Título

UNAM

ESCUDO UNAM

Adrián Antonio Rodríguez Pié

00/00/0000

Resumen

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean felis massa, rhoncus et luctus tincidunt, varius a magna. Donec iaculis nunc nec consectetur consequat. Aenean erat dolor, maximus vel justo quis, fringilla ultrices urna. Pellentesque mattis pellentesque ante, vitae malesuada odio maximus a. Pellentesque dapibus, mauris sed fringilla aliquet, eros nibh gravida massa, sit amet consequat eros purus in purus. Fusce eu ligula tincidunt, consectetur turpis eget, imperdiet risus. Morbi urna tortor, lacinia bibendum iaculis accumsan, pharetra sed lectus. Nullam posuere nisl turpis, non sollicitudin orci commodo eget. Aenean pharetra, elit ut dictum pulvinar, mi enim scelerisque tortor, scelerisque pharetra ligula felis a leo. Nulla at consequat nulla. Mauris elementum metus sed tellus tempor tempor. Nam lobortis ante in odio condimentum pretium. Fusce ut hendrerit massa. Praesent volutpat molestie augue non semper.

Índice general

Capítulo 1

Capitulo 1

1.1. Tipos de aprendizaje

El parendizaje de máquina se ha ~~branched~~ en varios subcampos, cada uno atacando un tipo de problema distinto y utilizando diferentes tipos de aprendizaje. Se describen a continuación cuatro diferentes parámetros bajo los cuáles es posible clasificar los distintos paradigmas de aprendizaje.

Supervisado/no supervisado Dado que el aprendizaje involucra una interacción entre el ~~learner~~ y el ambiente, es posible dividir el aprendizaje con respecto a la naturaleza de dicha interacción. La primera distinción a notar es la diferencia entre aprendizaje supervisado y no supervisado. En abstracto, viendo el aprendizaje como un proceso de "usar la experiencia para ganar ~~expertise~~", el aprendizaje supervisado describe un escenario en el que la "experiencia", un ejemplo de entrenamiento, contiene información significativa que no está contenida en los "ejemplos de prueba" que el sistema aún no ha visto, y en los que se busca aplicar el ~~expertise~~ adquirido. En esta configuración, el ~~expertise~~ adquirido ~~is aimed~~ a predecir la información faltante de los datos de prueba. Podemos pensar en el ambiente como un profesor que "supervisa" al ~~learner~~ al proporcionarle información extra (una clasificación). Por otro lado, en el aprendizaje no supervisado no hay distinción entre datos de entrenamiento y de prueba. El ~~learner~~ procesa los datos de entrada con el objetivo de generar una síntesis o versión comprimida de los datos que sea representativa.

~~learners~~ pasivos/activos En los paradigmas de aprendizaje el rol tomado por el ~~learner~~ varia. Es posible distinguir entre ~~learner~~ activo y pasivo. Un ~~learner~~ activo interactua con el ambiente durante el entrenamiento, realizando consultas o experimentos, mientras que un ~~learner~~ pasivo sólo observa la información provista por el ambiente (o el profesor) sin influenciarla o dirigirla.

Ayuda del profesor Cuando se piensa en el aprendizaje humano, el proceso por lo general involucra a un ~~helpful~~ profesor, que está tratando de proveer al ~~learner~~ con la información más útil para lograr el objetivo de aprendizaje. En contraste, cuando un científico aprender sobra la naturaleza, el ambiente,

tomando el rol de profesor, puede ser pensado como pasivo - las manzanas caen, las estrellas brillan y la lluvia cae sin cuidado por la necesidad del ~~learner~~. Dichos escenarios son modelados postulando que los datos de aprendizaje (o la experiencia del ~~learner~~) son generados por un proceso aleatoria.

Protocolo de entrenamiento lineal (o en línea)/~~batch~~ El último parámetro a mencionar es la distinción entre las situaciones en las que el ~~learner~~ tiene que responer de forma líneal, es decir que para cada ejemplo de entrenamiento tiene que dar una respuesta, y cuando el ~~learner~~ tiene que mostrar el aprendizaje adquirido después de tener la oportunidad de procesar grandes cantidades de datos.

1.2. Perceptron

En 1943, Warren McCullock y Walter Pitts publican la primera aproximación de una neurona simplificada, tratando de entender cómo funciona el cerebro biológico oara el dieseño de inteligencia artificial, la llamada neurona McCullock-Pits (MCP) [**?**, see p10].

Las neuronas son células nerviosas interconectadas del cerebro que están involucradas en el procesamiento y la transimición de señales químicas y eléctricas, como en la siguiente figura: [**?**, e.g. page 300] [**?**]

IMÁGEN

McCullock y Pitts describen a la neurona como una compuerta lógica sencilla con una salida binaria; múltiples señales llegan a las dendritas para ser integradas al cuerpo de la célula. Si la señal acumulada excede cierto umbral, se genera una señal de salida que se le pasa al axón.

Unos años después, Frank Rosenblatt publica la primera aproximación al concepto de perceptron basado en el modelo de neuronas MCP. FALTA REF. Intuitivamente, el algoritmo aprende automáticamente los coeficientes de pesos óptimos que luego se multiplican con las características de entrada para tomar la decisión de si la neurona se activa o no. Este algoritmo podría ser usado entonces para predecir si una muestra pertenece a una clase o a otra.

Formalmente, podemos plantearlo como un problema de clasificación binaria, donde nos referimos a nuestras dos clases, por simplicidad, como 1 (clase positiva) y -1 (clase negativa). Definimos también una *función de activación* $\phi(\mathbf{z})$ que toma una combinación lineal de ciertos valores de entrada $\mathbf{x}$ y un vector de pesos $\mathbf{w}$, donde $\mathbf{z}$ es la llamada *entrada de la red* ($z = w_1x_1 + ... + w_mx_m$):

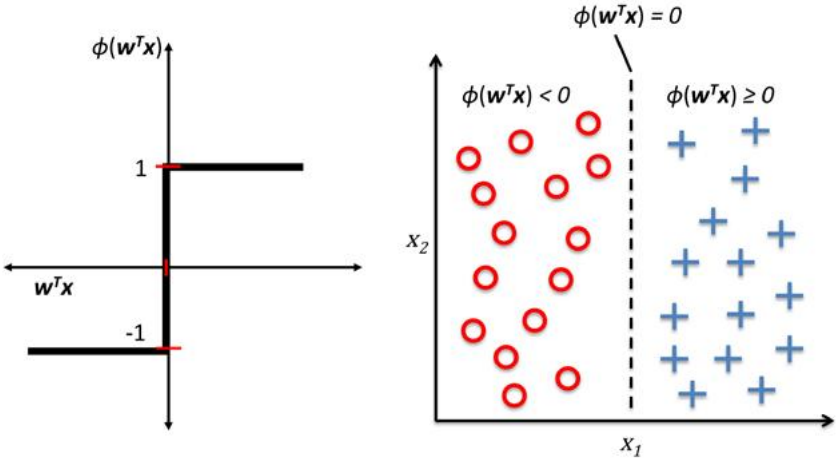
$$w = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, x = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

Si la activación de una muestra particular $x^{(i)}$ es mayor que un parámetro definido θ, predecimos la clase 1, y la clase -1 en caso contrario. En el algoritmo del perceptrón, la activación de función $\phi(\cdot)$ es una *función escalón*, que es llamada

a veces la *función de Heaviside*:

$$\phi(z) = \begin{cases} 1 & \text{si } z \geq \theta \\ -1 & \text{en otro caso} \end{cases}$$

Por simplicidad, definimios $w_0 = -\theta$ y $x_0 = 1$, escribiendo entonces a z de la forma $z = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m = \mathbf{w^T x}$. La siguiente imágen ilustra cómo la entrada de la red $z = w^T x$ es *aplanada* a una salida binaria (-1 o 1) por la función de activación del perceptrón (izquierda) y cómo puede ser usada para discriminar entre dos clases linealmente separables (derecha):



La idea detrás del modelo de perceptron de Rosenblatt es reducir a una abstracción de cómo funciona una neurona: se activa o no se activa. Así, la regla inicial de Roseblatt es relativamente simple y puede ser resumida en los siguientes pasos:

1. Inicializar los pesos en cero o en números aleatorios cercanos a cero.
2. Para cada muestra de entrenamiento $x^{(i)}$ realizar los siguientes pasos:

1. Caluclar el valor de salida $\hat{y}$. 2. Actualizar los pesos.

En este caso, el valor de salida es la clasificación dada por la función escalón definida previamente, y la actualización simultánea de cada peso w_j en el vector de pesos w puede ser escrito más formalmente cómo:

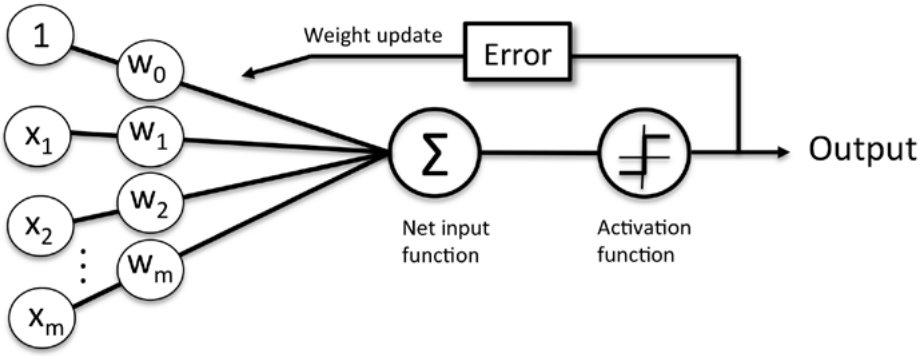
$$w_j := w_j + \Delta w_j \tag{1.1}$$

El valor de Δw_j, que es usado para actualizar el peso w_j, es calculado por la regla de aprendizaje de percetron:

$$\Delta w_j = \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)} \tag{1.2}$$

Donde η es el índice de aprendizaje (una constante entre 0 y 1), $y^{(i)}$ es la clasificación real de la i-ésima muestra, y $\hat{y}^{(i)}$ es la clasificación dada por la predicción. Es importante resaltar que todos los pesos en el vector de pesos son actualizados de manera simultánea, lo que significa que no recalculamos $\hat{y}^{(i)}$ hasta que todos los pesos Δw_j han sido actualizados.

Podemos resumir el concepto general de perceptron con la siguiente imágen:



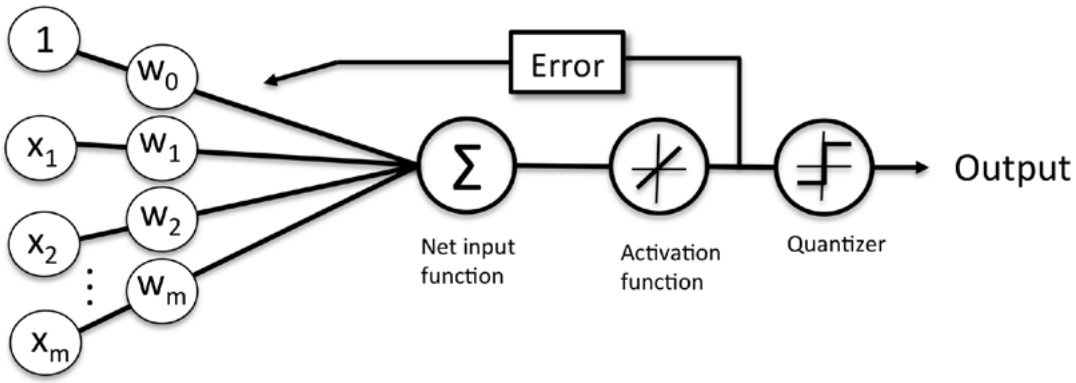
La imágen anterior muestra cómo el perceptron recibe las entradas de una muestra x y las combina con los pesos w para calcular la entrada neta. Esta entrada se le dá a la función de activación, que genera una salida binaria (-1 o 1 en este caso), representando la predicción de la clasifica- ción. Durante la fase de aprendizaje, la salida es usada para calcular el error de la predicción y actualizar los pesos.

1.3. Neuronas adaptativas lineales y la convergencia del entrenamiento

La neurona adaptativa lineal (**Adaline**) fué publicada unos años después del algoritmo del perceptrón de Frank Rosenblatt, por Bernard Widrow y es considerada un ¿improvement' sobre el perceptrón. El algoritmo de Adaline es particularmente interesante porque ilustra el concepto clave de definir y minimizar funciones de costo, lo que sienta las bases para algoritmos mas avanzados de clasificación, como la regresión logística o las máquinas de vectores de apoyo.

La principal diferencia entre las reglas de Adaline (también llamada de *Widrow-Hoff*) y la del perceptrón de Rosenblatt es que los pesos son actualizados con base en una función de activación lineal, en lugar de una función escalón. En Adaline, esta función de activación $\phi(z)$ es simplemente la función identidad de la entrada neta, así $\phi(w^T x) = w^T x$.

Mientras que la función de activación lineal es usada para la actualiza- ción de pesos, un *cuantificador*, similar a la función escalón descrita anteriormente, puede ser usado para hacer la clasificación, como se ilustra en la siguiente imágen:



Si se compara la imágen anterior con la ilustración del algoritmo del perceptrón, la diferencia es que se usa la salida con valores continuos de la función lineal de activación para calcular el error y actualizar los pesos, en lugar de hacer una clasificación binaria.

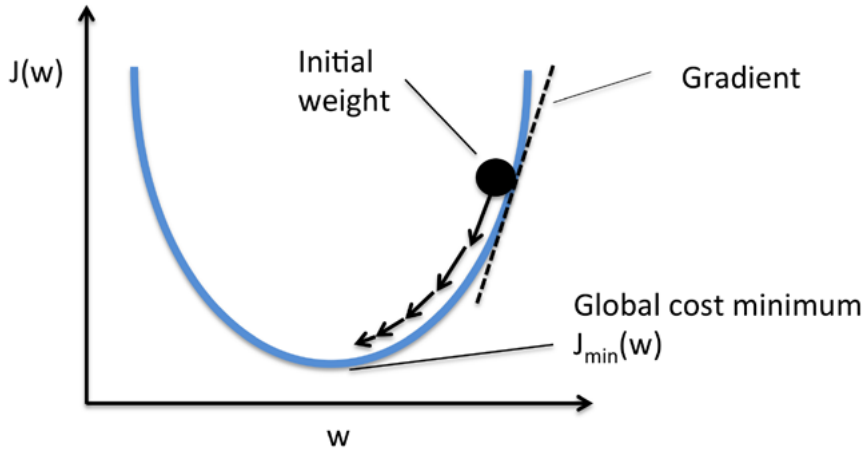
1.4. Gradient descent

Uno de los puntos clave del aprendizaje de máquina supervisado es definir una función objetivo que deberá ser optimizada durante el proceso de aprendizaje. Esta función objetivo usualmente es una *función de costo* que se quiere minimizar. En el caso de Adaline, se define la función de costos J como la **Suma de los errores cuadrados (SSE)** entre la salida calculada y el valor real.

$$J(w) = 1/2 \sum_i (y^{(i)} - \phi(z^{(i)}))^2 \tag{1.3}$$

La principal ventaja de esta función lineal de activación es que, en contraste con la función escañón, la función de costos se vuelve diferenciable. Otra propiedad de esta función de costos es que es convexa. Así, es posible usar un algoritmo llamado *gradient descent* para encontrar los pesos que minimizan la función de costos, maximizando así la ¿accuracy? de las predicciones.

La idea detrás del gradient descent es disminuir el gradiente hasta encontrar un mínimo (local o global) de la función de costo. En cada iteración, se reduce un *paso* del gradiente, donde cada *paso* está determinado por el valor del índice de aprendizaje, así como por la pendiente del gradiente.



Usando el gradient descent, se pueden actualizar los pesos quitandole un *paso* al gradiente $\nabla J(w)$ de la función de costos $J(w)$:

$$w := w + \Delta w \tag{1.4}$$

En donde el cambio de peso Δw se defino como el gradiente negativo multiplicado por el índice de aprendizaje η:

$$\Delta w := -\eta \Delta J(w) \tag{1.5}$$

Calculamos la derivada parcial de la función de costos SSE con respecto al j-ésimo peso de la siguiente manera:

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i (y^{(i)} - \phi(z^{(i)}))^2 \\
&= \frac{1}{2} \sum_i 2(y^{(i)} - \phi(z^{(i)})) \frac{\partial}{\partial w_j} (y^{(i)} - \phi(z^{(i)})) \\
&= \sum_i (y^{(i)} - \phi(z^{(i)})) \frac{\partial}{\partial w_j} (y^{(i)} - \phi(z^{(i)})) \\
&= \sum_i (y^{(i)} - \phi(z^{(i)}))(-x_j^{(i)}) \\
&= -\sum_i (y^{(i)} - \phi(z^{(i)})) x_j^{(i)}
\end{aligned}$$

Aunque la regla de aprendizaje de Adaline se ve idéntica a la del preceptrón, el término $\phi(z^{(i)})$ con $z^{(i)} = w^T x^{(i)}$ es un número real y no un número entero de clasificación. Más aún, la actualización de los pesos está basada en todas las muestras de entrenamiento (en lugar de actualizar de forma incremental después de cada muestra). Es por esto que a este acercamiento se le conoce como "batch" gradient descent.

1.5. Stochastic gradient descent

Si se considera el caso en que se tiene un ~~very large dataset~~ con millones de puntos con datos, correr un entrenamiento con GRADIENT BATCH DESCENT puede ser un proceso sumamente costoso computacionalmente ya que se requiere reevaluar todo el DATASET cada vez que se toma un *paso* hacia el mínimo global.

Una alternativa popular al algoritmo BATCH GRADIENT DESCENT es *STOCHASTIC GRADIENT DESCENT*, llamado también GRADIENT DESCENT *iterativo*. En lugar de actualizar los pesos basado en la suma de los errores acumulados de todas las muestras $x^{(i)}$:

$$\Delta w = \eta \sum_i (y^{(i)} - \phi(z^{(i)}))x^{(i)} \tag{1.6}$$

Se actualizan los datos de manera incremental para cada muestra del entrenamiento:

$$\eta(y^{(i)} - \phi(z^{(i)}))x^{(i)} \tag{1.7}$$

Aunque el STOCHASTIC GRADIENT DESCENT podria ser considerado una aproximación del GRADIENT DESCENT, por lo general converge mucho más rápido debido a las actualizaciones tan frecuentes de los pesos. Como cada gradiente se calcula basado en un sólo ejemplo de entrenamiento, ~~the error surface is noisier than in gradient descent, which can also have the advantage that stochastic gradient descent can escape shallow local minima more readily~~. Para obtener resultados ~~precisos~~ con ~~stochastic gradient descent~~ es importante que se tomen los datos de forma aleatoria.

Otra ventaja del ~~stochastic gradient descent~~ es que se puede usar para hacer *aprendizaje en línea*. Esto quiere decir que el modelo es entrenado ~~on the fly~~ al momento mientras más y más datos van llegando. Esto es especialmente útil cuando se están acumulando grandes cantidades de datos. Usando entrenamiento en línea, el sistema puede adaptarse inmediatamente a los cambios y los datos de entrenamiento pueden ser descartados después de actualizar el modelo, si el espacio de almacenamiento fuera un problema.