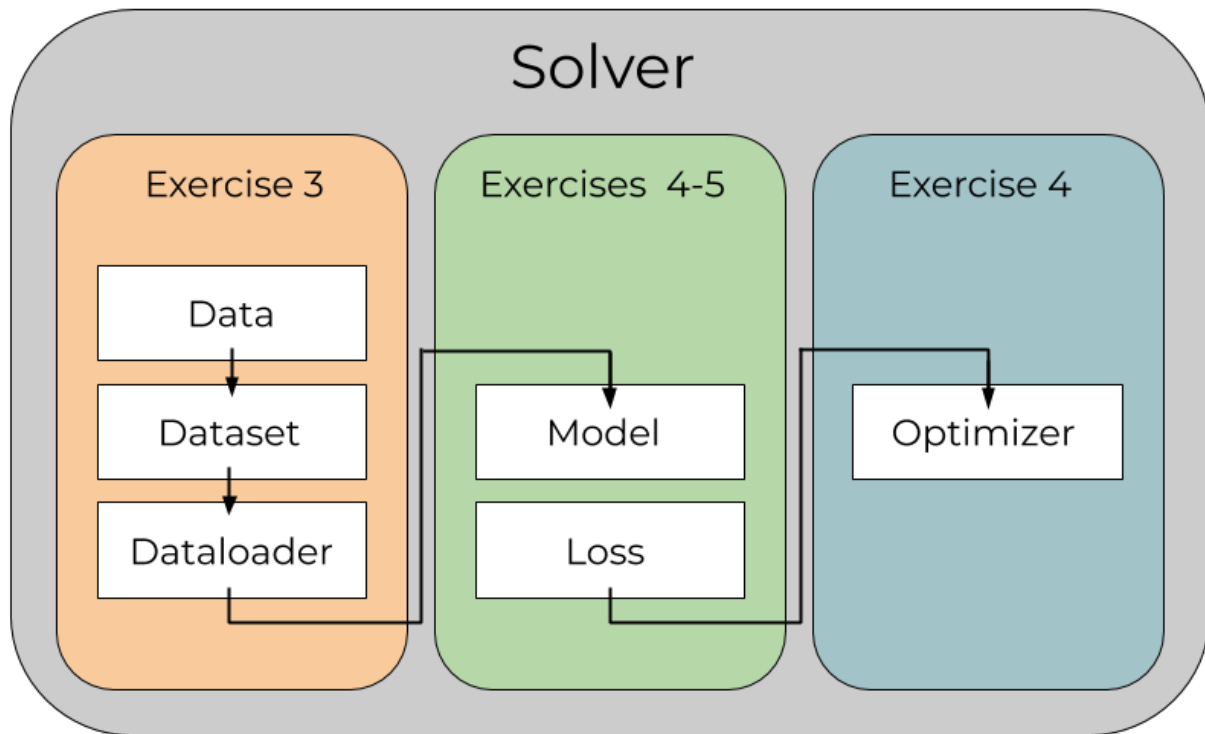


Introduction to Deep Learning (I2DL)

Exercise 6: Hyperparameter Tuning

Recap: Pillars of Deep Learning



Goal of exercise 6

- Use existing implementations
 - Reworked implementations of previous exercises
 - We will provide you with additional implementations of all required tools to run sample methods proposed in the lecture
- Learn about neural network debugging strategies and hyperparameter search



Previously: Dataset

```
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): ...
```

```
# 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

```
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): ...
```

```
# 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

```
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): """
```

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

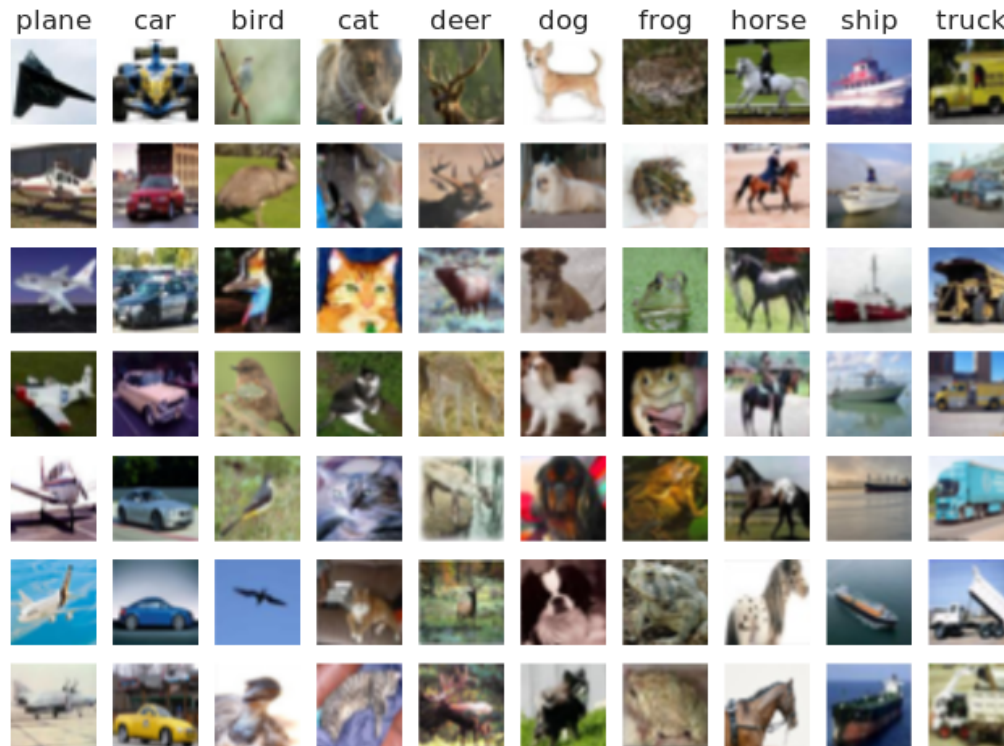
solver.train(epochs=epochs)
```

Previously: Classification Network

```
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):  
        """  
        """
```

```
# 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)
```

Submission Goal: Cifar10 Classification



Previously: Binary Crossentropy

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

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 Crossentropy

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

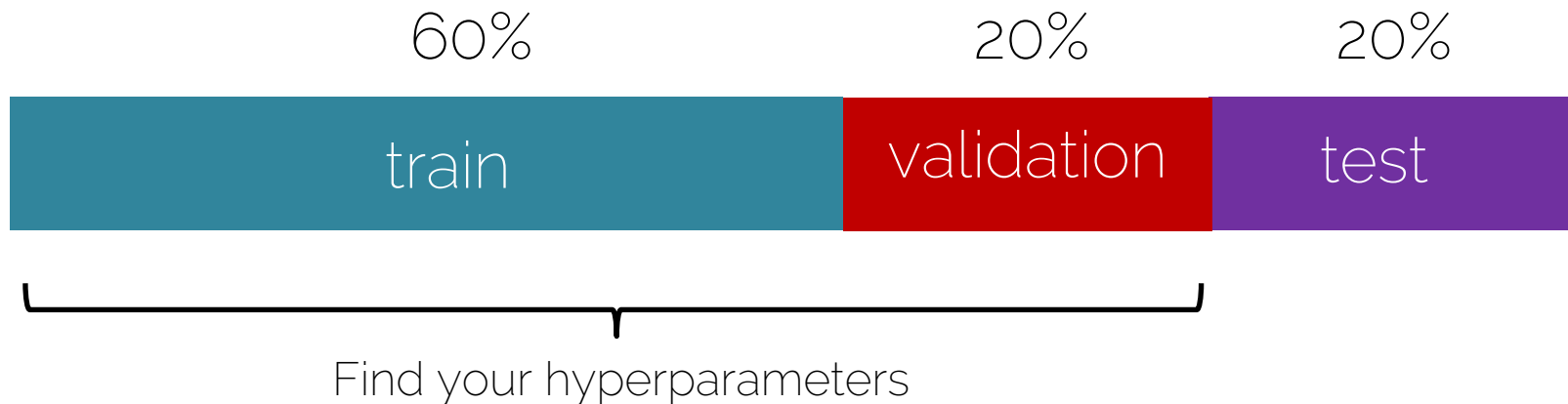
Where

- N is the number of samples
- \hat{y}_{ik} is the network's predicted probability for the k th class when given the sample i
- y_{ik} is the ground truth label which is either 1 if the i th sample is of class k or zero otherwise

We implemented this for you!
More on this topic in the next
lecture.

Basic Recipe for Machine Learning

- Split your data



Basic Recipe for Machine Learning

- Split your data



Example scenario

Ground truth error 1%

Training set error 5%

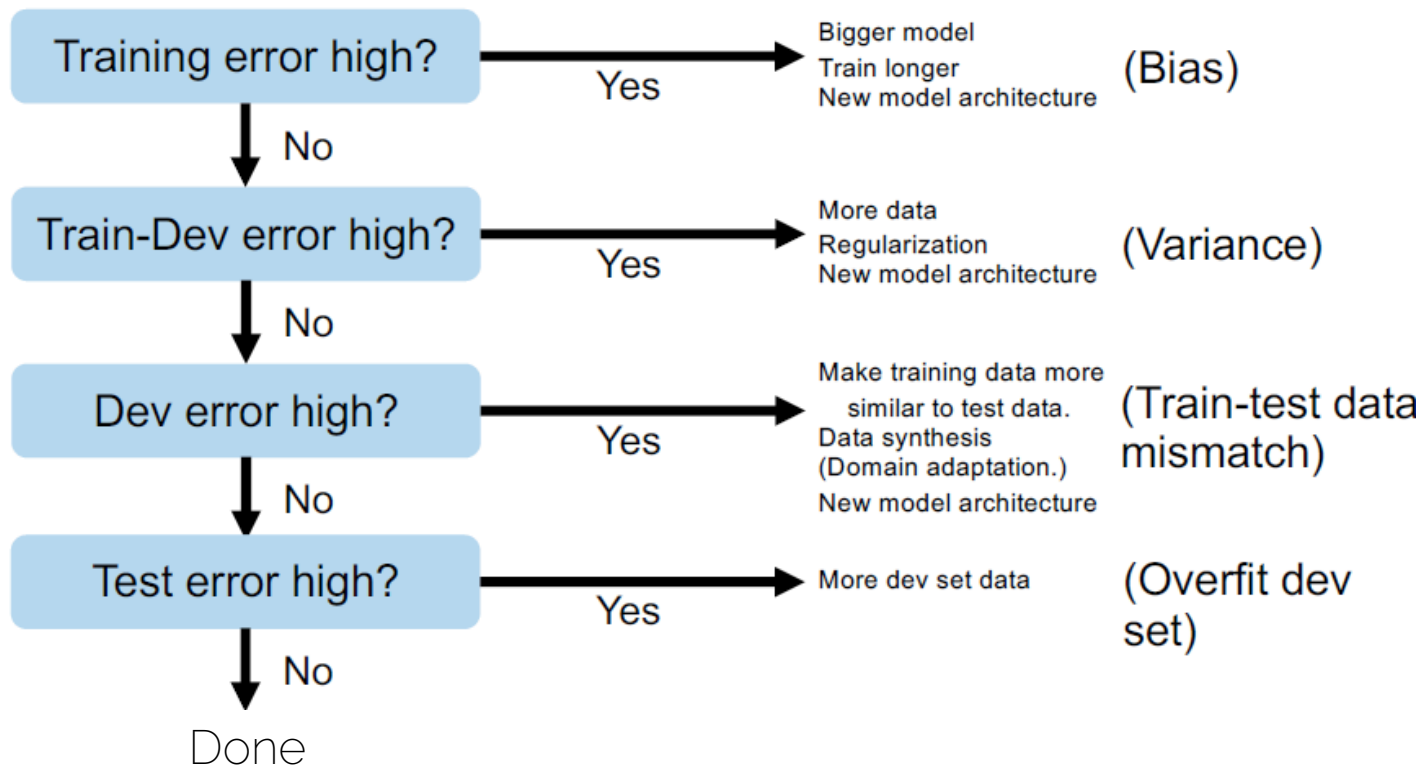
Val/test set error 8%



Bias
(underfitting)

Variance
(overfitting)

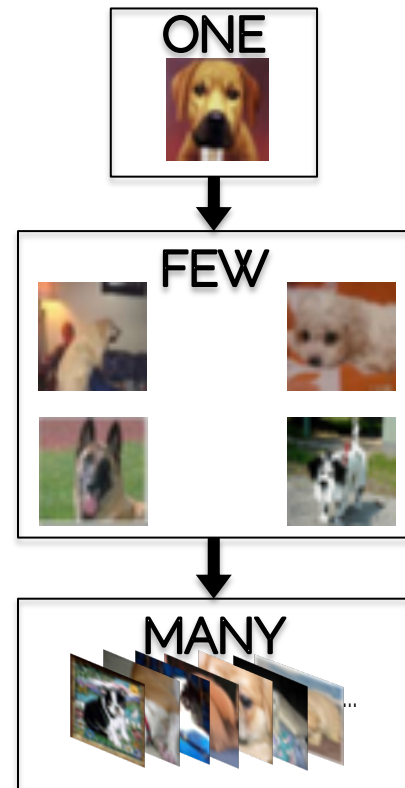
Basic Recipe for Machine Learning



Credits: A. Ng

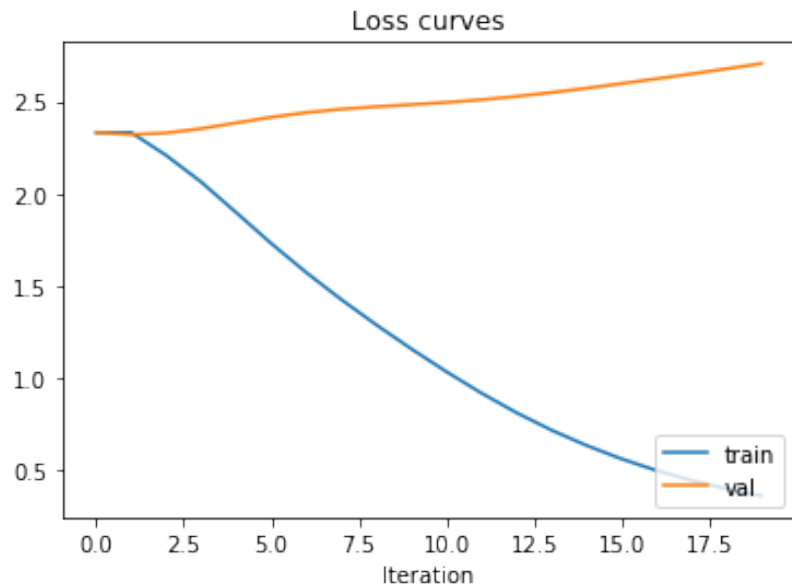
How to Start

- Start with single training sample
 - Check if output correct
 - Overfit → train accuracy should be 100% because input just memorized
- Increase to handful of samples
- Move from overfitting to more samples
 - At some point, you should see generalization

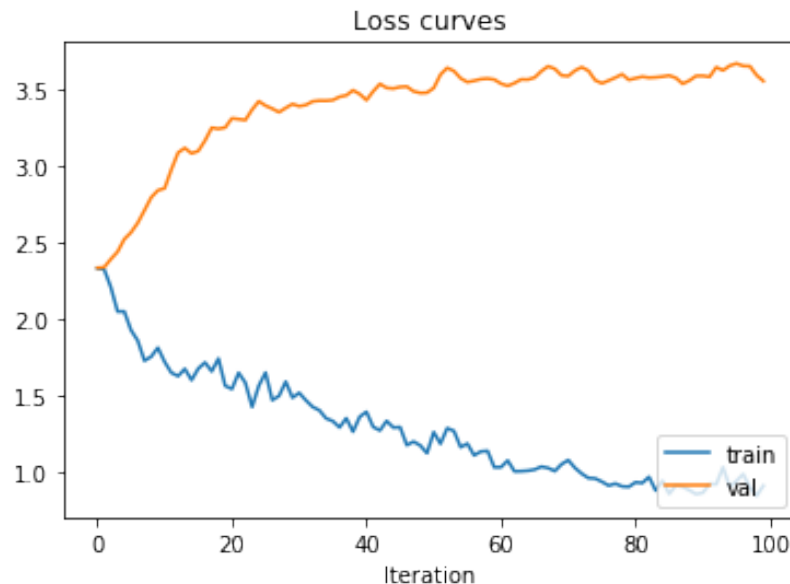


How to Start

- Overfit a single training sample



- Then a few samples



Hyperparameters

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

Hyperparameter Tuning

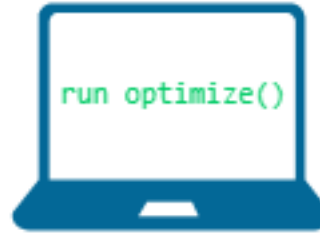


Hyperparameters

⚙️ n_layers = 3
n_neurons = 512
learning_rate = 0.1

⚙️ n_layers = 3
n_neurons = 1024
learning_rate = 0.01

⚙️ n_layers = 5
n_neurons = 256
learning_rate = 0.1



Parameters



Weights
optimization



Weights
optimization



Weights
optimization



Score

85%

80%

92%

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

How to find good Hyperparameters?

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

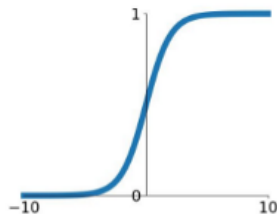
```
from exercise_code.hyperparameter_tuning import grid_search
best_model, results = grid_search(
    dataloaders['train_small'], dataloaders['val_500files'],
    grid_search_spaces = {
        "learning_rate": [1e-2, 1e-3, 1e-4, 1e-5, 1e-6],
        "reg": [1e-4, 1e-5, 1e-6]
    },
    epochs=10, patience=5,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

- Think about how different hyper parameters affect the model
 - E.g. Overfitting? -> Increase Regularization Strength, decrease model Capacity

Optional: Activation Functions

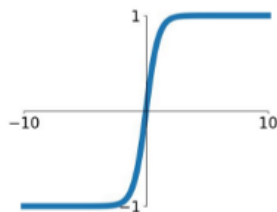
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



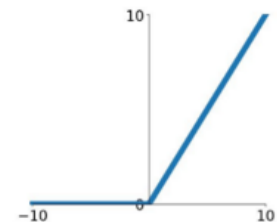
tanh

$$\tanh(x)$$



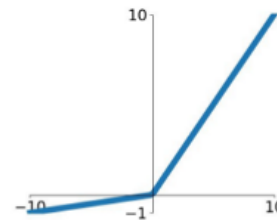
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

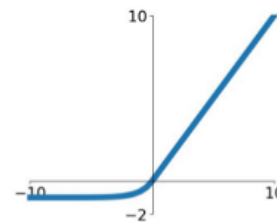


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Submission

- Submission Start: May 28, 2020 12.00
- Submission Deadline : June 03, 2020 23.59
- Your model's **accuracy** is all that counts!
 - At least **48%** to pass the submission
 - There will be a **leaderboard** of all students!

Leaderboard: Submission 6

Rank	User	Score	Pass
#1	s0270	51.65	✓
#2	s0262	42.98	✗
#3	s0265	10.35	✗

Exercise plan: Recap and 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: Cifar10 with Pytorch

Pytorch/Tensorboard

Exercise 09: Convolutional Neural
Networks
Exercise 10: Semantic Segmentation
Exercise 11: Recurrent Neural Networks

Applications
(Hands-off)

Good luck &
see you next week

