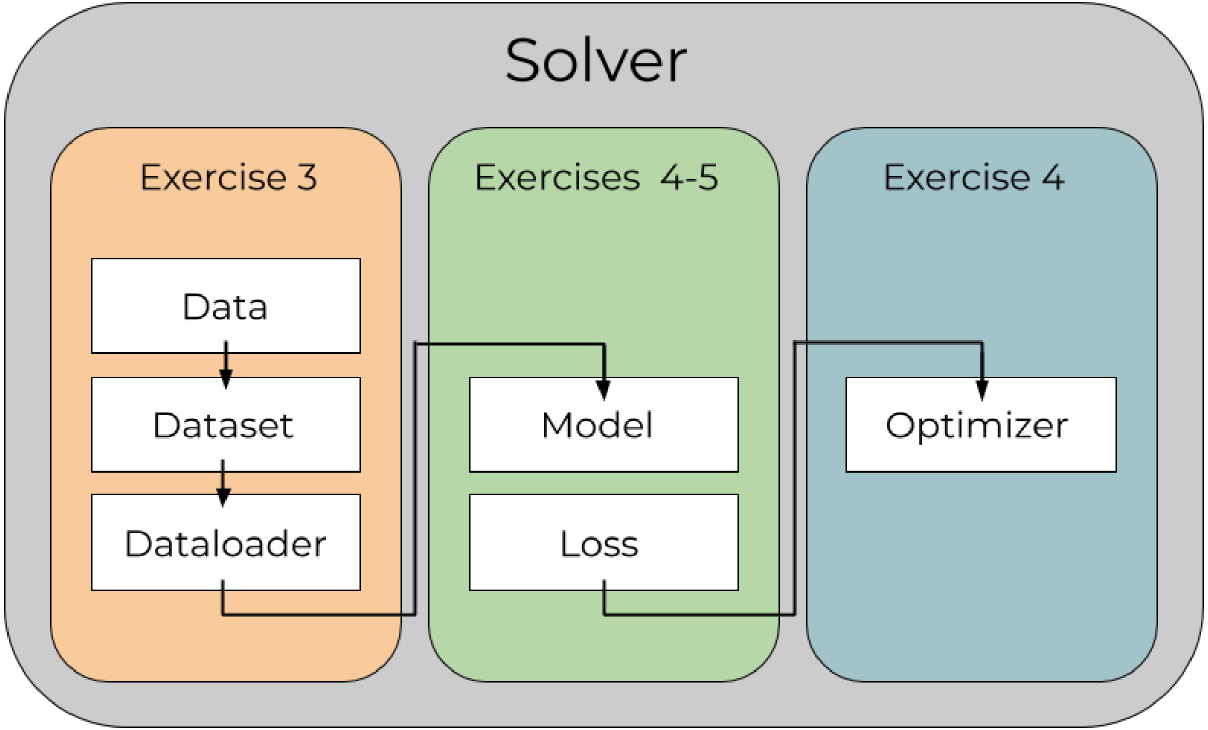
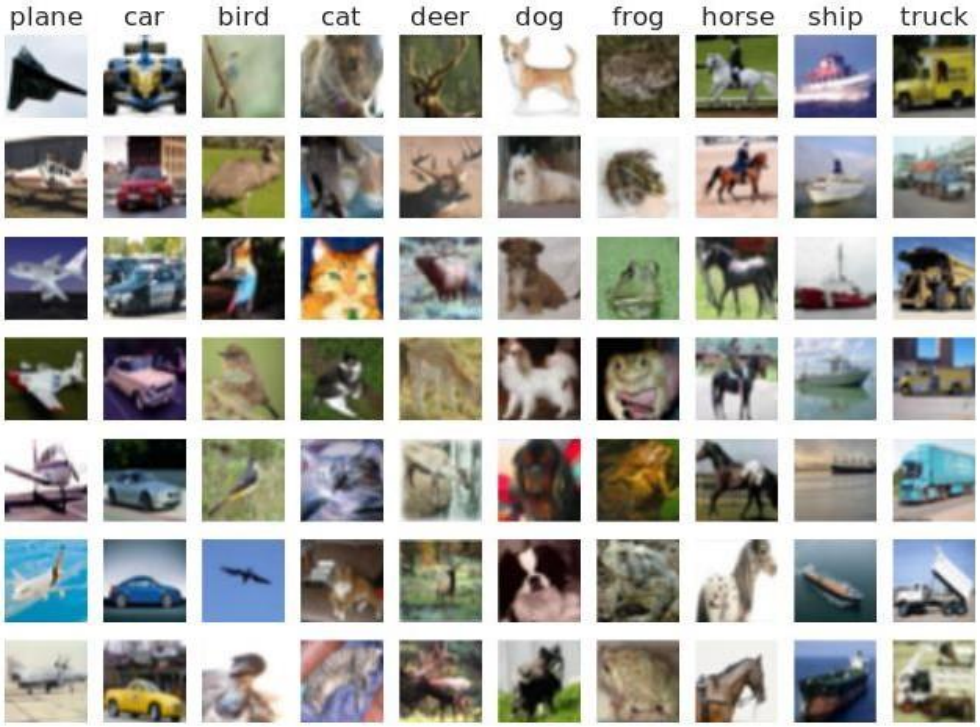
Ejercicio 6:
Hyperparameter Tuning

Repaso: Pillars of Deep Learning



Objetivo del Ejercicio 6



Cifar10

Objetivo del Ejercicio 6

- Utilizar las implementaciones existentes
  - Aplicaciones revisadas de ejercicios anteriores
  - Le proporcionaremos implementaciones adicionales de todas las herramientas necesarias para ejecutar los métodos de ejemplo propuestos en la clase.

- Aprenda de las neural network debugging strategies and hyperparameter search

Leaderboard

- La **precisión** de tu modelo es lo único que cuenta.
  - Al menos **un 48%** para aprobar el examen
  - Habrá una **clasificación** de todos los estudiantes.

Leaderboard

The leaderboard shows for each exercise the highest scoring submission from each user. Only valid submissions are displayed.

Exercise 1 | Exercise 3 | Exercise 4 | Exercise 5 | Exercise 6 | Exercise 7 | Exercise 8 | Exercise 9 | Exercise 10 | Exercise 11

#	User	Score
1	a0008	100.00
2	a0001	100.00
3	a0003	100.00
4	u0306	100.00
5	u1540	100.00

Previously: Dataset

```python
class ImageFolderDataset(Dataset):
    """CIFAR-10 dataset class"""
    def __init__(self, transform=None, mode='train',
        limit_files=None,
        split={'train': 0.6, 'val': 0.2, 'test': 0.2},
        *args, **kwargs): ...

    @staticmethod
    def _find_classes(directory): ...

    def select_split(self, images, labels, mode): ...

    def make_dataset(self, directory, class_to_idx, mode): ...

    def __len__(self): ...

    @staticmethod
    def load_image_as_numpy(image_path): ...

    def __getitem__(self, index): ...
```

```python
# Create a train, validation and test dataset.
datasets = {}
for mode in ['train', 'val', 'test']:
    crt_dataset = ImageFolderDataset(
        mode=mode,
        root=cifar_root,
        download_url=download_url,
        transform=compose_transform,
        split={'train': 0.6, 'val': 0.2, 'test': 0.2}
    )
    datasets[mode] = crt_dataset
```

Previously: Data Loader

```python
class DataLoader:
    """
    Dataloader Class
    Defines an iterable batch-sampler over a given dataset
    """
    def __init__(self,
        dataset,
        batch_size=1,
        shuffle=False,
        drop_last=False): ...

    def __iter__(self): ...

    def __len__(self): ...
```

```python
# Create a dataloader for each split.
dataloaders = {}
for mode in ['train', 'val', 'test']:
    crt_dataloader = DataLoader(
        dataset=datasets[mode],
        batch_size=256,
        shuffle=True,
        drop_last=True,
    )
    dataloaders[mode] = crt_dataloader
```

Previously: Solver

```python
class Solver(object):
    """
    A Solver encapsulates all the logic necessary for training classification
    or regression models.
    The Solver performs gradient descent using the given learning rate.
    """

    def __init__(self, model, train_dataloader, val_dataloader,
        loss_func=CrossEntropyFromLogits(), learning_rate=1e-3,
        optimizer=Adam, verbose=True, print_every=1,
        lr_decay = 1.0, **kwargs): ...

    def _reset(self): ...

    def _step(self, X, y, validation=False): ...

    def train(self, epochs=100, patience = None): ...

    def get_dataset_accuracy(self, loader): ...

    def update_best_loss(self, val_loss, train_loss): ...
```

```python
solver = Solver(model,
                dataloaders['train'],
                dataloaders['val'],
                learning_rate=0.001,
                loss_func=MSE(),
                optimizer=SGD)

solver.train(epochs=epochs)
```

Previously: Classification Network

```python
class ClassificationNet(Network):
    """
    A fully-connected classification neural network with configurable
    activation function, number of layers, number of classes, hidden size and
    regularization strength.
    """

    def __init__(self,
        activation=Sigmoid(), num_layer=2,
        input_size=3 * 32 * 32, hidden_size=100,
        std=1e-3, num_classes=10, reg=0, **kwargs): ...

    def forward(self, X): ...

    def backward(self, dy): ...

    def save_model(self): ...

    def get_dataset_prediction(self, loader): ...
```

```python
# Instantiate a new model.
model = ClassificationNet(activation=Sigmoid(),
                          num_layer=num_layer,
                          reg=reg,
                          num_classes=10)

# X is a batch of training features
# X.shape = (batch_size, features_size)
y_out = model.forward(X)

# dout is the gradient of the loss function
#   w.r.t the output of the network.
# dout.shape = (batch_size, )
model.backward(dout)
```

Previously: Binary Cross Entropy Loss

$$BCE\left(\hat{y}, y\right) = \frac{1}{N}\sum_{i=1}^{N}\left[-y_i \log\left(\hat{y}_i\right) - (1 - y_i)\log(1 - \hat{y}_i)\right]$$

Where

- N is the number of samples
- $\hat{y}_i$ **is the network's prediction for sample i**
- y_i **is the ground truth label (0 or 1)**

New: Multiclass Cross Entropy Loss

$$CE\left(\hat{y}, y\right) = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{C}\left[-y_{ik}\log\left(\hat{y}_{ik}\right)\right]$$

Implementamos esto por ustedes

Donde

- N es el número de muestras
- $\hat{y}_{ik}$ es la probabilidad predicha por la red para la k-ésima class dada la muestra i
- y_{ik} es la etiqueta de verdad que es 1 si la muestra i es de la clase k o cero en caso contrario.

Receta básica para el ML

- Split your data

60% 20% 20%

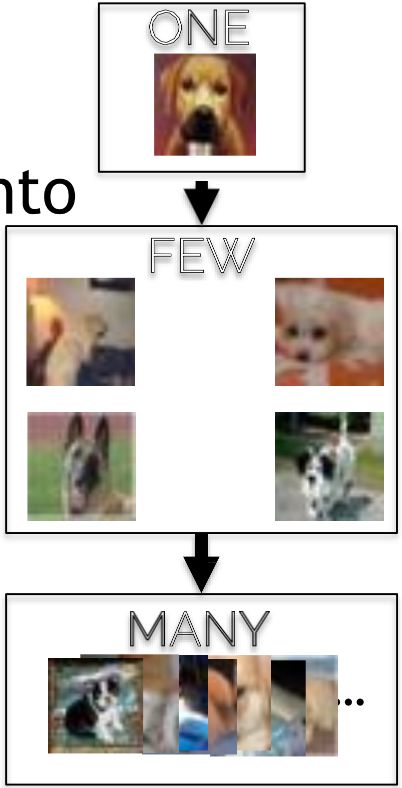
train | validation | test

Find your hyperparameters

Only used once at the end!

hidden test → Benchmarking: On our submission system

Como empezar



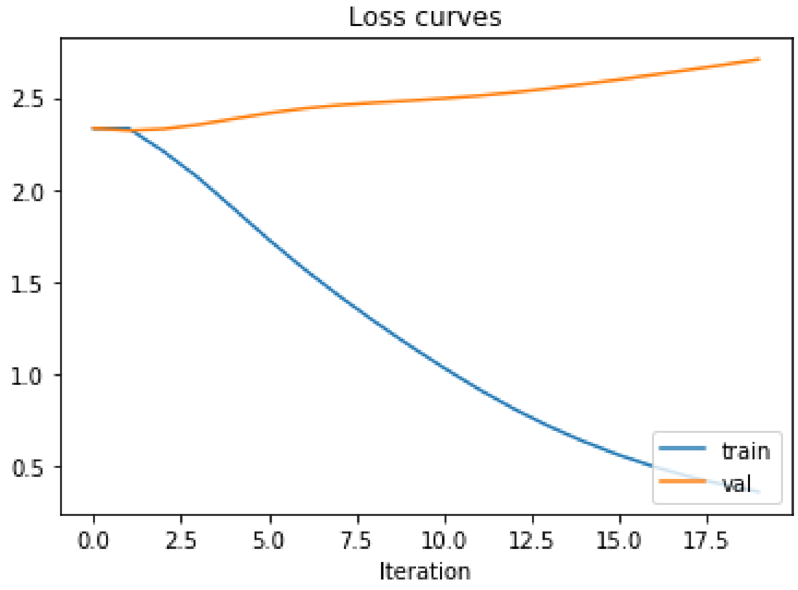
- Empezar con una sola muestra de entrenamiento
  - Comprobar si la salida es correcta
  - Overfit -> La precisión debe ser del 100% porque la entrada acaba de ser memorizada

- Incrementar a un puñado de muestras

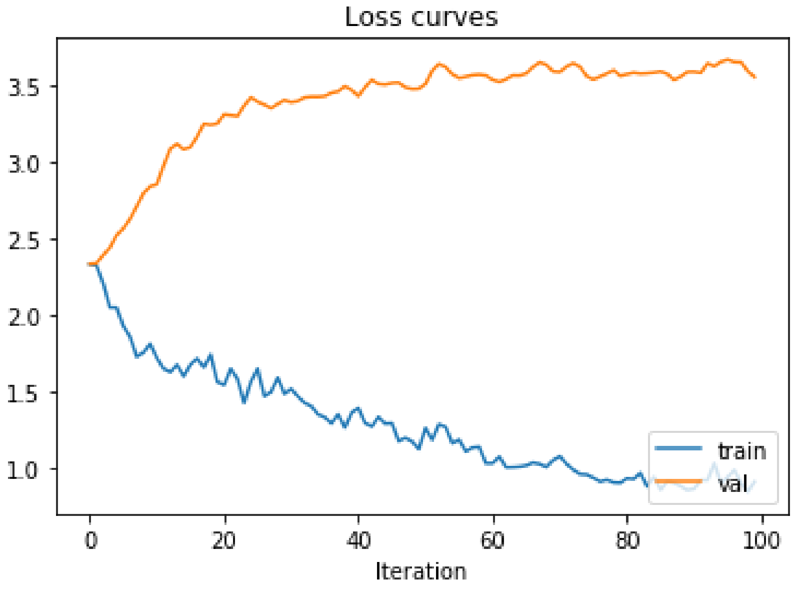
- Ir del overfitting a más muestras
  - En algún punto, tendrían que ver generalización

Como empezar

- Overfit a single training sample



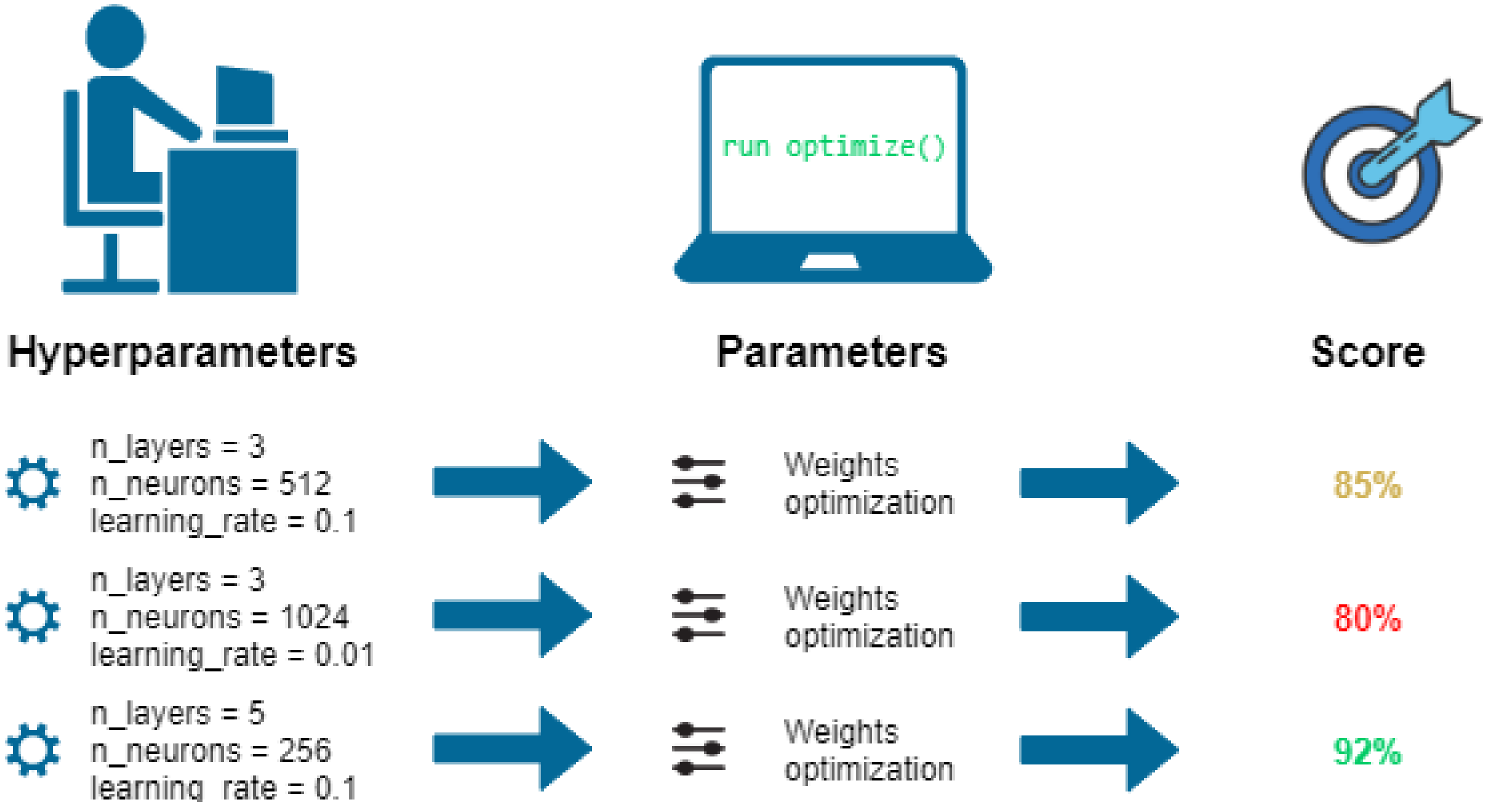
- Then a few samples



Hyperparameters

- Network architecture (e.g., num layers, hidden layer, activation function)
- Number of iterations
- Learning rate(s) (i.e., solver parameters, decay, etc.)
- Regularization (more later next lecture)
- Batch size
- …

Hyperparameter Tuning



Source: https://images.deepai.org/glossary-terms/05c646fe1676490aa0b8cab0732a02b2/hyperparams.png

¿Cómo encontrar buenos Hyperparameters?

- Manual Search (trial and error)
- Automated Search:
  - Grid Search
  - Random Search

```python
from exercise_code.hyperparameter_tuning import grid_search

best_model, results = grid_search(
    dataloaders['train_small'], dataloaders['val_500files'],
    grid_search_spaces = {
        "learning_rate": [1e-2, 1e-3, 1e-4, 1e-5, 1e-6],
        "reg": [1e-4, 1e-5, 1e-6]
    },
    epochs=10, patience=5,
    model_class=ClassificationNet)
```

- Piensen cómo diferentes hyperparametros pueden afectar al modelo
  - E.g. Overfitting? -> Increase Regularization Strength, decrease model capacity

Plan de la Práctica: Recap y Outlook

Exercise 03: Dataset and Dataloader Exercise 04: Solver and Linear Regression Exercise 05: Neural Networks Exercise 06: Hyperparameter Tuning	Numpy (Reinvent the wheel)
Exercise 07: Introduction to Pytorch Exercise 08: MNIST with Pytorch	Pytorch/Tensorboard
Exercise 09: Convolutional Neural Networks Exercise 10: Semantic Segmentation Exercise 11: Recurrent Neural Networks	Applications (Hands-off)

Nos vemos el próximo lunes ☺