Taller de Aprendizaje Automático

Actividades Taller 5

Instituto de Ingeniería Eléctrica
Facultad de Ingeniería

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Montevideo, 2024

Tabla de contenido

Objetivos del Taller 5

- Trabajar con modelos *Multilayer Perceptron* (MLP) utilizando la biblioteca *Keras*.
- Familiarizarse con algunas herramientas de búsqueda de hiperparámetros.
  - KerasTuner
  - Optuna

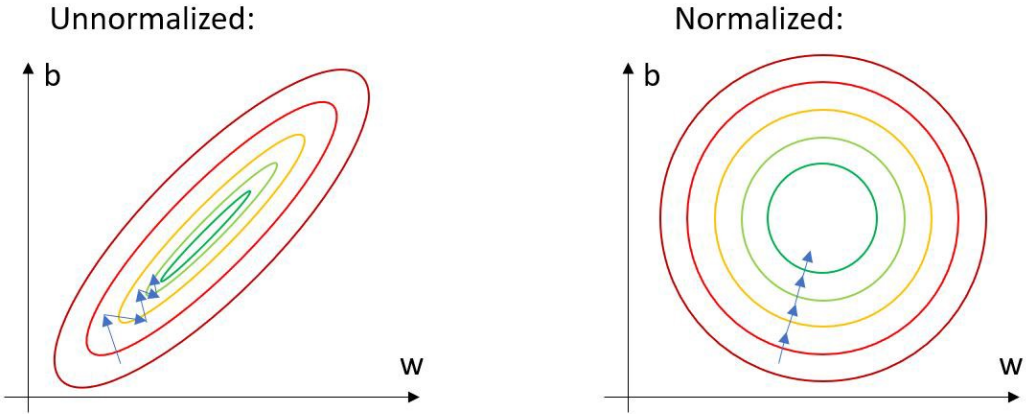
El problema y los datos

- Se trabajará con los mismos datos que se utilizaron en el Taller 3

Para hacer ahora

- **Parte 1**: Repensar el *pipeline* de preprocesamiento básico utilizado en el Taller 3
- **Parte 2**: Entrenar un modelo utilizando la arquitectura del ejemplo del Capítulo 10 del libro.
  - Ver sección *Building a Regression MLP Using the Sequential API* o repositorio del libro
  - ¿Cuántos parámetros tiene la red?
  - ¿Cuál es el desempeño? ¿Cómo lo mejoraría?
- **Parte 3**: Optimizar la arquitectura para este problema en particular usando las herramientas de búsqueda de hiperparámetros de *KerasTuner*
  - Ver ejemplos en sección *Fine-Tuning Neural Networks Hyperparameters*
  - Probar el *tip* que se sugiere en la sección *Number of Neurons per Hidden Layer*

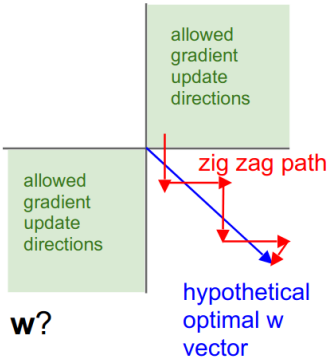
¿Por qué normalizar o estandarizar?



¿Por qué normalizar o estandarizar?

Consider what happens when the input to a neuron is always positive...

$$f\left(\sum_i w_i x_i + b\right)$$



What can we say about the gradients on **w**?
Always all positive or all negative :(
(this is also why you want zero-mean data!)

Figure: Figura extraída de slides del curso cs231n-Stanford

Parte 2

```
import tensorflow as tf
from tensorflow import keras

tf.random.set_seed(42)
norm_layer = tf.keras.layers.Normalization(input_shape=X_train.shape[1:])
model = tf.keras.Sequential([
    norm_layer,
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(1)
])


optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
model.compile(loss="mean_squared_logarithmic_error", optimizer=optimizer)

history = model.fit(X_train, y_train, epochs=20, validation_data=(X_valid,y_valid))
```

Parte 2

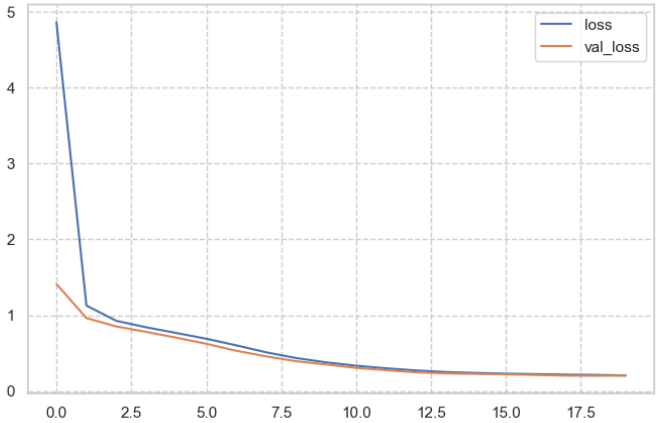


Figure: Evolución de la función de costo con parámetros por defecto

Búsqueda de hiperparámetros con KerasTuner

```python
import tensorflow as tf
import tensorflow.keras as keras
import keras_tuner as kt

def build_model(hp):
    n_hidden = hp.Int("n_hidden", min_value=0, max_value=8, default=2)
    n_neurons = hp.Int("n_neurons", min_value=16, max_value=256)
    learning_rate = hp.Float("learning_rate", min_value=1e-4, max_value=1e-2, sampling="log")
    optimizer = hp.Choice("optimizer", values=["sgd", "adam"])
    if optimizer == "sgd":
        optimizer = tf.keras.optimizers.SGD(learning_rate=learning_rate)
    else:
        optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model = tf.keras.Sequential()
    for _ in range(n_hidden):
        model.add(tf.keras.layers.Dense(n_neurons, activation="relu"))
    model.add(tf.keras.layers.Dense(1))
    model.compile(loss="mean_squared_logarithmic_error", optimizer=optimizer)
    return model
```

Búsqueda de hiperparámetros con KerasTuner

```
random_search_tuner = kt.RandomSearch(
    build_model, objective="val_loss", max_trials=5, overwrite=True,
    directory="my_bike_sharing_demand", project_name="my_rnd_search", seed=42)

random_search_tuner.search(X_train, y_train, epochs=10,
    validation_data=(X_valid, y_valid))
```

Shapley Values

- El valor de Shapley es la contribución del valor de una característica a la diferencia entre la predicción real y la predicción media del modelo.
- Una explicación más detallada se encuentra en el libro Interpretable Machine Learning de Christoph Molnar.
- La librería SHAP de python permite hacer el cálculo de los Shap Values.

Ejeplo Shapley Values