Optimización Continua y Aplicaciones (OCA)

Pablo Rodríguez-Bocca
Departamento de IO, Instituto de Computación
prboccab@fing.edu.uy

26 de agosto de 2023

Optimización y Aprendizaje Automático

Pablo Rodríguez-Bocca
Departamento de IO, Instituto de Computación
prboccab@fing.edu.uy

26 de agosto de 2023

Aprendizaje automático supervisado

Aprendizaje automático supervisado

- Aprendizaje automático supervisado
  - conocemos N muestras de datos de entrada y salida: (x, y)
  - queremos aprender $y = f(\mathrm{x})$
  - $\mathrm{x} \in \mathbb{R}^p$ son los atributos
  - tipos de problemas de aprendizaje
    - regresión: $y \in \mathbb{R}$
    - clasificación: $y \in \mathbb{N}$ (llamada clase, etiqueta)
- Métodos de aprendizaje
  - limitan las opciones de $f()$ a un modelo paramétrico: $\hat{y} = f(\mathrm{x}; \boldsymbol{\theta})$
  - el problema es encontrar los mejores parámetros θ, que reduzcan el error del modelo

    $\Rightarrow$ debe definirse una función de pérdida (error) entre el resultado del modelo y la realidad

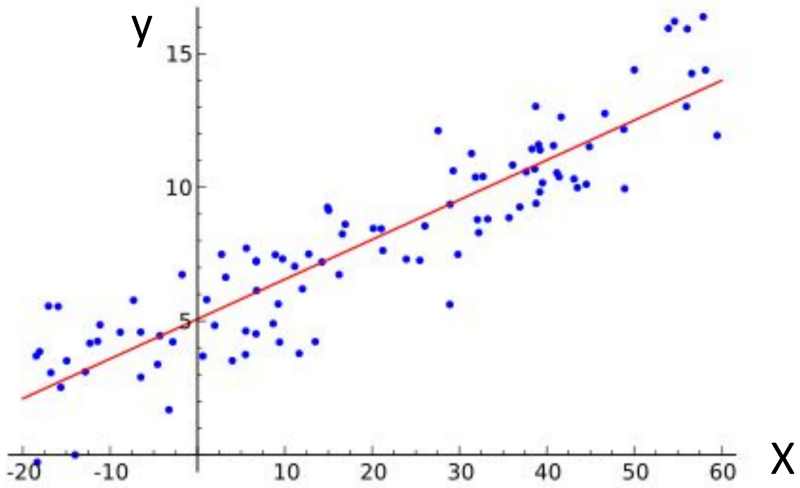
$$\mathcal{L}(\hat{y}, y)$$

el aprendizaje es un problema de optimización

$$\boldsymbol{\theta}^* \leftarrow \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}(f(\mathrm{x}; \boldsymbol{\theta}), y)$$

  - hay muchos métodos, veremos tres para fijar ideas . . .

Regresión lineal



- buscamos la mejor recta que explique los datos

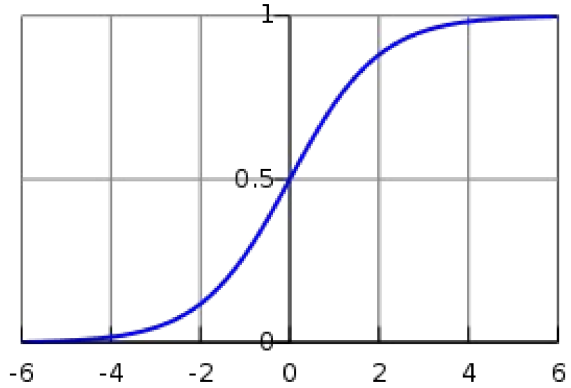
- el modelo es $\hat{y} = f(\mathrm{x}; \boldsymbol{\theta}) = \theta_0 + \sum_{j=1}^{p} \mathrm{x}_j \theta_j$
- y la función de pérdida es: $\mathcal{L}(f(\mathrm{x}; \boldsymbol{\theta}), y) = \sum_{i=1}^{N} (y_i - f(\mathrm{x}_i; \boldsymbol{\theta}))^2$
- problema de optimización con solución óptima conocida, pero su cálculo es costoso para muchos datos, y se usan métodos iterativos (ver más adelante...)

Regresión logística

- clasificador binario (0/1), su salida es la probabilidad de estar en la clase 1

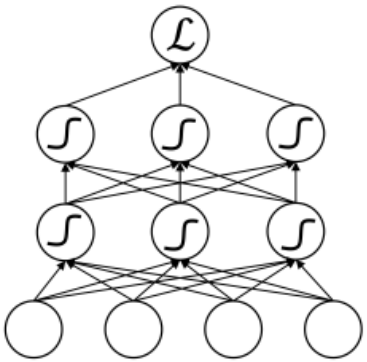
  $\Rightarrow$ el modelo es $p(y = 1|\mathrm{x} = \mathrm{x}) = f(\mathrm{x}; \boldsymbol{\theta}) = \frac{1}{(1+e^{-\mathrm{x}\boldsymbol{\theta}})}$

- la función logística mapea $(-\infty, \infty)$ a $[0, 1]$



- la función de pérdida es la exactitud (*accuracy*) total
  - para cada entrada (x_i, y_i), la exactitud es $a_i = y_i f(\mathrm{x}_i; \boldsymbol{\theta}) + (1 - y_i)(1 - f(\mathrm{x}_i; \boldsymbol{\theta}))$
  - log accuracy total: $a = \sum_{i=1}^{N} log(a_i)$
- Puede entrenarse usando descenso por gradiente

Redes Neuronales - Aprendizaje Profundo (DL)



- El modelo se define para cada muestra

$$a_L(\mathrm{x}_i; \mathrm{w}_{1\ldots L}) = h_L(h_{L-1}(\ldots h_1(\mathrm{x}_i, \mathrm{w}_1)\ldots, \mathrm{w}_{L-1}), \mathrm{w}_L)$$

  - los parámetros son los pesos w

- La función de pérdida es una suma en la cantidad de muestras, siendo el aprendizaje el problema de minimizar el error empírico

$$\mathrm{w}^* \longleftarrow \underset{\mathrm{w}}{\operatorname{argmin}} \sum_{i=1}^{N} \mathcal{L}(a_L(\mathrm{x}_i; \mathrm{w}_{1\ldots L}), y_i)$$

- El método para resolver la optimización es alguna variante de descenso por gradiente (más adelante . . .)

$$\mathrm{w}_{t+1} := \mathrm{w}_t - \eta_t \nabla_{\mathrm{w}} \mathcal{L}$$

Optimización en el Aprendizaje Automático

Aprendizaje automático supervisado

Optimización en el Aprendizaje Automático

Algoritmos de Aprendizaje (optimización)
- Descenso de gradiente
- Descenso de gradiente estocástico
- Learning Rate adaptativo

Buenas prácticas de optimización en Aprendizaje Profundo

Optimización y Aprendizaje Automático (ML)

- Optimización y ML son equivalentes en la práctica (técnicas, . . .)
- ML es indirecto, no busca minimizar una medida de performance P, sino una función de pérdida $\mathcal{L}$ que esperamos mejore a P

  $\Rightarrow$ se elige $\mathcal{L}$ para que sea fácil de optimizar
- Otras (no) diferencias:
  - Optimización es *offline* y ML puede ser *online*. Falso: la optimización muchas veces tiene restricciones de tiempo real
  - Optimización busca el óptimo global, y ML se conforma con menos. Falso: optimización por lo general se conforma con menos por restricción de recursos. Cuando se busca el óptimo global se llama optimización pura

Optimización Pura, ML y Minimización empírica del riesgo

- Optimización pura busca el óptimo global
  - Modela el problema como una optimización (lo mejor posible)
  - Resuelve la optimización (lo mejor posible)
- Aprendizaje Automático
  - $y = f(\mathrm{x}, \boldsymbol{\theta})$ salida del modelo cuando la entrada es x
  - $\mathcal{L}$ función de pérdida por muestra
  - p_{data} distribución real de los datos
  - se busca minimizar el error para cualquier dato real

$$\boldsymbol{\theta}^* \leftarrow \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathbb{E}_{(\mathrm{x},y)\sim p_{data}} \mathcal{L}(f(\mathrm{x};\boldsymbol{\theta}), y)$$

$\Rightarrow$ p_{data} no conocida $\rightarrow$ aparece sobre-ajuste

- Minimización empírica del riesgo
  - problema de optimización relacionado al problema de ML

$$\boldsymbol{\theta}^* \leftarrow \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathbb{E}_{(\mathrm{x},y)\sim \hat{p}_{data}} \mathcal{L}(f(\mathrm{x};\boldsymbol{\theta}), y)$$

  - $\hat{p}_{data}$ distribución empírica de los datos de entrenamiento
  - no asegura ser lo mejor para el problema ML
  - simplificaciones en la función de pérdida
    - elegida para facilitar la optimización
    - suma para cada muestra $\rightarrow$ facilita métodos iterativos

Problemas en la minimización empírica del riesgo (1/2)

- distribución real de los datos desconocida. En DL significa que el gradiente es aproximado siempre!
- la función de pérdida $\mathcal{L}()$ no puede ser optimizada eficientemente. Por ejemplo en un problema de clasificación 0/1, no es diferenciable
  - En estos casos usamos una surrogate loss function, por ejemplo en clasificación 0/1 se usa la probabilidad de las clases 0 o 1
- el criterio de parada temprana, sucede lo mismo que en optimización pura: limitar la cantidad de cómputo, cantidad de iteraciones, mejora en cada iteración, etc . . . Con el agregado que en ML además se puede usar como parada monitorear la $\mathcal{L}()$ sobre un conjunto extra de validación (hacerlo es muy común porque evita el sobre-ajuste).

Problemas en la minimización empírica del riesgo (2/2)

- comunes a otros problemas de optimización:
  - Mínimos locales, platos, punto silla
  - Hessiana mal condicionada
  - Acantilados y gradientes explosivos: se soluciona con "gradient clipping", donde el paso en el descenso se acota
  - Gradientes que desaparecen. Específico cuando se optimiza descendiendo por gradiente. ver más adelante ...
- batch y mini-batch. Aprovechar que $\mathcal{L}$ es una suma sobre muestras y usar un subconjunto para optimizar. Muy útil cuando el conjunto de datos es grande.
  - batch $\rightarrow$ determinista. El algoritmo no usa esta propiedad y cada iteración entrena con todos los datos
  - mini-batch $\rightarrow$ estocástico. Se usa solo un subconjunto de los datos para entrenar en cada iteración.
    $\Rightarrow$ introduce error, debe ser aleatorio, permite paralelización!

No todos son problemas, muchos métodos de ML se diseñan para ser problemas de optimización convexos. Lamentablemente no es el caso de DL

Algoritmos de Aprendizaje (optimización)

Descenso de gradiente en una dimensión (1/3)

- Caso de una dimensión es bien ilustrativo!, buscamos

$$x^* \longleftarrow \operatorname*{argmin}_x f(x)$$

- $f : \mathbb{R} \to \mathbb{R}$ función continua y diferenciable
- Taylor de primer orden

$$f(x+\epsilon) = f(x) + \epsilon f'(x) + O(\epsilon^2)$$

- Elegimos dar un paso fijo de tamaño $\eta > 0$ hacia donde decrece la función, eligiendo $\epsilon = -\eta f'(x)$, entonces

$$f(x - \eta f'(x)) = f(x) - \eta f'^2(x) + O(\eta^2 f'^2(x))$$

- Si la derivada no desaparece ($f'(x) \neq 0$), efectivamente el paso decrece ($\eta f'^2(x) > 0$), y si el paso es pequeño ($O() \approx 0$)

$$f(x - \eta f'(x)) \lessapprox f(x)$$

$\Rightarrow$ Si iteramos $x_{t+1} := x_t - \eta f'(x_t)$, iremos reduciendo $f(x)$ (mientras se cumplan las hipótesis)

Descenso de gradiente en una dimensión
Pseudocódigo (2/3)

- El algoritmo más sencillo de descenso por gradiente (GD) es

Input : Starting x_0, η fixed step size (learning rate), E epochs
Output: x cercano a mínimo local

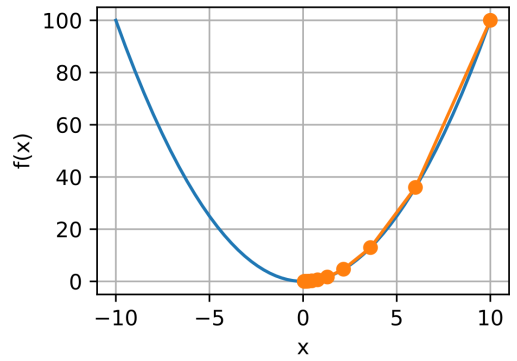
$x := x_o$; $e = 1$; % Initialize trayectory
% Repeat until the stop condition is reached
while $e \leq E$ **do**
  $x := x - \eta f'(x)$
end
return x;

- Varios criterios de parada deben usarse:
  - cantidad de iteraciones
  - tiempo de cómputo
  - $|f'(x)|$ es suficientemente pequeño
  - pérdida en un conjunto de validación ...

Descenso de gradiente en una dimensión
Epochs y learning rate (3/3)

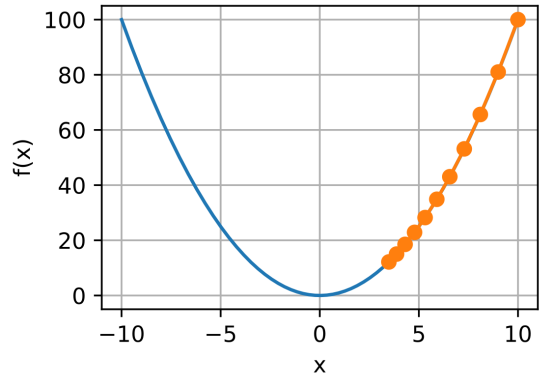
Ejemplo $f(x) = x^2$ y $f'(x) = 2x$

- $E = 10$, $\eta = 0{,}2 \Rightarrow$
  $x = 0{,}0604\ldots$



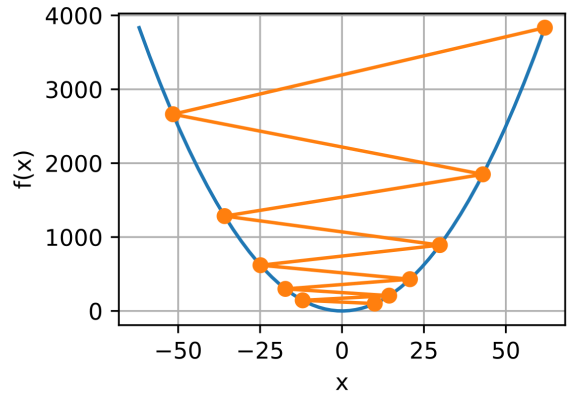
- $E = 10$, $\eta = 0{,}05 \Rightarrow$
  $x = 3{,}4867\ldots$

  muy lento y no se acerca a 0



- $E = 10$, $\eta = 1{,}1 \Rightarrow$
  $x = 61{,}9173\ldots$

  muy rápido y diverge



Descenso de gradiente

- $x \in \mathbb{R}^d$, $f : \mathbb{R}^d \to \mathbb{R}$ función continua y diferenciable
- el gradiente es

$$\nabla f(x) = \left[\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \ldots, \frac{\partial f(x)}{\partial x_d} \right]^\top$$

- Taylor, con $\epsilon \in \mathbb{R}^d$

$$f(x + \epsilon) = f(x) + \epsilon^\top \nabla f(x) + O(\|\epsilon\|^2)$$

$\Rightarrow$ Ahora la iteración que reduce $f(x)$ es

$$x_{t+1} := x_t - \eta \nabla f(x_t)$$

Descenso de gradiente en Aprendizaje Profundo (DL)

- Recordar, la función de pérdida es una suma en la cantidad de muestras, y el aprendizaje es el problema de minimizar el error empírico para los datos de entrenamiento

$$\mathrm{w}^* \longleftarrow \operatorname*{argmin}_{\mathrm{w}} \sum_{i=1}^{N} \mathcal{L}(a_L(\mathrm{x}_i; \mathrm{w}_{1\ldots L}), y_i)$$

- El método de descenso de gradiente y variantes se basa en

$$\mathrm{w}_{t+1} := \mathrm{w}_t - \eta_t \nabla_{\mathrm{w}} \mathcal{L}$$

donde nos reservamos variar η_t en el tiempo, y el gradiente sobre todo el conjunto de datos de entrenamiento (Batch Gradient Descent) es

$$\nabla_{\mathrm{w}} \mathcal{L}(\mathrm{x}) = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\mathrm{w}} \mathcal{L}(a_L(\mathrm{x}_i; \mathrm{w}_{1\ldots L}), y_i)$$

Recordar: es una aproximación al gradiente real, $\nabla^*_{\mathrm{w}} \mathcal{L}$, si conociéramos la distribución real de los datos

Batch Gradient Descent

Batch Gradient Descent (GD)

Usar todo el conjunto de datos de entrenamiento para estimar el gradiente

- Ventajas
  - condiciones de convergencia conocidas
  - se puede acelerar la convergencia usando segundo orden de derivada (basado en la Hessiana)
- Desventajas
  - datos demasiado grandes para calcular todo el gradiente, muy lento
  - sin garantías de alcanzar un buen óptimo
  - difícil de calcular el gradiente empírico, si la función es poco convexa

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent

Usar una porción aleatoria del conjunto de datos de entrenamiento (mini-batch) para estimar el gradiente

- puede verse como que es una aproximación Monte Carlo del gradiente empírico
- no parece grave aproximar, dado que ya era aproximado (?...)

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$

Require: Initial parameter $\boldsymbol{\theta}$

$k \leftarrow 1$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$

$k \leftarrow k + 1$

end while

Stochastic Gradient Descent (SGD)

- por lo general se usa η constante o decreciendo linealmente en las primeras épocas hasta estancarse en un valor pequeño
- Ventajas
  - condiciones de convergencia conocidas para algunos η, pero no usadas en la práctica. $\rightarrow$ en la práctica graficar la pérdida en las épocas y ajustar el η
  - aleatoriedad ayuda a evitar sobre-ajuste
  - más rápido que GD
  - por lo general consigue mejores resultados que GD para grandes datos, esto puede deberse a que escapa de óptimos locales más comúnmente
  - posible de aplicar con datos que varían con el tiempo (streaming). GD tiene sesgo hacia datos viejos
- algunos problemas persisten
  - mal condicionado, desaparición y explosión de gradientes
  - escapar de mínimos locales, existencia de platos y acantilados
  - difícil de reflejar dependencia de largo aliento (RNN)

Mejorar SGD con optimización de segundo orden?

- en SGD todos los pesos se actualizan de la misma forma
- idea: adaptar el aprendizaje para cada peso

$$\mathrm{w}_{t+1} := \mathrm{w}_t - {\mathcal{H}_{\mathcal{L}}}^{-1} \eta_t \nabla_{\mathrm{w}} \mathcal{L}$$

donde $\mathcal{H}_{\mathcal{L}}$ es la matriz Hessiana de $\mathcal{L}$, cuyos elementos son

$${\mathcal{H}_{\mathcal{L}}}^{ij} = \frac{\partial^2 \mathcal{L}}{\partial \mathrm{w}_i \partial \mathrm{w}_j}$$

- Consideraciones
  - $\mathcal{H}_{\mathcal{L}}$ fácil de calcular con los datos
  - matriz prohibitivamente grande
  - computar la inversa de $\mathcal{H}_{\mathcal{L}}$ es costoso
  - $\mathcal{H}_{\mathcal{L}}$ puede ser aproximada, por ejemplo con el algoritmo $L - BFGS$
  - en la práctica se usan momentos, más sencillo y fácil

Stochastic Gradient Descent (SGD) con momentos (impulso, inercia)

- El aprendizaje en SGD puede ser lento

  $\Rightarrow$ usar una variable de velocidad (inercia) en la dirección de decrecimiento

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
  end while

Adagrad [Duchi2011]

- Se adapta el learning rate de cada peso escalándolo inversamente proporcional a la raíz de la suma de los cuadrados de todos los gradientes anteriores

  $\Rightarrow$ el gradiente va reduciéndose con el tiempo

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ

Require: Initial parameter $\boldsymbol{\theta}$

Require: Small constant δ, perhaps 10^{-7}, for numerical stability

Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$.

Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$.

end while

RMSProp [Hinton2012]

- Parecido a AdaGrad, el learning rate se adapta de acuerdo a un promedio móvil ponderado exponencialmente

  $\Rightarrow$ adapta mejor a problemas no convexos

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ

Require: Initial parameter $\boldsymbol{\theta}$

Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers

Initialize accumulation variables $\boldsymbol{r} = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$.

Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$.

end while

Adam [Kingma2014]

- Combina RMSProp y momentos $\Rightarrow$ se llama "ADAptive Moments"

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

¿Cuál variante elegir?

- La mayoría de las variantes implementadas en paquetes comerciales
- No hay un algoritmo mejor, depende del caso [Schaul et al. (2014)]

Buenas prácticas en Aprendizaje Profundo

Aprendizaje automático supervisado

Optimización en el Aprendizaje Automático

Algoritmos de Aprendizaje (optimización)

- Descenso de gradiente
- Descenso de gradiente estocástico
- Learning Rate adaptativo

Buenas prácticas de optimización en Aprendizaje Profundo

Normalización de la entrada (1/2)

Es más difícil y más lento aprender datos no normalizados/estandarizados

Estandarizar

1. calcular media μ y desviación σ de los datos de entrenamiento
2. restar esa media y luego dividir por la desviación a los tres conjunto de datos *train/validation/test* $\left(\frac{x-\mu}{\sigma}\right)$

Normalizar

1. calcular mínimo x_{min} y máximo x_{max} de los datos de entrenamiento
2. normalizar los tres conjuntos de datos *train/validation/test* $\left(\frac{x-x_{min}}{x_{max}-x_{min}}\right)$

- ¿normalizar o estandarizar?
  - distribución gaussiana de los datos $\Rightarrow$ estandarizar
  - existencia de valores atípicos $\Rightarrow$ estandarizar
  - modelos de ML asumen datos gaussianos (no es el caso de DL) $\Rightarrow$ estandarizar

Normalización de la entrada (2/2)

- En DL, la funciones de activación *ReLu*() y *tanh*() centradas en cero
  - normalización se propagará a la siguiente capa
  - mejor sensibilidad al gradiente entorno al cero
  - *sigmoid*() no centrada en cero $\Rightarrow$ hay bias al cambiar de capa ...
- puede bastar centrar en la media, si toda la entrada tiene el mismo rango, ej. imágenes

De-correlación de la entrada

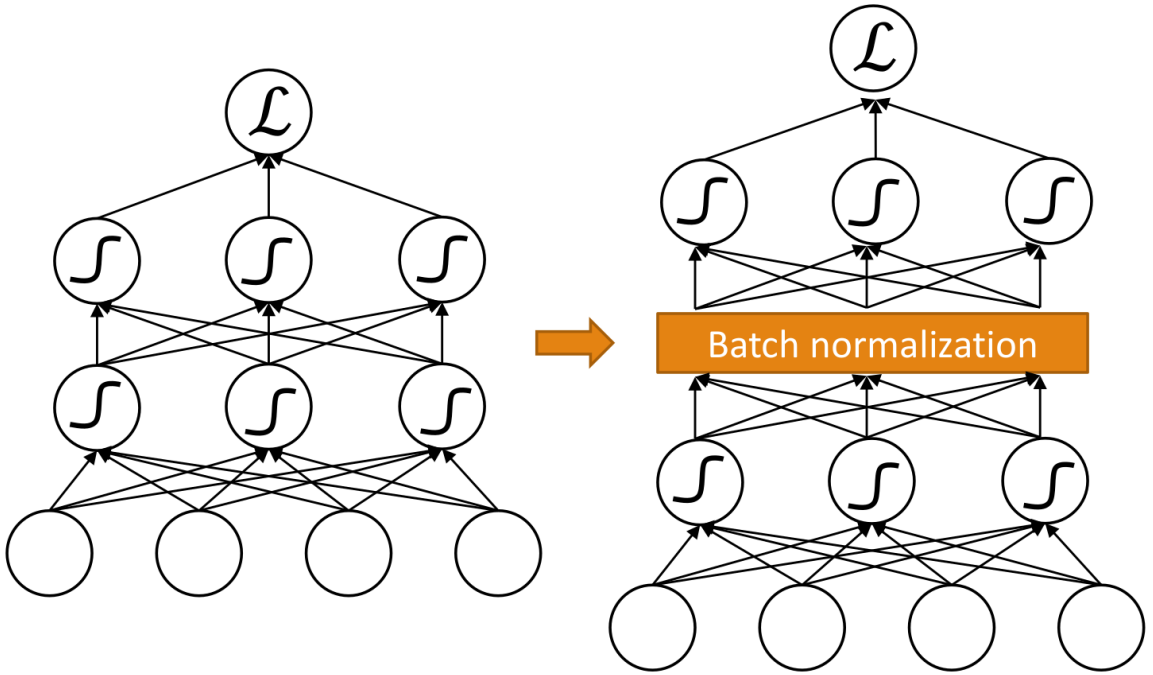
Decorrelacionar

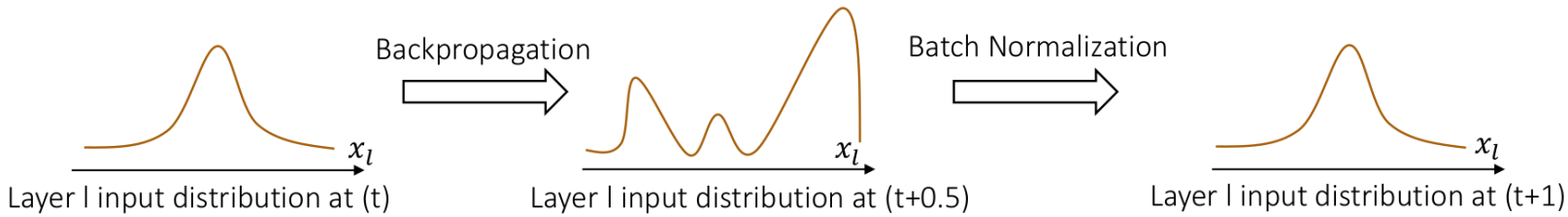
reducir la autocorrelación en los datos

- Si los datos de entrada x están correlacionados vale transformarlos y de-correlacionarlos
  - el aprendizaje se enfoca en aprender cosas no triviales
  - excepto cuando se quiera aprender la correlación, como en las secuencias y series de tiempo
- hay muchas técnicas, su elección depende de los datos

Normalización Batch (1/2)

- en cada capa, los pesos se adaptan a los datos y además se adaptan para seguir la distribución de la entrada





Normalización Batch (2/2)

- Ajuste en la regla de actualización de SGD
  1. computar media y varianza del mini-batch
     $\mu_{\mathcal{B}} := \frac{1}{m}\sum_{i=1}^{m} x_i$, $\sigma_{\mathcal{B}} := \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$
  2. normalizar la entrada de la capa $\hat{x}_i := \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
  3. escalar y desplazar la entrada de la capa $\hat{y}_i := \gamma\hat{x}_i + \beta$
     (γ y β entrenados)
- Normalización Batch muy útil para
  - hacer redes más profundas
  - permitir más velocidad con mayores learning rates
  - reducir sobre-ajuste

Lidear con el sobre-ajuste: Regularización y otros

- En DL tenemos miles/millones de parámetros $\Rightarrow$ altas chances de sobre-ajuste
- la regularización de la función objetivo en la optimización permite evitar el sobre-ajuste

$$\mathrm{w}^* \longleftarrow \underset{\mathrm{w}}{\operatorname{argmin}} \sum_{i=1}^{N} \mathcal{L}(a_L(\mathrm{x}_i; \mathrm{w}_{1\ldots L}), y_i) + \lambda \Omega(\mathrm{w})$$

- varios métodos para lidear con el sobre-ajuste
  - regularización $\ell 2$
  - regularización $\ell 1$
  - criterio de parada temprano
  - dropout
  - ...

Regularización $\ell 2$

- la regularización más importante y más popular

$$\mathrm{w}^* \longleftarrow \underset{\mathrm{w}}{\operatorname{argmin}} \sum_{i=1}^{N} \mathcal{L}(a_L(\mathrm{x}_i; \mathrm{w}_{1\ldots L}), y_i) + \frac{\lambda}{2} \sum_{j=1}^{L} \mathrm{w}_j^2$$

- ajuste en la regla de actualización de SGD

$$\mathrm{w}_{t+1} := \mathrm{w}_t - \eta_t \left(\nabla_{\mathrm{w}} \mathcal{L} + \lambda \mathrm{w}_t \right)$$

entonces

$$\mathrm{w}_{t+1} := (1 - \eta_t \lambda) \mathrm{w}_t - \eta_t \nabla_{\mathrm{w}} \mathcal{L}$$

(los pesos disminuyen)

- usualmente $\lambda \approx 10^{-2}, 10^{-1}$

Regularización $\ell 1$

- la regularización $\ell 1$ también es muy popular

$$\mathrm{w}^* \longleftarrow \underset{\mathrm{w}}{\operatorname{argmin}} \sum_{i=1}^{N} \mathcal{L}(a_L(\mathrm{x}_i; \mathrm{w}_{1\ldots L}), y_i) + \frac{\lambda}{2} \sum_{j=1}^{L} |\mathrm{w}_j|$$

- ajuste en la regla de actualización de SGD

$$\mathrm{w}_{t+1} := \mathrm{w}_t - \eta_t \left(\nabla_{\mathrm{w}} \mathcal{L} + \lambda \frac{\mathrm{w}_t}{|\mathrm{w}_t|} \right)$$

entonces

$$\mathrm{w}_{t+1} := \left(1 - \frac{\eta_t \lambda}{|\mathrm{w}_t|} \right) \mathrm{w}_t - \eta_t \nabla_{\mathrm{w}} \mathcal{L}$$

- la regularización $\ell 1$ genera muchos pesos nulos
- si aumentamos λ obtenemos más pesos nulos

Criterio de parada temprano en el entrenamiento

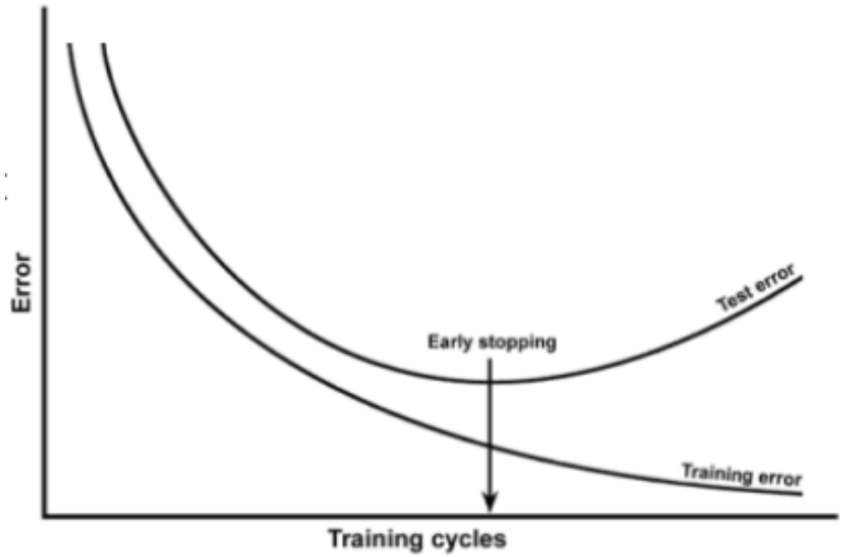
- Se controla la parada para evitar el sobre-ajuste
- se monitorea con un conjunto independiente de validación

- el entrenamiento debe disminuir la pérdida en los datos de entrenamiento y en los datos de validación

  aunque usualmente con una tasa más lenta

- el entrenamiento se detiene cuando la pérdida aumenta en validación

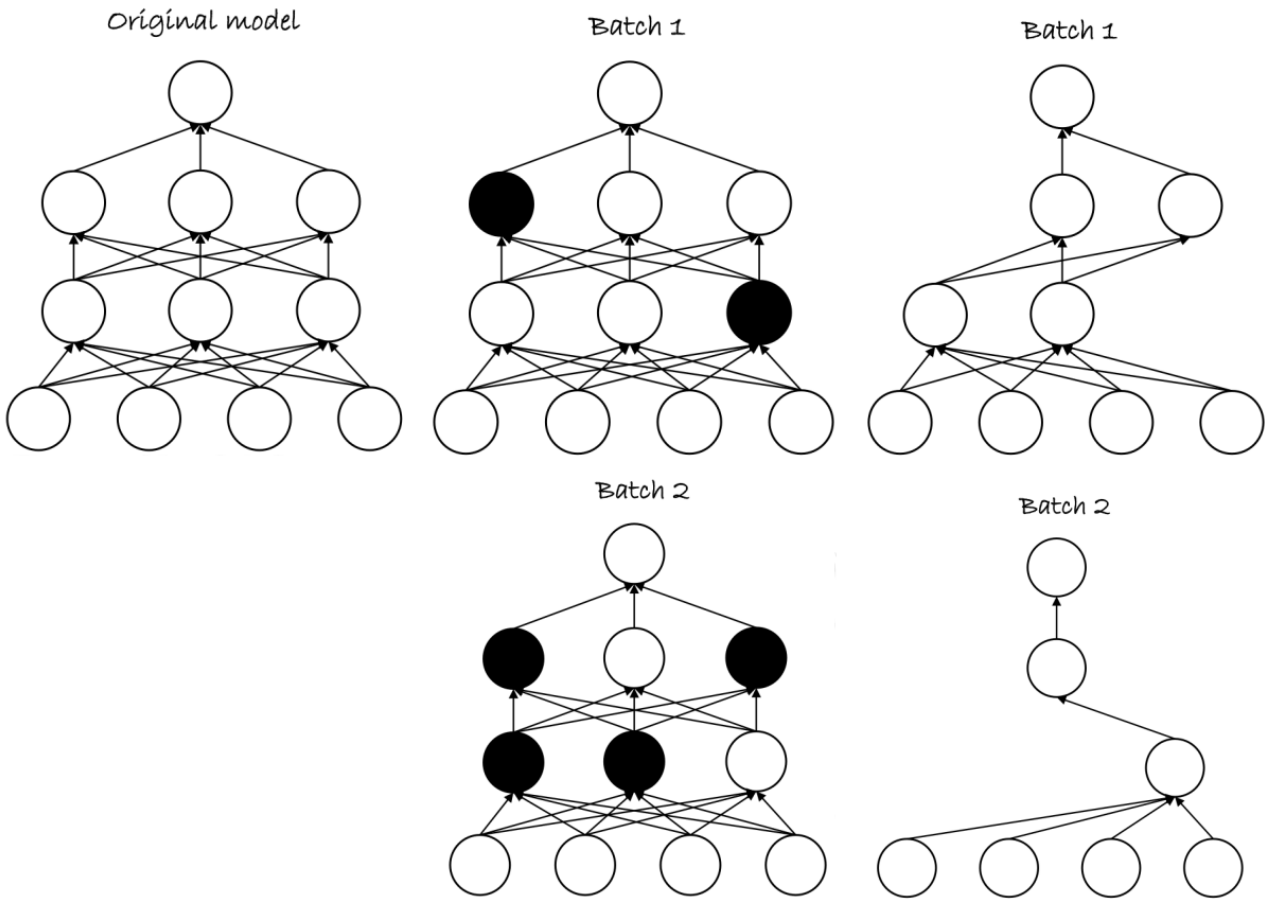
  que usualmente significa que comienza a sobre-ajustar



Dropout [Srivastava2014]

Dropout

aleatoriamente definir activaciones nulas de algunas neuronas (en el entrenamiento)



- una red distinta para cada mini-batch $\approx$ ensamblado de modelos
- en testing todas las neuronas son usadas
- beneficio: la red es más robusta, y reduce el sobre-ajuste

Learning rate adaptativo por peso

- vimos que es muy relevante en velocidad, exactitud y convergencia
- es la principal diferencia entre los algoritmos de SGD

Inicialización de pesos $w_{1...L}$

- pesos muy pequeños $\Rightarrow$ las señales se propagan muy poco en la red y es difícil aprender
- pesos muy grandes $\Rightarrow$ la red satura
- se conocen buenas elecciones dependiendo de las funciones de activación ...

Bibliografía

- Ian Goodfellow, and Yoshua Bengio, and Aaron Courville. Deep Learning, Chapter 8. MIT Press. 2016. http://www.deeplearningbook.org
- UVA Deep Learning Course. MSc in Artificial Intelligence for the University of Amsterdam. http://uvadlc.github.io/ *(estas filminas se basan en este curso...)*