# Lecture 07:
# Intro a Deep Learning, Redes Neuronales
## Aprendizaje y Minería de Datos para los Negocios

Ignacio Sarmiento-Barbieri

Universidad de los Andes

November 5, 2021

# Agenda

1. Recap
2. Deep Learning: Intro
3. Multilayer Perceptrons
   - Worked Example
   - Minimalist Theory
     - Activation Functions
     - Output Functions
     - Architecture Design
4. Break
5. `R para ML`
   - Demo

# Deep Learning: Intro

- Word Embedding surgen de las redes neuronales
- Redes neuronales son modelos simples
- Su fuerza reside en la simplicidad porque patrones básicos hacen que sean fácil de procesar y entrenar.
- El modelo tiene combinaciones lineales que pasan a través de funciones no lineales de activación que se llaman nodos (o neuronas).
- Un conjunto de nodos que toma diferente sumas ponderadas de los mismos insumos se llama capa (layer)
- El resultado de los nodos de una capa se convierte en el insumo de la otra capa.

# Deep Learning: Recap

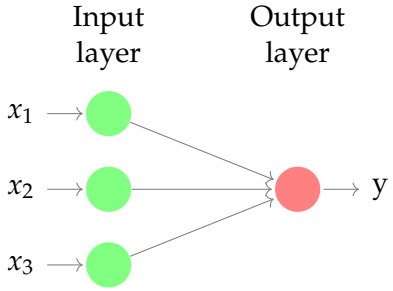
- Empecemos con un modelo simple y familiar, el modelo lineal

$$y = f(X) + u \tag{1}$$
$$y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + u$$

# Deep Learning: Recap

- Empecemos con un modelo simple y familiar, el modelo lineal

$$y = f(X) + u \tag{1}$$
$$y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + u$$



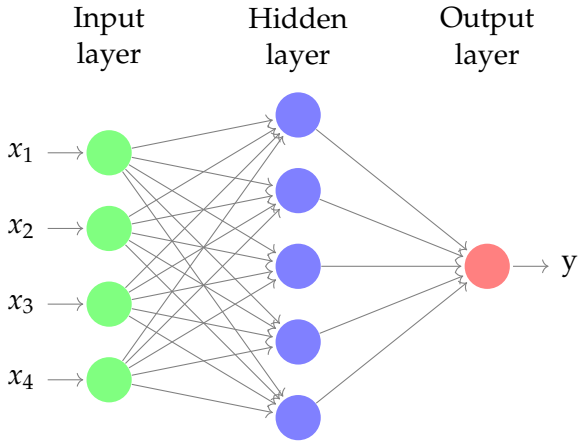
# Multilayer Perceptrons

- Los modelos lineales pueden ser muy sencillos y perder la no linealidad que mejor aproximan $f^*(x)$
- Podemos superar estas limitaciones de los modelos lineales y manejar una clase de funciones más generales al incorporar una o más capas ocultas.
- Las redes feedforward, también llamadas redes neuronales de alimentación directa, o perceptrones multicapa (MLP), son los modelos de aprendizaje profundo por excelencia

# Multilayer Perceptrons

- Las redes neuronales feedforward son llamadas redes porque generalmente se representan componiendo muchas funciones diferentes.
- El modelo se asocia con un gráfico acíclico dirigido que describe cómo las funciones se componen juntas. Por ejemplo, podemos tener dos funciones $f^{(2)}, f^{(1)}$ y conectadas en una cadena para formar $f(x) = f^{(2)}(f^{(1)}(x))$
- Estas estructuras de cadena son las estructuras de redes neuronales más utilizadas.

# Multilayer Perceptrons



# Multilayer Perceptrons

- La longitud total de la cadena le otorga la profundidad al modelo. El nombre “deep learning” surge de esta terminología.
- La capa final de una red feedforward network se denomina capa de salida
- Durante el entrenamiento de red neuronal, tratamos de entrenar $f(x)$ para estimar $f^*(x)$

# Multilayer Perceptrons

- En el entrenamiento de datos observamos la primer capa, inputs ($x$), y la última capa, output ($y$)
- No observamos las capas intermedias, llamadas capas ocultas.
- Finalmente, estas redes son llamadas neuronales porque son inspiradas por neurociencia.

# Worked Example: The "Exclusive OR (XOR)" Function

- La disyunción exclusiva de un par de proposiciones, (p, q), supone que p o q es verdadero, pero no ambos
- La tabla de la verdad es:

| q | p | q v p |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

- Cuando exactamente uno de estos valores binarios es igual a 1, la función XOR regresa 1. De otra forma, regresa 0

# Worked Example: The "Exclusive OR (XOR)" Function

- Usemos un modelo lineal

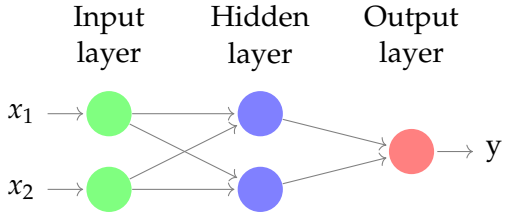
$$y = X\beta + \iota\alpha \tag{2}$$

$$y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} X = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \iota = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \tag{3}$$

- Solución $\alpha = \frac{1}{2}$ $\beta = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$
- Predicción $\hat{y} = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$

# Worked Example: The "Exclusive OR (XOR)" Function

- Usemos perceptroens multicapas (red feedforward) con una capa oculta conteniendo dos unidades ocultas




- Esta red tiene un vector de unidades ocultas $h$ que son calculadas por una función $f^{(1)}(x; W, c)$.
- Los valores de estas unidades ocultas se utilizan luego como entrada para una segunda capa.
- La segunda capa es la capa de salida de la red. La capa de salida sigue siendo sólo un modelo de regresión lineal, pero ahora se aplica a $h$ en vez de a $x$
- La red ahora contiene dos funciones encadenadas juntas, $f(x; W, c, w, b) = f^{(2)}(f^{(1)}(x))$

# Worked Example: The "Exclusive OR (XOR)" Function

- Qué $f^{(1)}$ deberíamos especificar?
  - Claramente **no** lineal, de lo contrario no tendria sentido la tarea
  - Vamos a usar una unidad lineal rectificada o ReLU (usualmente la recomendación predeterminada, hay muchas otras!!)
  - ReLU se define como $g(z) = max\{0, z\}$
- Para $f^{(2)}$? Para este ejemplo, un modelo lineal será suficiente

$$f^{(2)} = f^{(1)}w + b \tag{4}$$

- Entonces, el modelo final es

$$f(x, W, C, w, b) = max\{0, XW + c\}\, w + b \tag{5}$$

# Worked Example: The "Exclusive OR (XOR)" Function

- Suponga que esta es la solución al problema XOR

$$f(x) = max\{0, XW + c\}\, w + b$$

$$W = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$c = \begin{pmatrix} 0 & -1 \\ 0 & -1 \\ 0 & -1 \\ 0 & -1 \end{pmatrix}$$

$$w = \begin{pmatrix} 1 & -2 \end{pmatrix}$$

$$b = 0$$

# Worked Example: The "Exclusive OR (XOR)" Function

- Trabajemos el ejemplo paso a paso

$$f(x) = max\{0, XW + c\}\, w + b \tag{6}$$

$$XW = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{pmatrix}$$

$$XW + c = \begin{pmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

$$max\{0, XW + c\} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

# Worked Example: The "Exclusive OR (XOR)" Function

$$\hat{y} = max\{0, XW + c\}\, w + b = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 1 & -2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

- La red neuronal obtuvo la respuesta correcta para cada punto de los datos

# Worked Example: The "Exclusive OR (XOR)" Function

- En este ejemplo, simplemente especificamos la solución, luego se muestra que obtuvo cero error.
- En una situación real, puede haber miles de millones de parámetros y miles de millones de ejemplos de entrenamientos, por lo que simplemente no se puede adivinar la solución como se hizo aquí.
- En cambio, un algoritmo de optimización basado en gradientes puede encontrar parámetros que produce muy pocos errores.

# Multilayer Perceptrons: Theory

- Por qué no un modelo lineal?

$$\mathbf{H} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)} \tag{7}$$
$$\mathbf{Y} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$

- donde $\mathbf{X} \in \mathbb{R}^{n \times d}$, $n$ obs. y $d$ inputs (features).
- H es una capa oculta con $h$ unidades ocultas, $\mathbf{H} \in \mathbb{R}^{n \times h}$
- Debido a que las capas ocultas y de salida están completamente conectadas, tenemos
  - pesos de capa oculta $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times h}$ y sesgos $\mathbf{b}^{(1)} \in \mathbb{R}^{1 \times h}$
  - pesos de la capa de salida $\mathbf{W}^{(2)} \in \mathbb{R}^{h \times q}$ y sesgos $\mathbf{b}^{(2)} \in \mathbb{R}^{1 \times q}$

# Multilayer Perceptrons: Theory

- Por qué no un modelo lineal?
- Tenga en cuenta que después de agregar la capa oculta lineal, no ganamos nada por nuestros problemas!

$$\begin{aligned}\mathbf{Y} &= (\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)} \\ &= \mathbf{X}\mathbf{W}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)} \\ &= \mathbf{X}\mathbf{W} + \mathbf{b}.\end{aligned} \tag{8}$$

# Multilayer Perceptrons: Theory

- La ganancia viene de usar la función de activación no lineal $\sigma$
- Tener cuenta que, con la activación de la función, ya no es posible colapsar nuestro MLP en un modelo lineal:

$$\begin{aligned} \mathbf{H} &= \sigma(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}), \\ \mathbf{Y} &= \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}. \end{aligned} \tag{9}$$

- Tener en cuenta que podemos construir MLP más generales, apilando capas ocultas, produciendo modelos cada vez más expresivos.

$$\begin{aligned} \mathbf{H}^{(1)} &= \sigma_1(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}) \\ \mathbf{H}^{(2)} &= \sigma_2(\mathbf{H}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}) \\ \mathbf{Y} &= \mathbf{H}^2\mathbf{W}^{(3)} + \mathbf{b}^{(3)}. \end{aligned} \tag{10}$$

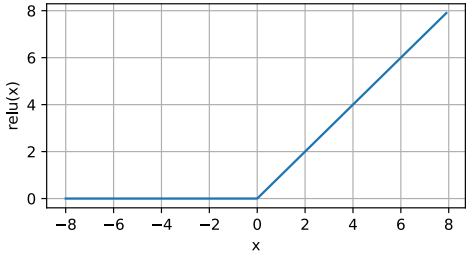
# Activation Functions

- Las funciones de activación son fundamentales para el aprendizaje profundo, analicemos brevemente algunas funciones de activación comunes.
- ReLU Function
  - La opción más popular, debido tanto a la simplicidad de implementación como a su buen desempeño en una variedad de tareas predictivas, es la unidad lineal rectificada. (ReLU).
  - ReLU proporciona una transformación no lineal muy simple. Dado un elemento $x$, la función es definida como el máximo de ese elemento y 0:

$$\text{ReLU}(x) = \max\{x, 0\}.$$

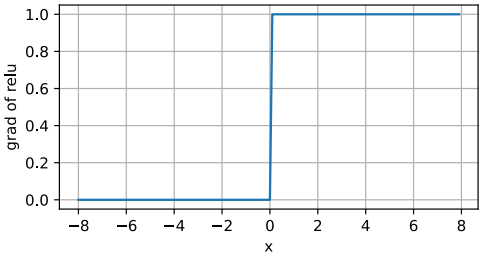
# Activation Functions

- La función ReLU retiene sólo los elementos positivos y descarta todos los elementos negativos estableciendo las activaciones correspondientes en 0.
- Es lineal por partes.



# Activation Functions

- Parte del atractivo de ReLU tiene que ver con el buen comportamiento de su derivada.
  - Tener en cuenta que la función ReLU no puede diferenciarse cuando la entrada toma un valor exactamente igual a 0. En estos casos, tomamos por defecto la derivada del lado izquierdo y decimos que la derivada es 0 cuando la entrada es 0 (incluso podemos salirse con la nuestra porque la la entrada nunca puede ser realmente cero!)
  - Esto hace que la optimización se comporte mejor y mitiga el problema de los gradientes que desaparecen.



# Activation Functions

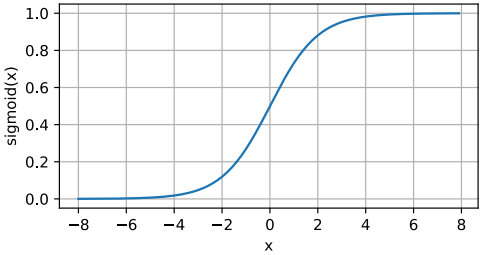
- Función sigmoidea
  - La función sigmoidea transforma sus entradas, cuyos valores se encuentran en el dominio $\mathbb{R}$, a salidas que se encuentran en el intervalo (0, 1).
  - Por esa razón, el sigmoide a menudo se denomina función de aplastamiento: aplasta cualquier entrada en el rango (-inf, inf) a algún valor en el rango (0, 1):

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}.$$

  - En las primeras redes neuronales, los científicos estaban interesados en modelar neuronas biológicas que se disparan o no. Así, los pioneros de este campo, desde McCulloch y Pitts, los inventores de la neurona artificial, se centraron en las unidades de umbral. Una activación de umbral toma el valor 0 cuando su entrada está por debajo de algún umbral y el valor 1 cuando la entrada excede el umbral.
  - Cuando la atención se centró en el aprendizaje basado en gradientes, la función sigmoidea fue una elección natural porque es una aproximación suave y diferenciable a una unidad de umbral.

# Activation Functions

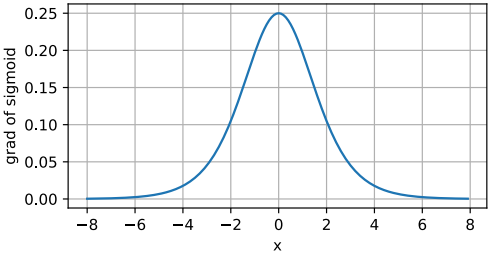
- Cuando la atención se centró en el aprendizaje basado en gradientes, la función sigmoidea fue una elección natural porque es una aproximación suave y diferenciable a una unidad de umbral.
- Función sigmoidea



# Activation Functions

- La derivada de la función sigmoidea está dada por la siguiente ecuación:

$$\frac{d}{dx}\,\text{sigmoid}(x) = \frac{\exp(-x)}{(1+\exp(-x))^2} = \text{sigmoid}(x)\,(1-\text{sigmoid}(x))\,.$$

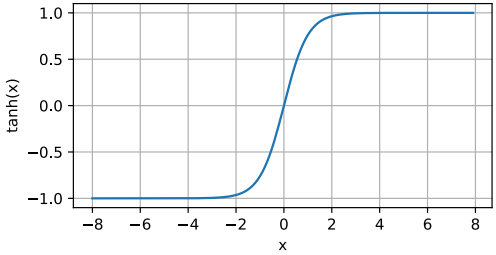


# Activation Functions

- Función *Tanh*
  - Como la función sigmoidea, la función tanh (tangente hiperbóloca) también aplasta sus entradas, transformándolas en elementos en el intervalo entre -1 y 1:

$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}.$$

  - Aunque la forma de la función es similar a la de la función sigmoidea, la función *tanh* exhibe simetría puntual sobre el origen del sistema de coordenadas.

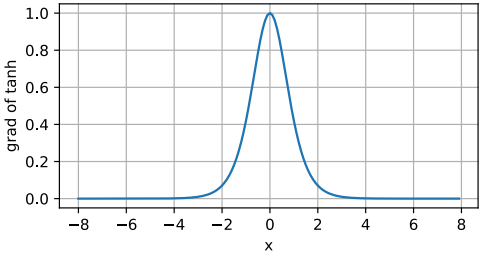


# Activation Functions

- La derivada de la función *tanh* es:

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x).$$

- La derivada de la función *tanh* está marcada abajo. A medida que la entrada se acerca 0, la derivada de la función *tanh* se acerca a un máximo de 1. Y como vimos con la función sigmoidea, a medida que la entrada se aleja de 0 en cualquier dirección, la derivada de la función *tanh* se acerca a 0.



# Activation Functions

- Otras funciones de activación
  - $h = cos(Wx + b)$ Goodfellow et all (2016) afirman que en el conjunto de datos MNIST obtuvieron una tasa de error de menos del 1 por ciento
  - La función de base radial (RBF): $exp\left(\frac{1}{\sigma^2)}||W - x||^2\right)$
  - Softplus: $log(1 + e^x)$
  - Hard tanh: $max(-1, min(1, x))$
- El diseño de unidades ocultas sigue siendo un área activa de investigación, y quedan muchos tipos de unidades ocultas útiles por descubrir.

# Output Functions

- La elección de la función de costo está estrechamente relacionada con la elección de la unidad de salida.
- La mayoría de las veces, simplemente usamos la distancia entre la distribución de datos y la distribución del modelo.
  - Linear $y = W'h + b \rightarrow \mathbb{R}$
  - Sigmoid (Logistic) $\frac{1}{1+\exp(-x)} \rightarrow$ clasificación $\{0, 1\}$
  - Softmax $\frac{exp(x)}{\sum exp(x))} \rightarrow$ clasificación de múltiples categorías

# Architecture Design

- Otra consideración de diseño clave para las redes neuronales es determinar la arquitectura.
- La palabra arquitectura se refiere a la estructura general de la red: cuántas unidades debería tener y cómo estas unidades deberían estar conectadas entre sí.
- En estas arquitecturas basadas en cadenas, las principales consideraciones arquitectónicas son elegir la profundidad de la red y el ancho de cada capa.
- Un perceptrón multicapa (redes feedforward) con capas ocultas proporcionan un marco de aproximación universal.
  - El teorema de aproximación universal (Hornik et al., 1989; Cybenko, 1989) establece que una red de retroalimentación con una capa de salida lineal y al menos una capa oculta con cualquier función de activación de ”aplastamiento” (como la función de activación sigmoidea logística) puede aproximar cualquier función medible de Borel de un espacio de dimensión finita a otro con cualquier valor distinto de cero cantidad de error, siempre que la red tenga suficiente unidad oculta.
  - La derivada de las redes feedforward también aproximar arbitrariamente bien las derivadas de las funciones (Hornik et al., 1990).

# Architecture Design

- El teorema de aproximación universal significa que independientemente de la función que estemos tratando de aprender, sabemos que un MLP grande podrá representar esta función.
- Sin embargo, no garantiza que el algoritmo de entrenamiento pueda aprender esa función. Incluso si el MLP puede representar la función, el aprendizaje puede fallar por dos razones diferentes.
  1. El algoritmo de optimización utilizado para el entrenamiento es posible que no pueda encontrar el valor de los parámetros que corresponde a la función deseada.
  2. El algoritmo de entrenamiento puede elegir la función incorrecta como resultado de un ajuste excesivo
- Una red de retroalimentación con una sola capa es suficiente para representar cualquier función, pero la capa puede ser inviable grande y puede fallar en aprender y generalizar correctamente.
- En muchas circunstancias, el uso de modelos más profundos puede reducir el número de unidades necesarias para representar la función deseada y puede reducir la cantidad de error de generalización.
- La arquitectura de red ideal para una tarea se debe encontrar a través de la

# Volvemos en 15 mins con R

# R para ML


photo from https://www.dailydot.com/parsec/batman-1966-labels-tumblr-twitter-vine/

# Deep Learning: Demo

```
library(keras)
fashion_mnist <- dataset_fashion_mnist()
```

# Deep Learning: Demo

```
c(train_images, train_labels) %<-% fashion_mnist$train
c(test_images, test_labels) %<-% fashion_mnist$test

class_names = c('T-shirt/top',
                'Trouser',
                'Pullover',
                'Dress',
                'Coat',
                'Sandal',
                'Shirt',
                'Sneaker',
                'Bag',
                'Ankle boot')
```

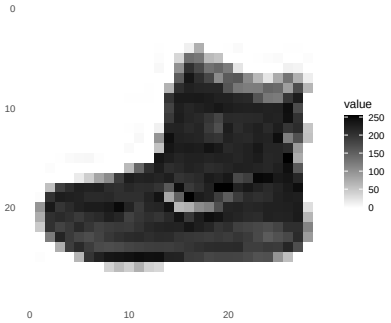
# Deep Learning: Demo

```
library(tidyr)
library(ggplot2)

image_1 <- as.data.frame(train_images[1, , ])
colnames(image_1) <- seq_len(ncol(image_1))
image_1$y <- seq_len(nrow(image_1))
image_1 <- gather(image_1, "x", "value", -y)
image_1$x <- as.integer(image_1$x)

ggplot(image_1, aes(x = x, y = y, fill = value)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "black", na.value = NA) +
  scale_y_reverse() +
  theme_minimal() +
  theme(panel.grid = element_blank())   +
  theme(aspect.ratio = 1) +
  xlab("") +
  ylab("")
```

# Deep Learning: Demo

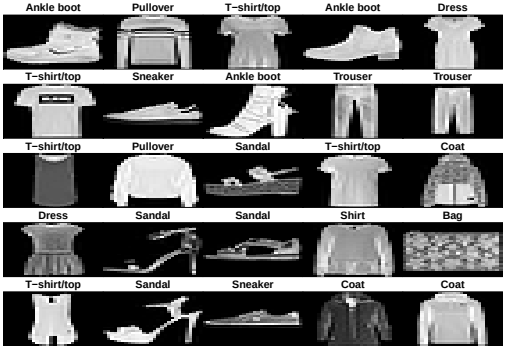


# Deep Learning: Demo

```
train_images <- train_images / 255
test_images <- test_images / 255
```

```
par(mfcol=c(5,5))
par(mar=c(0, 0, 1.5, 0), xaxs='i', yaxs='i')
for (i in 1:25) {
  img <- train_images[i, , ]
  img <- t(apply(img, 2, rev))
  image(1:28, 1:28, img, col = gray((0:255)/255), xaxt = 'n', yaxt = 'n',
        main = paste(class_names[train_labels[i] + 1]))
}
```

# Deep Learning: Demo



# Deep Learning: Demo

```
model <- keras_model_sequential()
model %>%
  layer_flatten(input_shape = c(28, 28)) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')
```

```
model %>% compile(
  optimizer = 'adam',
  loss = 'sparse_categorical_crossentropy',
  metrics = c('accuracy')
)
model %>% fit(train_images, train_labels, epochs = 5, verbose = 2)
```

# Deep Learning: Demo

```
## Epoch 1/5
## 1875/1875 - 2s - loss: 0.5003 - accuracy: 0.8238
## Epoch 2/5
## 1875/1875 - 2s - loss: 0.3782 - accuracy: 0.8643
## Epoch 3/5
## 1875/1875 - 2s - loss: 0.3362 - accuracy: 0.8784
## Epoch 4/5
## 1875/1875 - 2s - loss: 0.3141 - accuracy: 0.8844
## Epoch 5/5
## 1875/1875 - 2s - loss: 0.2934 - accuracy: 0.8922
```

```
score <- model %>% evaluate(test_images, test_labels, verbose = 0)

cat('Test loss:', score[1], "\n")
```

```
## Test loss: 0.3377942

## Test accuracy: 0.8792
```