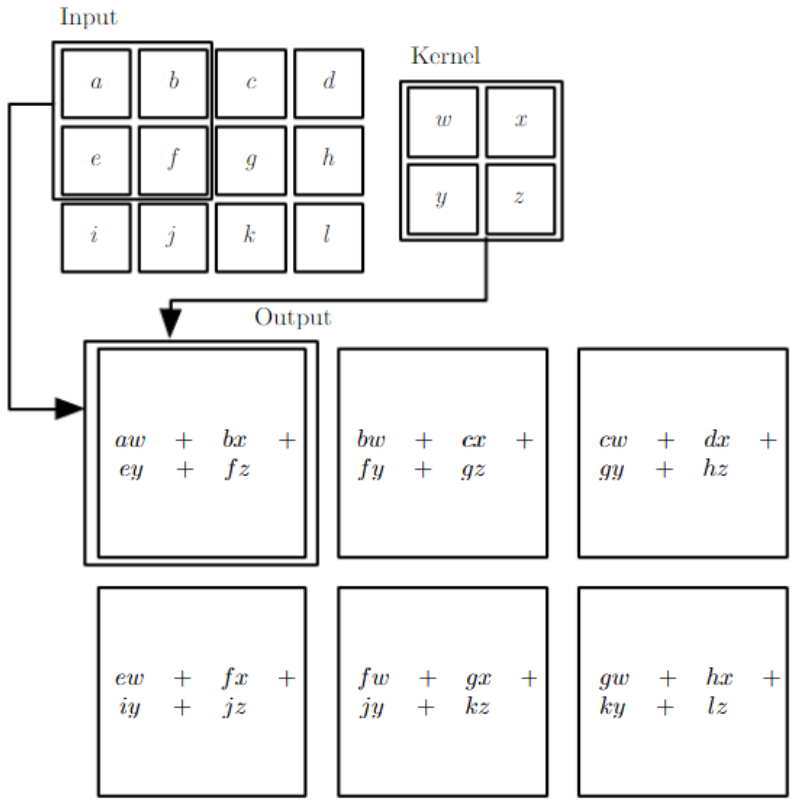


Fig.. Ejemplo de una convolución 2-D, el valor de *stride* es (1,1) (Goodfellow y cols., 2016)

Una vez extraídas las características mediante la utilización de variados kernel según se requiera, éstas son reubicadas y usadas como entrada para una *feedforward neural network* que se encargará de entregar la salida correspondiente.

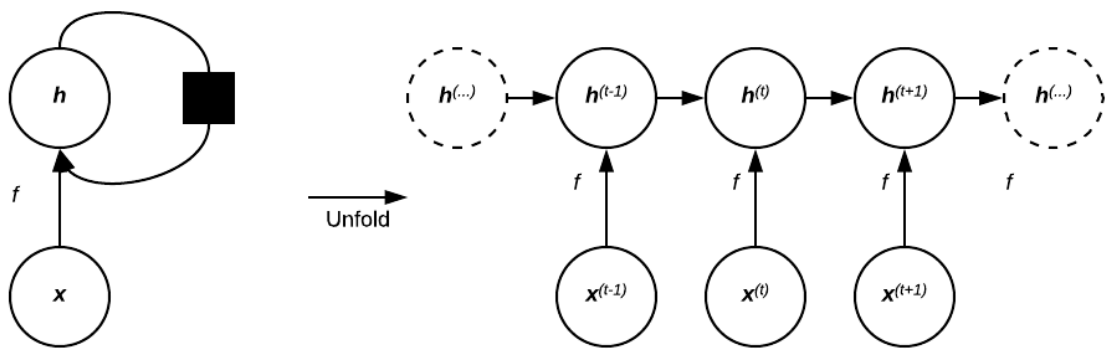
Redes Neuronales Recurrentes

Las **redes neuronales recurrentes** (o **RNNs**) son una familia de modelos especializados en procesamiento de datos secuenciales, $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(\tau)}$. Las RNNs también comparten parámetros, pero en una forma muy distinta que las CNNs. En una RNN, cada entrada en una etapa es una función de la salida de la etapa anterior.

Se denota por $\boldsymbol{h}^{(t)}$ al estado de un sistema dinámico que involucra una recurrencia conducido por una entrada externa $\boldsymbol{x}^{(t)}$:

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}; \boldsymbol{x}^{(t)}, \boldsymbol{\theta}).$$

Fig.. Ejemplo de una red recurrente sin salida (Goodfellow y cols., 2016)

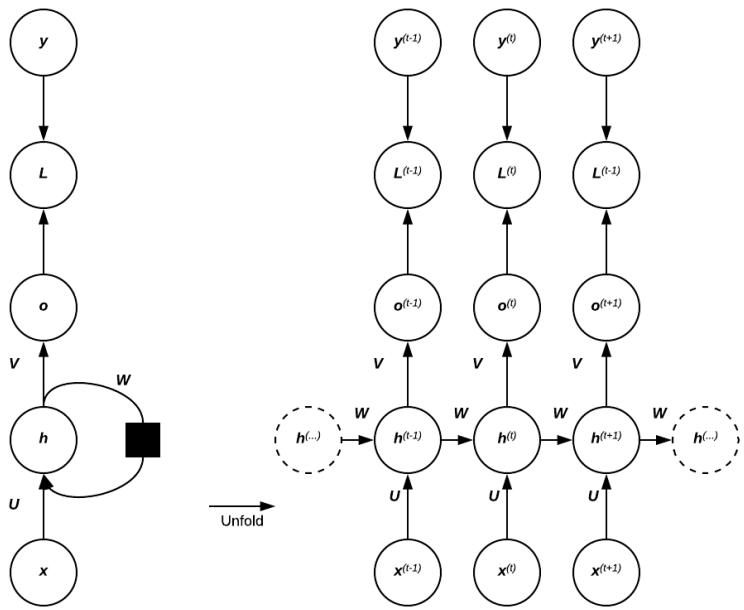


Redes Neuronales Recurrentes

Existen varios tipos de RNNs que se han diseñado para distintos fines.

- Redes recurrentes que producen una salida en cada instante de tiempo y tienen conexiones entre todas las unidades escondidas
- Redes recurrentes que producen una salida en cada instante de tiempo y tienen conexiones entre la salida a la unidad escondida del siguiente instante
- Redes recurrentes con conexiones entre las unidad escondidas, que procesan una secuencia entera antes de producir la salida

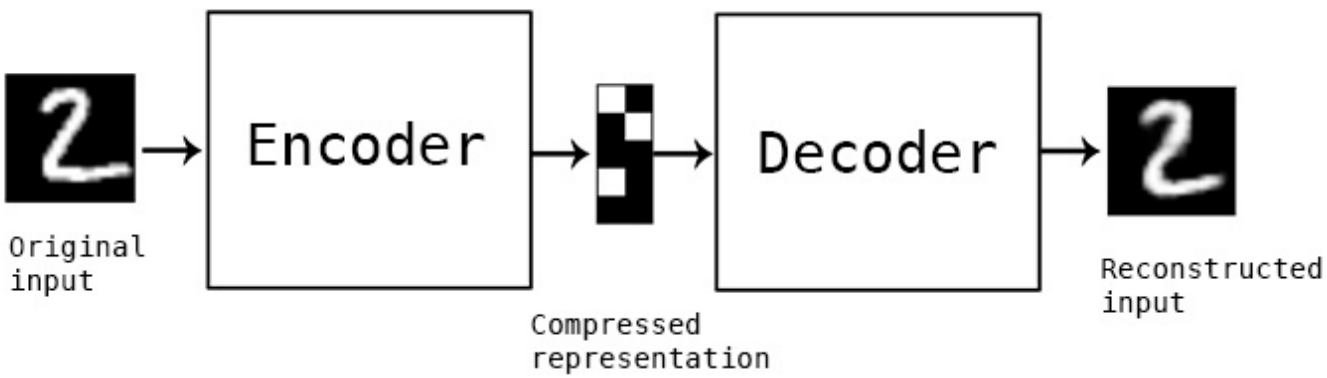
Fig.. Red recurrente para el primer ejemplo (Goodfellow y cols., 2016)



Autoencoders

Un **autoencoder** es una red neuronal que busca replicar la entrada hacia la salida, es decir, busca que la información que entra a la red sea lo más parecida posible a la de salida, para lo cual cuenta con una capa interna $\boldsymbol{h} = f(\boldsymbol{x})$ (**encoder**) con menor dimensión que la entrada y que lo codifica (genera una representación de baja dimensión de la entrada) y una función que produce la reconstrucción $\boldsymbol{r} = g(\boldsymbol{h})$, el **decoder**.

Un autoencoder buscará aprender los *encoder* y *decoder* tales que $g(f(\boldsymbol{x})) = \boldsymbol{x}$ para todo $\boldsymbol{x}$. Como el modelo está forzado a aprender los atributos más importantes para que pueda efectivamente reproducir la entrada en su salida, este aprenderá en general propiedades útiles de los datos de entrenamiento. Los *autoencoders* modernos modelan mappings estocásticos $p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$ y $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$, en vez de funciones determinísticas.



Redes Generativas Adversariales

Una **red generativa adversarial** (o **GAN**) se basa en un escenario de teoría de juegos, en donde una **red generadora** debe competir con un adversario. La red generadora produce muestras $\boldsymbol{x} = g(\boldsymbol{z}; \boldsymbol{\theta}^{(g)})$, mientras que una **red discriminadora** trata de distinguir entre muestras obtenidas de los datos de entrenamiento y muestras generadas por la red generadora. El discriminador retorna una probabilidad, $d(\boldsymbol{x}; \boldsymbol{\theta}^{(d)})$, indicando la probabilidad de que $\boldsymbol{x}$ sea un dato real y no uno simulado.

Para formular el aprendizaje, se describe un juego de suma cero en donde una función $v(\boldsymbol{\theta}^{(g)}, \boldsymbol{\theta}^{(d)})$ determina el pago del discriminador, y el generador recibe $-v(\boldsymbol{\theta}^{(g)}, \boldsymbol{\theta}^{(d)})$ como pago. Así, durante el entrenamiento cada jugador intenta maximizar su propio pago, para que en convergencia se tenga

$$g^* = \text{argmin}_g \text{max}_d v(g, d).$$

Esto motiva a que el discriminador aprenda a clasificar correctamente entre muestras reales y falsas y, simulatáneamente, el generador intenta engañar al clasificador para que crea que las muestras generadas son reales. En convergencia, las muestras del generador son indistinguibles de los datos reales. Una motivación del uso de GANs es que cuando $\text{max}_d v(g, d)$ es convexa en $\boldsymbol{\theta}^{(g)}$, el procedimiento asegura la convergencia.

Redes Generativas Adversariales

Fig.. Imágenes generadas por una GAN entrenada con el set de datos LSUN. (Izquierda) Imágenes de dormitorios generadas por el modelo DCGAN (imagen de Radford et al., 2015). (Derecha) Imágenes de iglesias generadas por el modelo LAPGAN (imagen de Denton et al., 2015)

Clase 19: Redes neuronales (parte 3)

MA5204 Aprendizaje de Máquinas

Felipe Tobar

Department of Mathematical Engineering &
Center for Mathematical Modelling
Universidad de Chile

8 de junio de 2021

UNIVERSIDAD
DE CHILE

Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep learning*. MIT Press. (`http://www.deeplearningbook.org`)