Taller de Aprendizaje Automático

Entrenamiento de Redes Neuronales Profundas (Parte 2)

Instituto de Ingeniería Eléctrica
Facultad de Ingeniería

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

1. Tiempo de entrenamiento: variantes de SGD
   - Descenso por gradiente estocástico (SGD)
   - SGD con *momentum*
   - *Momentum* de Nesterov
   - AdaGrad
   - RMSProp
   - ADAM

2. Regularización
   - Regularización por norma de parámetros
   - Regularización por Perturbaciones Aleatorias

Entrenamiento de redes profundas: dificultades mayores

- Desvanecimiento o explosión del gradiente
- Cantidad de datos insuficientes para el problema a abordar
- Tiempo de entrenamiento
- Sobre-ajuste

Aprendizaje como optimización

- Queremos encontrar f aproximación de una función en un conjunto de puntos $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$,
- Definimos una arquitectura de red: $f(\mathbf{x}; \mathbf{W})$, con parámetros $\mathbf{W}$.
- Minimización de la discrepancia/costo (+Regularización):

$$\hat{\mathbf{W}} = \text{argmin}_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^{n} L(f(\mathbf{x}_i; \mathbf{W}), y_i) + R(\mathbf{W})$$

$L(\hat{\mathbf{y}}, \mathbf{y})$: Función de costo/discrepancia,

$R(\mathbf{W})$: Regularizador (penaliza según valores de $\mathbf{W}$)

Descenso por Gradiente

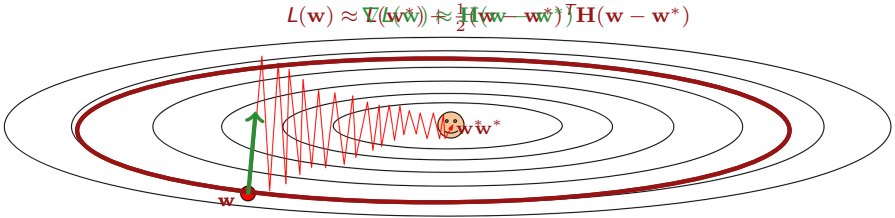
$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \nabla_{\mathbf{W}} L(\mathbf{W}),$$

donde $\eta > 0$ es el paso o *learning rate*

Descenso por Gradiente: Desafíos

¿Qué sucede si la *loss* es mal condicionada?

(mal condicionada: cambia mucho en una dirección y poco en otra)

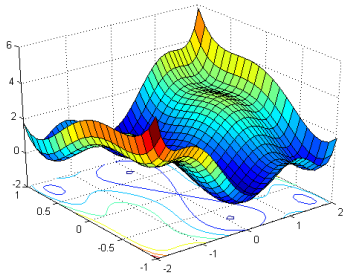


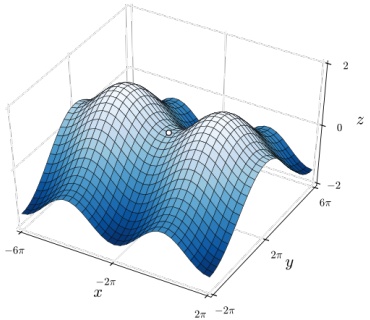
Mal condicionamiento se mide por el número de condición del Hessiano $\mathbf{H}$ (derivada segunda de la *loss*)

Descenso por Gradiente: Desafíos

¿Qué sucede con los mínimos locales y los puntos silla?

- En mínimos locales o puntos silla: $\nabla L(\mathbf{W}) = 0$; **GD** se detiene.
- Punto silla: Hessiano $\mathbf{H}$ tiene valores propios de distinto signo; frecuente en alta dimensión





- Cerca de los puntos silla: se avanza muy muy lento
- Análisis del espacio de funciones generadas por redes neuronales: área **activa** de investigación

Descenso por Gradiente Estocástico (SGD)

Descenso por gradiente (GD)

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \nabla_{\mathbf{W}} L(\mathbf{W}) \quad \eta > 0 \ \ (\text{“}\textit{learning rate}\text{”}).$$

- Dataset muy grande $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n \gg 1$, es muy costoso calcular:

$$\nabla_{\mathbf{W}} L(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\mathbf{W}} L(\mathbf{x}_i, y_i; \mathbf{W})$$

Descenso por gradiente Estocástico (SGD)

- **Aproximar** suma utilizando pequeño conjunto (minibatch), $n_{\text{mb}} = 1, 2, 4, ..., 128, ... \ll n$.
- En cada paso se utiliza un subconjunto (minibatch) diferente $\text{mb}(t)$:

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \left[\frac{1}{n_{mb}} \sum_{i \in \text{mb}(t)} \nabla_{\mathbf{W}} L(\mathbf{x}_i, y_i; \mathbf{W}) \right]$$

- Gradiente estimado **ruidoso**

Descenso por Gradiente Estocástico (SGD)

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

"Deep Learning" - Goodfellow-Bengio-Courville, 2016

SGD con *Momentum*

Momentum SGD	SGD
$v_{t+1} = \rho v_t - \eta \nabla f(x_t)$ $x_{t+1} = x_t + v_{t+1}$	$x_{t+1} = x_t - \eta \nabla f(x_t)$

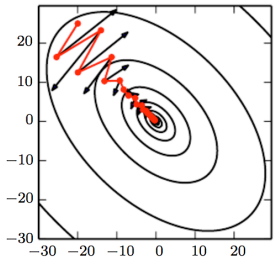
- Diseñado para tener **inercia**
- ρ: parámetro de memoria/fricción; típicamente $\rho = 0.9$ o 0.99
- **Dirección de descenso:** promedio de velocidad y gradiente
- Especialmente adaptado para cuando hay alta curvatura o ruido en el gradiente
- **v** : velocidad, $v_0 = 0$

SGD con *Momentum*

Momentum SGD

$$v_{t+1} = \rho v_t - \eta \nabla f(x_t)$$

$$x_{t+1} = x_t + v_{t+1}$$



- En SGD el largo del paso es $\eta\|\mathbf{g}\|$, donde $\mathbf{g}$ es el gradiente
- En Momentum SGD depende de toda la secuencia (histórico)
- Si el gradiente siempre apunta para el mismo lado $\mathbf{g}$, el paso llega a velocidad terminal, $\frac{\eta\|\mathbf{g}\|}{1-\rho}$

Influencia de η y ρ: https://distill.pub/2017/momentum/

SGD con *Momentum*

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

```
optimizer = keras.optimizers.SGD(lr=0.001, momentum=0.9)
```

"Deep Learning" - Goodfellow-Bengio-Courville, 2016

Nesterov *Momentum*[*†]

Nesterov Momentum

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

Momentum SGD

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

- **Nesterov:** Calcular el gradiente en el punto que se llegaría si se continúa a la misma velocidad v
- En caso convexo se puede probar que acelera la convergencia

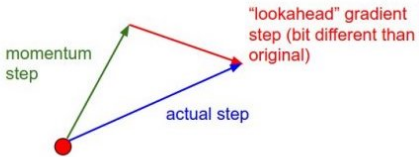
[*] Y. Nesterov, "A method for solving the convex programming problem with convergence rate $o(1/k^2)$," *Proceedings of the USSR Academy of Sciences*, vol. 269, pp. 543–547, 1983

[†] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, p. III–1139–III–1147, JMLR.org, 2013

Nesterov *Momentum*

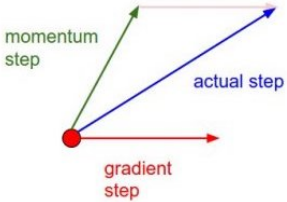
Nesterov Momentum

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



Momentum SGD

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



Nesterov *Momentum*

Nesterov Momentum

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

Requiere calcular un nuevo punto intermedio $x_t + \rho v_t$ y evaluar $\nabla f(x_t + \rho v_t)$

Sea $\tilde{x}_t = x_t + \rho v_t$, entonces:

$$\begin{aligned} v_{t+1} &= \rho v_t - \eta \nabla f(\tilde{x}_t) \\ \tilde{x}_{t+1} &= x_{t+1} + \rho v_{t+1} \\ &= (x_t + v_{t+1}) + \rho v_{t+1} \\ &= \tilde{x}_t - \rho v_t + (1+\rho) v_{t+1} \\ &= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t) \end{aligned}$$

Se puede evitar agregando un término de corrección: $\rho(v_{t+1} - v_t)$ y trabajando directamente con $\tilde{x}_t$.

```
optimizer = keras.optimizers.SGD(lr=0.001, momentum=0.9, nesterov=True)
```

Algoritmos con *learning rate* adaptativo

AdaGrad*

AdaGrad

$$r_t = r_{t-1} + \nabla f(x_t) \odot \nabla f(x_t)$$

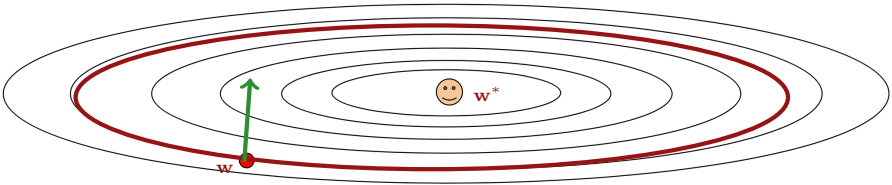
$$x_{t+1} = x_t - \eta \frac{1}{\sqrt{r_t} + \delta} \odot \nabla f(x_t)$$

Re-escalado en cada dimensión inversamente proporcional a la suma histórica de los cuadrados del gradiente.

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

¿Qué hace? ¿Qué pasa cuándo $t \gg 1$?

- Direcciones con **más** información (mayor gradiente) se mueven menos
- Luego de varias iteraciones: el *learning rate* es muy pequeño
- Consecuencia de la acumulación desde el inicio ($t = 0$)



*J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, p. 2121–2159, July 2011.

AdaGrad

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

"Deep Learning" - Goodfellow-Bengio-Courville, 2016

RMSProp*

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Re-escalado en cada dimensión inversamente proporcional a la suma histórica de los cuadrados del gradiente.

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Se mantiene una **media móvil** con el cuadrado del gradiente para cada componente

- Extensión natural de AdaGrad: Coeficiente de olvido (*decay rate* ρ)
- La tasa de "olvido" es exponencial, es una especie de *Momentum* pero en el cuadrado del gradiente (no en el gradiente)
- Es uno de los algoritmos más populares

*T. Tieleman, G. Hinton. Coursera: Neural networks for machine learning, 2012. "Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude."

RMSProp

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

```
optimizer = keras.optimizers.RMSprop(lr=0.001, rho=0.9)
```

"Deep Learning" - Goodfellow-Bengio-Courville, 2016

ADAM: Adaptive Moments

1. Mantener inercia (*momentum*, dir. $=$ grad $+$ dir. previa)
2. Mantener un estimador del cuadrado del gradiente e ir dividiendo el *learning rate*

Usando las dos ideas: **Adam = ADAptive Moments**

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Mantiene una estimación con media móvil del primer momento (Momentum) y del segundo momento (AdaGrad/RMSProp). Típicamente: $\beta_1 = 0.9$, $\beta_2 = 0.999$.

¿Qué pasa en t=0? **Primer paso:** `second_moment` ≈ 0, entonces al principio el paso es muy grande (independientemente de la geometría del problema; es un problema de la inicialización)

ADAM: Adaptive Moments*

ADAM (con corrección de *bias*)

```
first_moment = 0
second_moment = 0
for t in range(num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

- Mantiene las ideas previas (RMSProp + Momentum)
- Agrega corrección de sesgo (momentos) en func. del tiempo
- En la práctica: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\eta = 10^{-3}/10^{-4}$
- Funciona "bien" en gran variedad de problemas.
- Usar por defecto :)

*D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015

ADAM: 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

```
optimizer = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
```

"Deep Learning" - Goodfellow-Bengio-Courville, 2016

Comparación de optimizadores

https://imgur.com/a/Hqolp

Imágenes de *Alec Radford*

Learning Rate

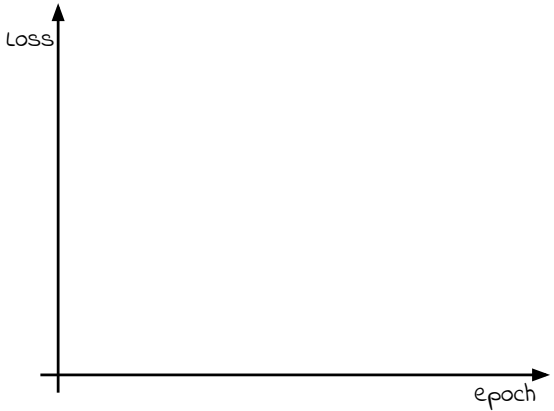
Todos los métodos vistos tienen un hiperparámetro: *learning rate*

Learning Rate

Todos los métodos vistos tienen un hiperparámetro: *learning rate*

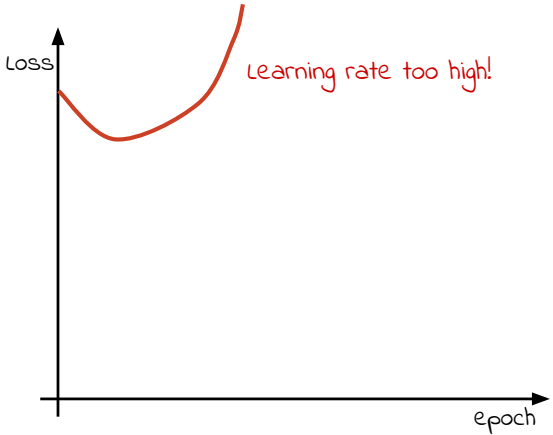
Learning Rate

Todos los métodos vistos tienen un hiperparámetro: *learning rate*



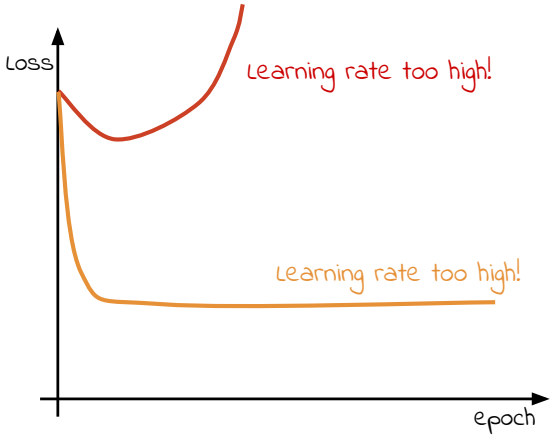
Learning Rate

Todos los métodos vistos tienen un hiperparámetro: *learning rate*



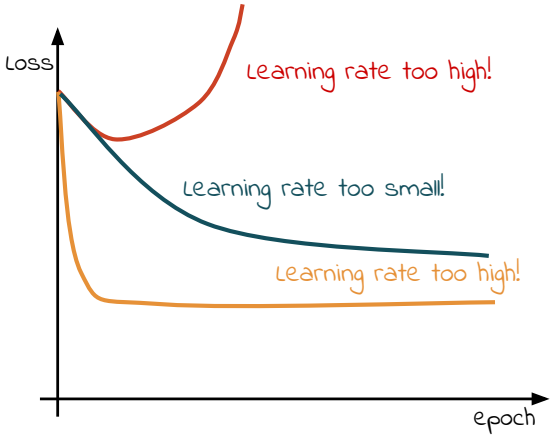
Learning Rate

Todos los métodos vistos tienen un hiperparámetro: *learning rate*



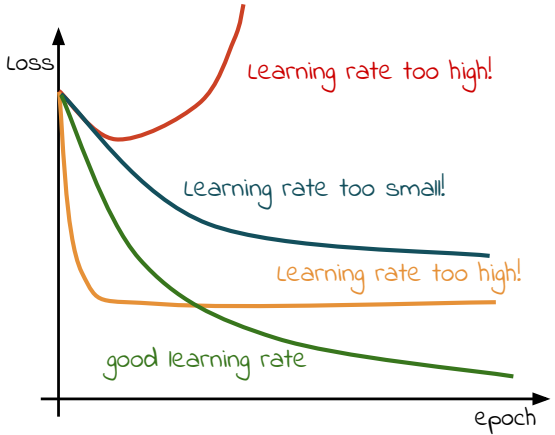
Learning Rate

Todos los métodos vistos tienen un hiperparámetro: *learning rate*



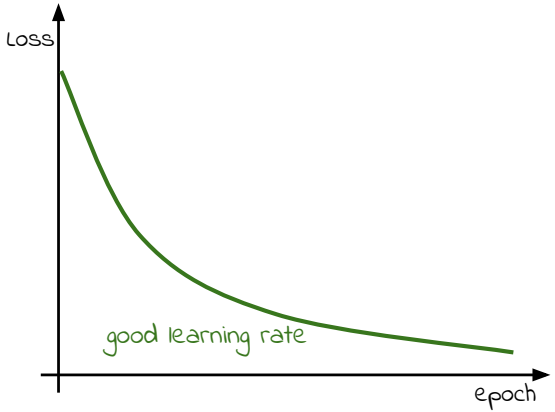
Learning Rate

Todos los métodos vistos tienen un hiperparámetro: *learning rate*



Learning Rate

Todos los métodos vistos tienen un hiperparámetro: *learning rate*



Learning rate *decay*

No hay por qué mantener el mismo *learning rate* durante todo el entrenamiento.

Reducción del *learning rate*:

- Power scheduling: $\eta(t) = \eta_0/(1 + t/s)^c$ (η_0, c, s: hiperparámetros)
- Exponential scheduling: $\eta(t) = \eta_0 \cdot 0.1^{t/s}$
- Piecewise constant scheduling: e.g. $\eta_0 = 0.1$ durante 5 épocas, $\eta_1 = 0.001$ durante 50 épocas, etc.
- Performance scheduling: se calcula el error de validación cada N pasos, y se reduce el learning rate un factor λ cuando se estanca.
- 1Cycle scheduling

Learning rate *decay*

No hay por qué mantener el mismo *learning rate* durante todo el entrenamiento.

Reducción del *learning rate*:

- Power scheduling: $\eta(t) = \eta_0/(1 + t/s)^c$ (η_0, c, s: hiperparámetros)
- Exponential scheduling: $\eta(t) = \eta_0 \cdot 0.1^{t/s}$
- Piecewise constant scheduling: e.g. $\eta_0 = 0.1$ durante 5 épocas, $\eta_1 = 0.001$ durante 50 épocas, etc.
- Performance scheduling: se calcula el error de validación cada N pasos, y se reduce el learning rate un factor λ cuando se estanca.
- 1Cycle scheduling

Learning rate *decay*

No hay por qué mantener el mismo *learning rate* durante todo el entrenamiento.

Reducción del *learning rate*:

- Power scheduling: $\eta(t) = \eta_0/(1 + t/s)^c$ (η_0, c, s: hiperparámetros)
- Exponential scheduling: $\eta(t) = \eta_0 \cdot 0.1^{t/s}$
- Piecewise constant scheduling: e.g. $\eta_0 = 0.1$ durante 5 épocas, $\eta_1 = 0.001$ durante 50 épocas, etc.
- Performance scheduling: se calcula el error de validación cada N pasos, y se reduce el learning rate un factor λ cuando se estanca.
- 1Cycle scheduling

Learning rate decay en Keras

Power scheduling con $c = 1$, $s =$ `decay`:

```
optimizer = keras.optimizers.SGD(lr=0.01, decay=1e-4)
```

Exponential scheduling:

```
def exponential_decay(lr0, s):
    def exponential_decay_fn(epoch):
        return lr0 * 0.1**(epoch / s)
    return exponential_decay_fn

exponential_decay_fn = exponential_decay(lr0=0.01, s=20)
```

```
lr_scheduler =
keras.callbacks.LearningRateScheduler(exponential_decay_fn)
history = model.fit(X_train_scaled, y_train, [...], callbacks=
[lr_scheduler])
```

`LearningRateScheduler` actualiza el `learning_rate` al comienzo de cada época.

Learning rate decay en Keras

Piecewise constant scheduling:

```python
def piecewise_constant_fn(epoch):
    if epoch < 5:
        return 0.01
    elif epoch < 15:
        return 0.005
    else:
        return 0.001
```

Luego se define el `lr_scheduler` con `piecewise_constant_fn` en lugar de `exponential_decay_fn`, y se ejecuta el método `fit`.

Performance scheduling:

```python
lr_scheduler = keras.callbacks.ReduceLROnPlateau(factor=0.5, patience=5)
```

Learning rate decay en Keras

- Una alternativa es implementar el *scheduling* mediante `tf.keras`, utilizando uno de los *schedulers* disponibles en `keras.optimizer.schedules`.
- En este caso la actualización del `learning_rate` se realiza en cada paso, en lugar de por época.
- Para implementar un *exponential schedule* debemos dividir por el tamaño del batch:

```
s = 20 * len(X_train) // 32 # number of steps in 20 epochs (batch size =
32)
learning_rate = keras.optimizers.schedules.ExponentialDecay(0.01, s, 0.1)
optimizer = keras.optimizers.SGD(learning_rate)
```

Regularización

La **regularización** es *"cualquier modificación que le introducimos a un algoritmo de aprendizaje con el objetivo de reducir el error de generalización (pero no el error de entrenamiento)"*

Tipos de Regularización:

- Regularización por penalización en la norma de parámetros
- Regularización por perturbaciones aleatorias
  - Data Augmentation (simetrías, deformaciones, repetición con ruido)
  - Dropout
  - Batch Normalization
- Regularización en la optimización
  - Detención temprana (*early stopping*)
  - SGD (aleatoriedad)
- Regularización por promediado de modelos (Bagging)

Penalización de norma de parámetros

Método (más) clásico de regularización: limitar capacidad del modelo agregando a la función de costo una penalidad a la norma de los parámetros $R(W)$:

$$L_R(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n} L_i(f(\mathbf{x}_i, \mathbf{W}), y_i) + \lambda R(\mathbf{W})$$

- $\lambda \geq 0$, es un hiperparámetro.
- En redes neuronales se penaliza solo $\mathbf{W}$ (no se penaliza los *bias* $\mathbf{b}$)
- Se podría tener distintos λ para pesar distintos parámetros (e.g., distintas capas), en general se utiliza el mismo λ.

Regularizaciones ℓ_1 y ℓ_2 en Keras

```
layer = keras.layers.Dense(100, activation="elu",
                           kernel_initializer="he_normal",

kernel_regularizer=keras.regularizers.l2(0.01))
```

```
from functools import partial

RegularizedDense = partial(keras.layers.Dense,
                           activation="elu",
                           kernel_initializer="he_normal",

kernel_regularizer=keras.regularizers.l2(0.01))

model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    RegularizedDense(300),
    RegularizedDense(100),
    RegularizedDense(10, activation="softmax",
                     kernel_initializer="glorot_uniform")
])
```

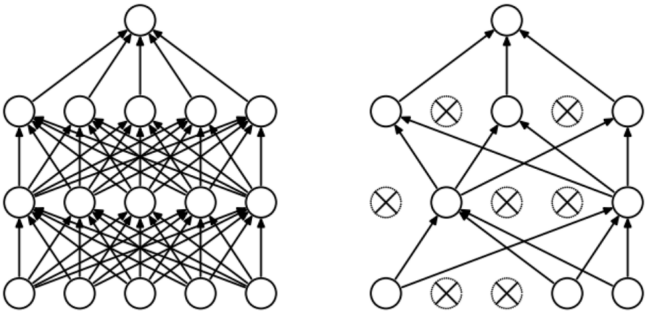
Regularización Max-Norm

- En lugar de agregar un término de regularización, restringe los pesos de forma que $\|\mathbf{w}\|_2 \leq r$
- Se implementa calculando $\|\mathbf{w}\|_2$ luego de cada paso de entrenamiento, y se re-escala si excede r ($\mathbf{w} \leftarrow \mathbf{w}r/\|\mathbf{w}\|_2$)
- A menor el hiperparámetro r, mayor la regularización.
- En Keras:

```
keras.layers.Dense(100, activation="elu", kernel_initializer="he_normal",
                   kernel_constraint=keras.constraints.max_norm(1.))
```

Regularización por ***dropout***[*†]

En la pasada hacia adelante durante el entrenamiento, se fijan a cero algunas neuronas de manera aleatoria (con probabilidad p)



(a) Standard Neural Net

(b) After applying dropout.

[*] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012. cite arxiv:1207.0580

[†] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, p. 1929–1958, Jan. 2014

Regularización por *dropout*

En la pasada hacia adelante durante el entrenamiento (*forward pass*), fijar a cero algunas neuronas de manera aleatoria (con probabilidad p)

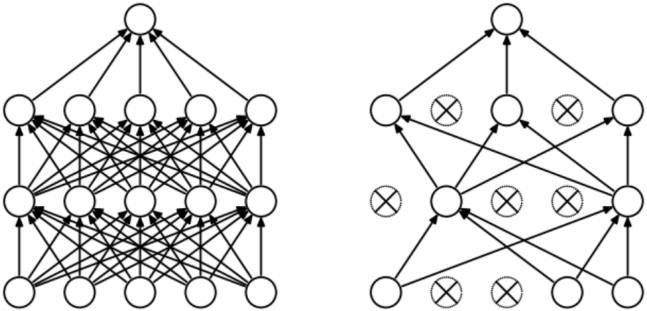
```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

Imagen tomada de cs231n (Stanford) - Fei-Fei Li & Justin Johnson & Serena Yeung

Regularización por *dropout*

Interpretación (I)

Se fuerza a la red a ser redundante, a aprender características robustas que puedan ser útiles con distintos subconjuntos de otras características provenientes de otras neuronas



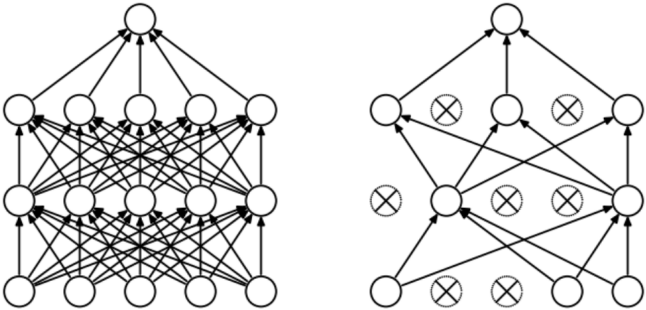
(a) Standard Neural Net

(b) After applying dropout.

Regularización por *dropout*

Interpretación (II)

Se produce un efecto *similar* al de entrenar distintos modelos, cada uno de ellos con una máscara binaria distinta ($2^{\#\text{neuronas}}$ posibles modelos). En *test* se combinan todos simultáneamente.



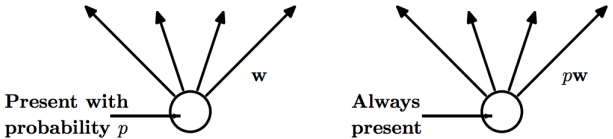
(a) Standard Neural Net

(b) After applying dropout.

Regularización por *dropout*

En predicción (*test*)

- En *test* todas las neuronas ven todas las entradas.
- Queremos que la salida de cada neurona en test sea igual a la salida esperada (o media) en entrenamiento
- Necesitamos re-escalar la salida en test por p
- Sea $\mathbf{x}$ la salida de una neurona sin dropout, con dropout la salida esperada será $p\mathbf{x} + (1-p)0$ (con probabilidad $1-p$ se fija a 0)
- En test si esta neurona está siempre activa debemos ajustar su valor $\mathbf{x} \rightarrow p\mathbf{x}$ para lograr la misma salida esperada

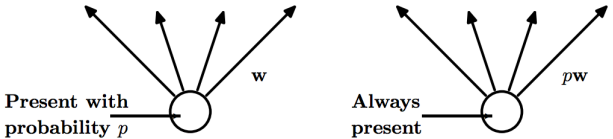


(a) At training time (b) At test time

Regularización por *dropout*

En predicción (*test*)

- En *test* todas las neuronas ven todas las entradas.
- Queremos que la salida de cada neurona en test sea igual a la salida esperada (o media) en entrenamiento
- Necesitamos re-escalar la salida en test por p
- Sea $\mathbf{x}$ la salida de una neurona sin dropout, con dropout la salida esperada será $p\mathbf{x} + (1-p)0$ (con probabilidad $1-p$ se fija a 0)
- En test si esta neurona está siempre activa debemos ajustar su valor $\mathbf{x} \rightarrow p\mathbf{x}$ para lograr la misma salida esperada

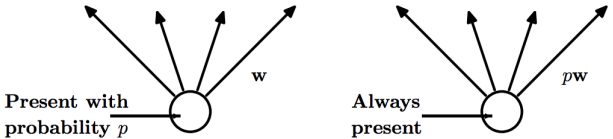


(a) At training time (b) At test time

Regularización por *dropout*

En predicción (*test*)

- En *test* todas las neuronas ven todas las entradas.
- Queremos que la salida de cada neurona en test sea igual a la salida esperada (o media) en entrenamiento
- Necesitamos re-escalar la salida en test por p
- Sea $\mathbf{x}$ la salida de una neurona sin dropout, con dropout la salida esperada será $p\mathbf{x} + (1-p)0$ (con probabilidad $1-p$ se fija a 0)
- En test si esta neurona está siempre activa debemos ajustar su valor $\mathbf{x} \rightarrow p\mathbf{x}$ para lograr la misma salida esperada

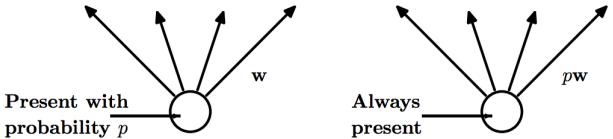


(a) At training time (b) At test time

Regularización por *dropout*

En predicción (*test*)

- En *test* todas las neuronas ven todas las entradas.
- Queremos que la salida de cada neurona en test sea igual a la salida esperada (o media) en entrenamiento
- Necesitamos re-escalar la salida en test por p
- Sea $\mathbf{x}$ la salida de una neurona sin dropout, con dropout la salida esperada será $p\mathbf{x} + (1-p)0$ (con probabilidad $1-p$ se fija a 0)
- En test si esta neurona está siempre activa debemos ajustar su valor $\mathbf{x} \rightarrow p\mathbf{x}$ para lograr la misma salida esperada



(a) At training time (b) At test time

Regularización por *dropout*

En predicción (*test*)

- En *test* todas las neuronas ven todas las entradas.
- Queremos que la salida de cada neurona en test sea igual a la salida esperada (o media) en entrenamiento
- Necesitamos re-escalar la salida en test por p
- Sea $\mathbf{x}$ la salida de una neurona sin dropout, con dropout la salida esperada será $p\mathbf{x} + (1-p)0$ (con probabilidad $1-p$ se fija a 0)
- En test si esta neurona está siempre activa debemos ajustar su valor $\mathbf{x} \rightarrow p\mathbf{x}$ para lograr la misma salida esperada.

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

Regularización por *dropout*

Para no **enlentecer** la etapa de test, se hace **dropout invertido** (re-escalado durante el entrenamiento). El código de predicción es igual que si no se hace dropout.

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

Imagen tomada de cs231n (Stanford) - Fei-Fei Li & Justin Johnson & Serena Yeung

Regularización por *dropout*

Implementación en Keras: capa de dropout.

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dropout(rate=0.2),
    keras.layers.Dense(300, activation="elu",
kernel_initializer="he_normal"),
    keras.layers.Dropout(rate=0.2),
    keras.layers.Dense(100, activation="elu",
kernel_initializer="he_normal"),
    keras.layers.Dropout(rate=0.2),
    keras.layers.Dense(10, activation="softmax")
])
```

Monte Carlo Dropout*

El trabajo de Gal et al. establece motivaciones adicionales para el uso de dropout:

- Revela conexiones entre las redes con dropout y la inferencia bayesiana (bases teóricas)
- Introduce la técnica de MC Dropout, que permite elevar el desempeño de cualquier red entrenada con dropout sin necesida de re-entrenarla.
- Muy fácil de implementar. En Keras:

```
y_probas = np.stack([model(X_test_scaled, training=True)
                     for sample in range(100)])
y_proba = y_probas.mean(axis=0)
```

- Hacemos 100 predicciones con `training=True` para asegurarnos que la capa de Dropout está activa. Estas 100 predicciones son distintas; inferimos tomando el promedio.
- Esto nos da una estimación Monte Carlo que suele mejorar la predicción.

*Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 1050–1059, PMLR, 20–22 Jun 2016

Monte Carlo Dropout

Comparemos las predicciones sin y con MC Dropout en Fashion MNIST:

```
>>> np.round(model.predict(X_test_scaled[:1]), 2)
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.99]],
      dtype=float32)
```

El model tiene la casi total certeza que la imagen pertenece a la clase 9 (*ankle boot*).
MC Dropout da el resultado siguiente:

```
>>> np.round(y_probas[:, :1], 2)
array([[[0.  , 0.  , 0.  , 0.  , 0.  , 0.14, 0.  , 0.17, 0.  , 0.68]],
       [[0.  , 0.  , 0.  , 0.  , 0.  , 0.16, 0.  , 0.2 , 0.  , 0.64]],
       [[0.  , 0.  , 0.  , 0.  , 0.  , 0.02, 0.  , 0.01, 0.  , 0.97]],
       [...]


>>> np.round(y_proba[:1], 2)
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.22, 0.  , 0.16, 0.  , 0.62]],
      dtype=float32)
```

El modelo sigue optando por la clase 9, pero conserva cierta duda de si no se trata de una instancia de la clase *sandal* o *sneaker*, lo cual tiene más sentido.

Monte Carlo Dropout

MC Dropout también permite cuantificar la incertidumbre asociada a las estimaciones de probabilidades, calculando el desvío estándar sobre las 100 predicciones:

```
>>> y_std = y_probas.std(axis=0)
>>> np.round(y_std[:1], 2)
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.28, 0.  , 0.21, 0.02, 0.32]],
      dtype=float32)
```

Finalmente, vemos que además permite alcanzar un mejor desempeño (sin MC Dropout era 86.8%):

```
>>> accuracy = np.sum(y_pred == y_test) / len(y_test)
>>> accuracy
0.8694
```

Referencias I


Y. Nesterov, "A method for solving the convex programming problem with convergence rate $o(1/k^2)$," *Proceedings of the USSR Academy of Sciences*, vol. 269, pp. 543–547, 1983.


I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, p. III–1139–III–1147, JMLR.org, 2013.


J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, p. 2121–2159, July 2011.


D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.


G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012.
cite arxiv:1207.0580.


N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, p. 1929–1958, Jan. 2014.


Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 1050–1059, PMLR, 20–22 Jun 2016.


A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition*.
O'Reilly Media, Inc., 2022.


T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*.
Springer Series in Statistics, New York, NY, USA: Springer New York Inc., 2001.

Referencias II


C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*.
Berlin, Heidelberg: Springer-Verlag, 2006.


X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, vol. 9, pp. 249–256, May 2010.


S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.


K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.


K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015.


S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, p. 448–456, JMLR.org, 2015.


J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, p. I–647–I–655, JMLR.org, 2014.


Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.


G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, p. 1527–1554, July 2006.