

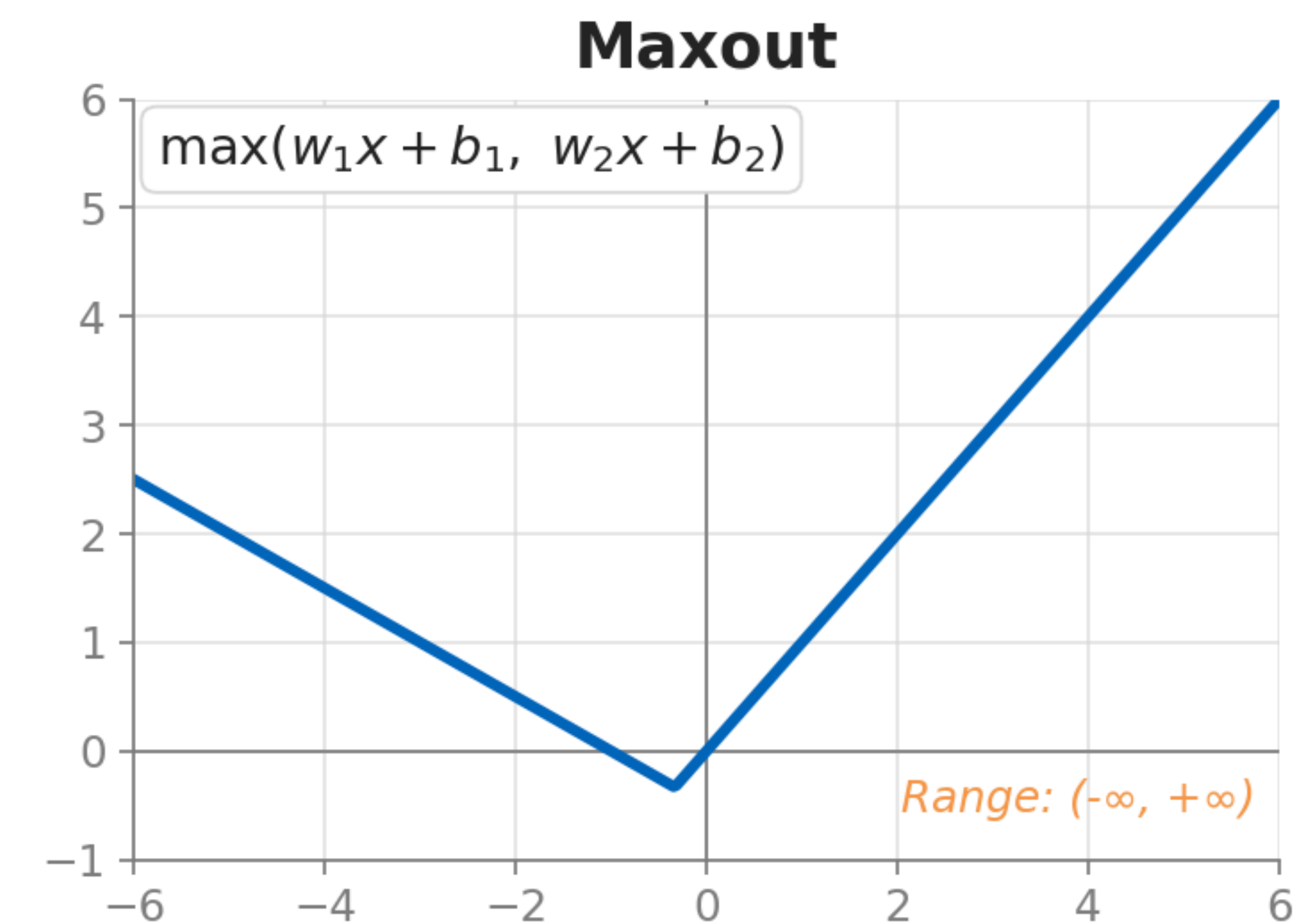
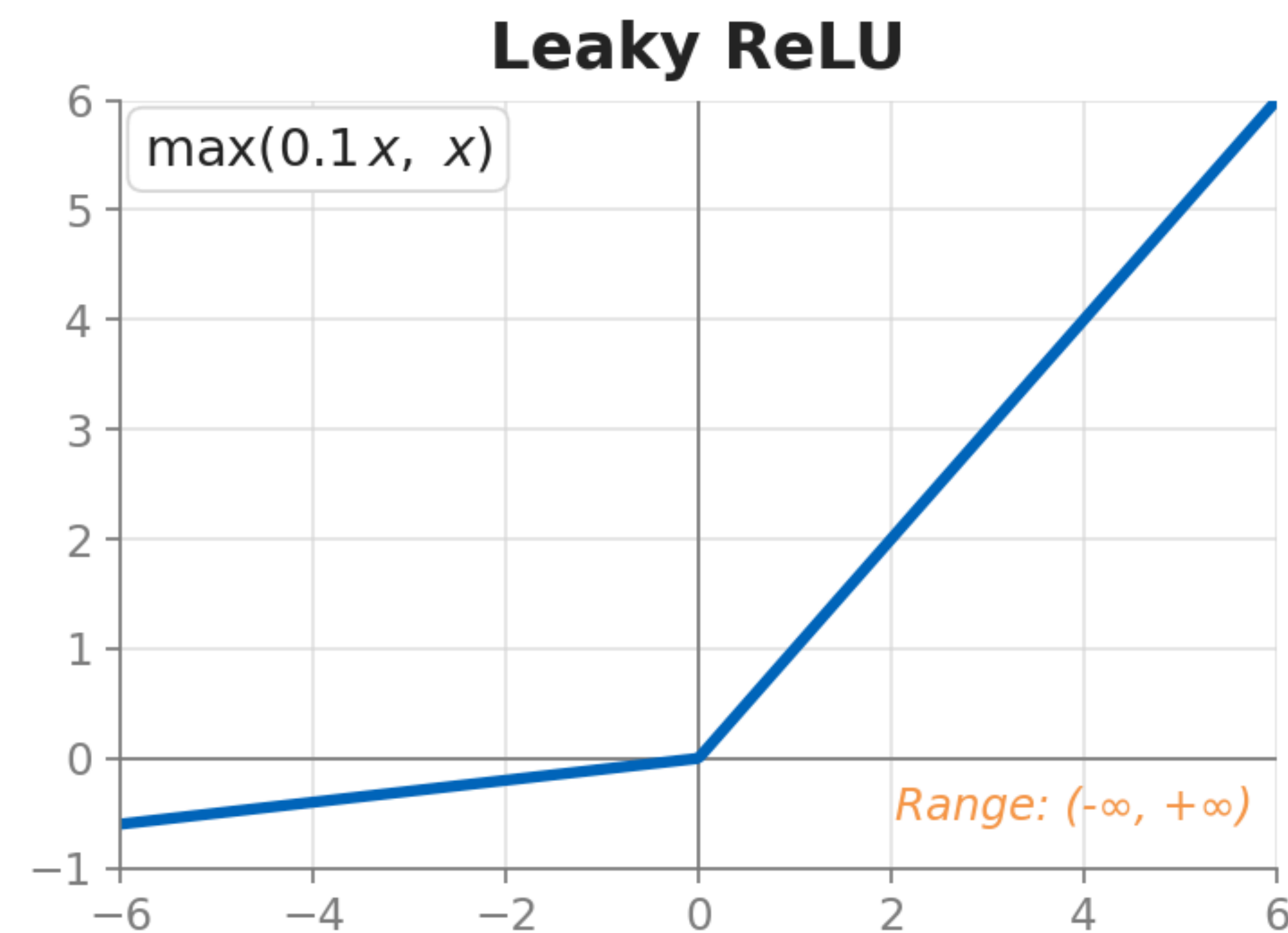
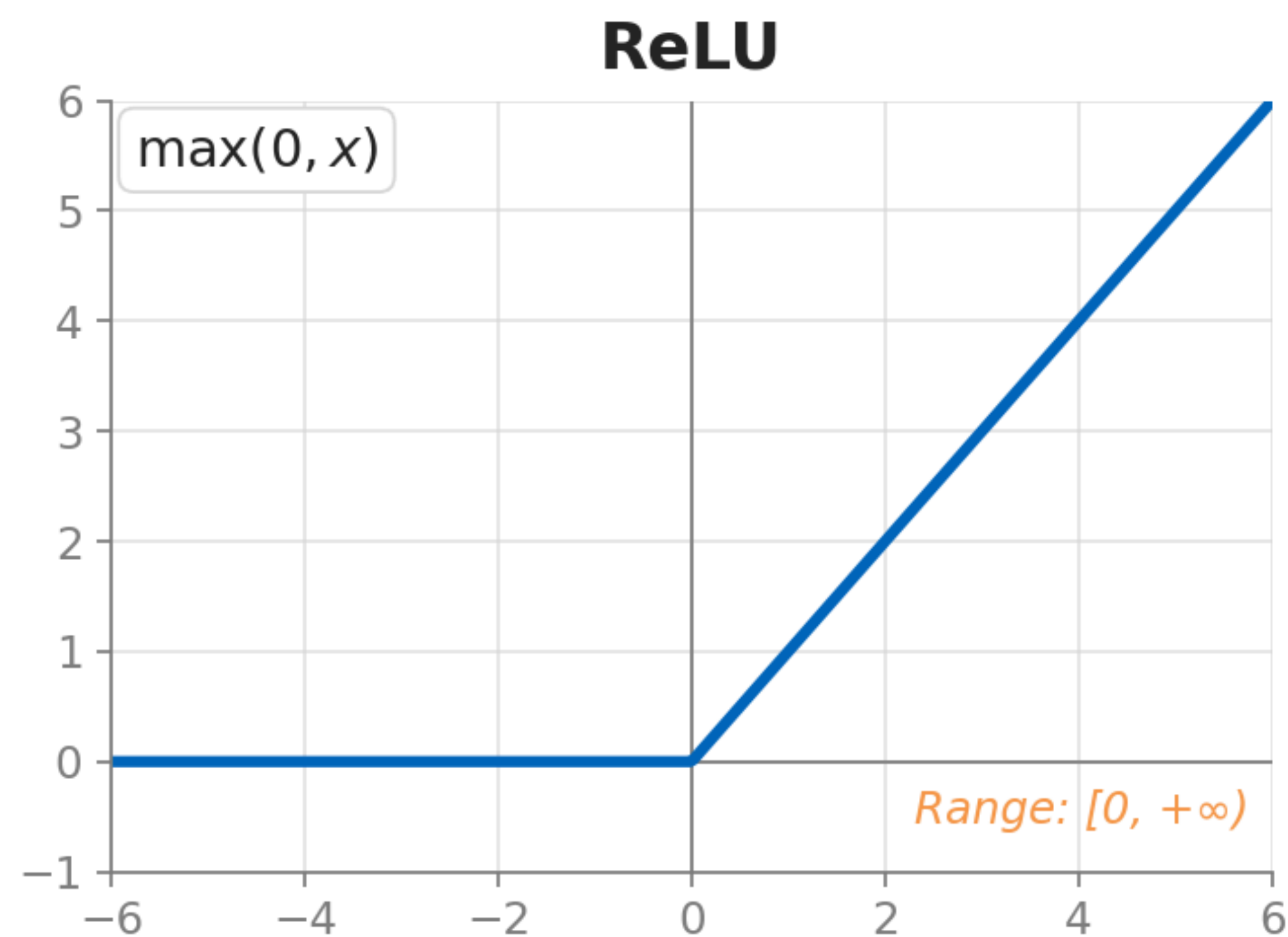
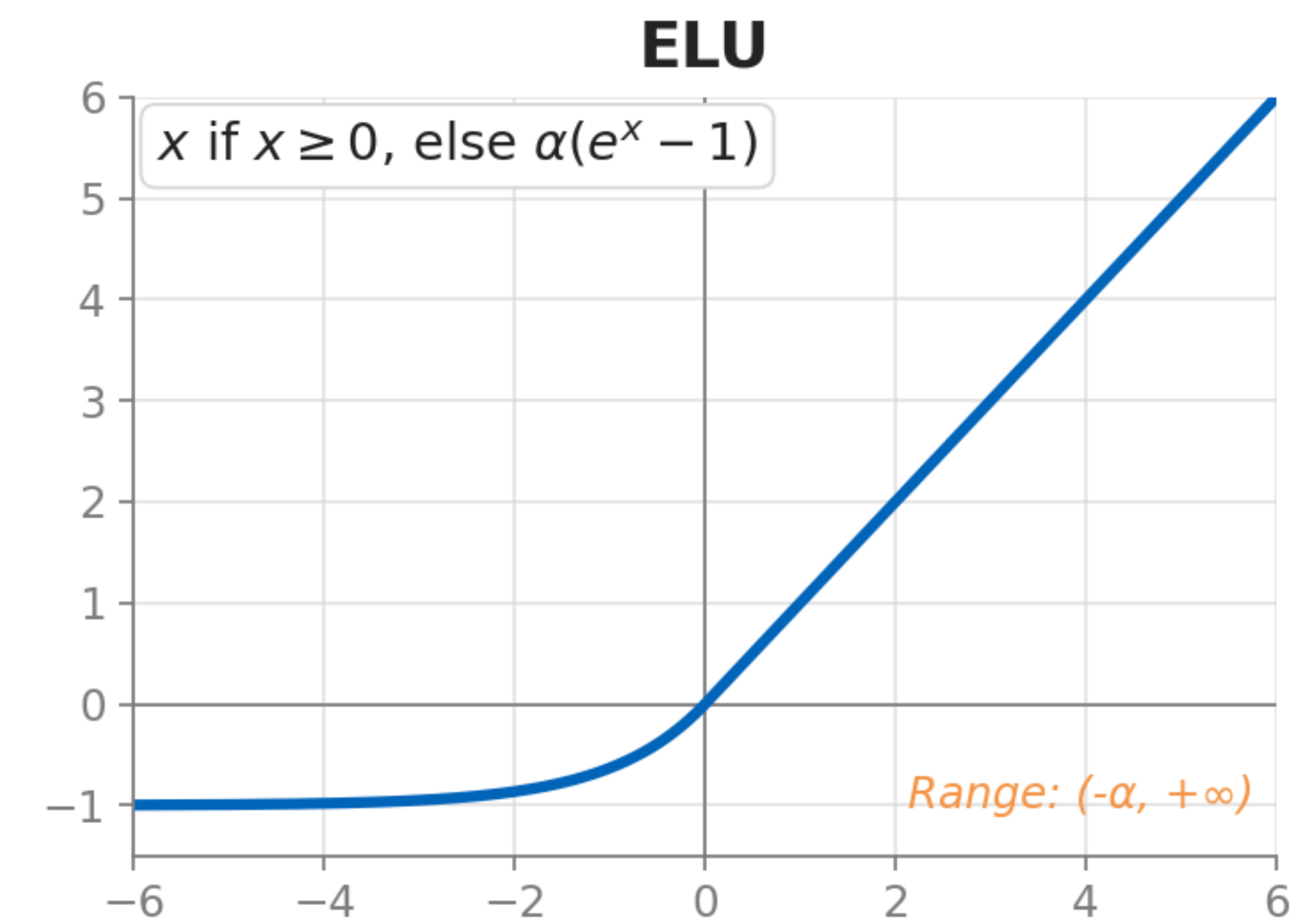
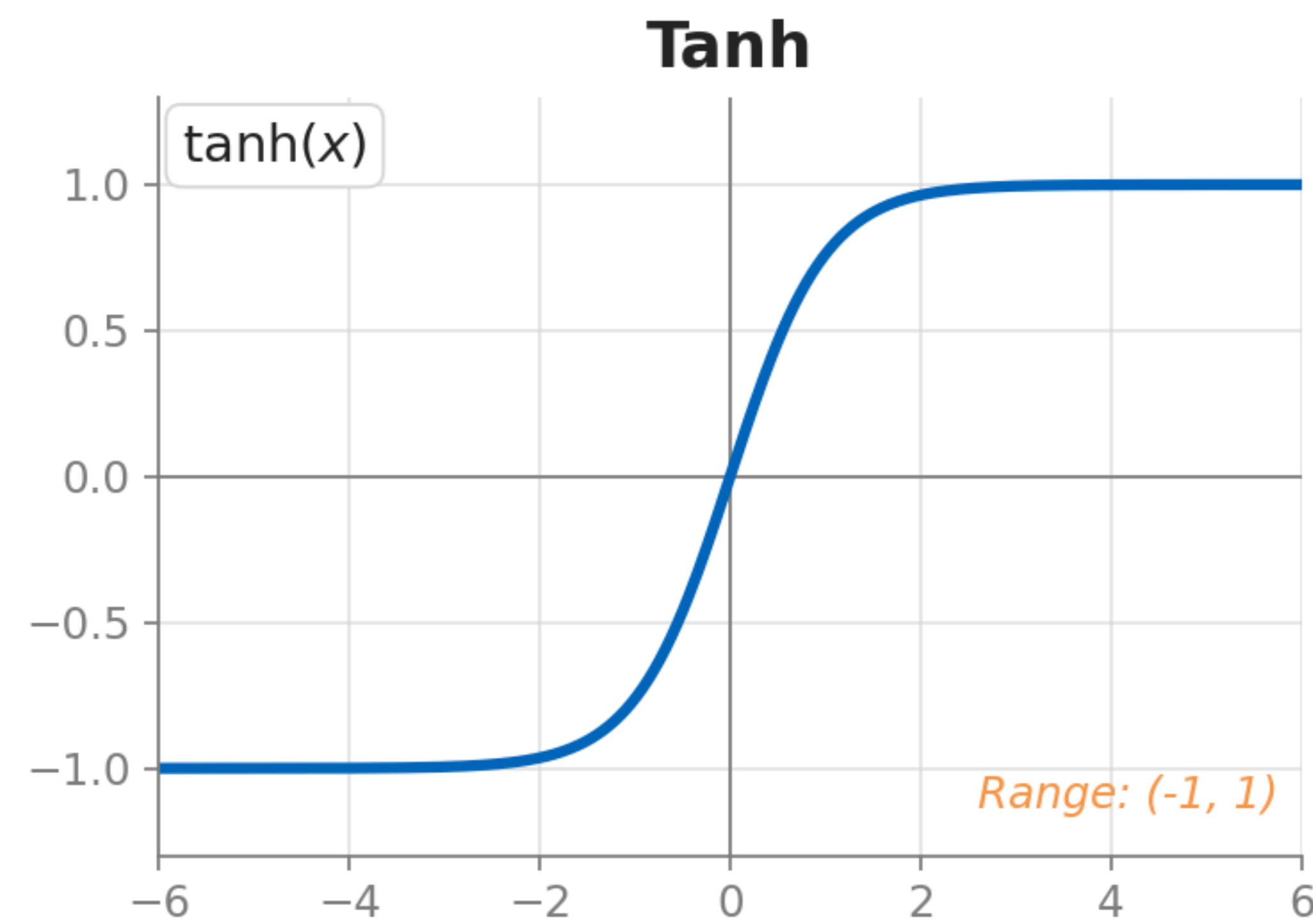
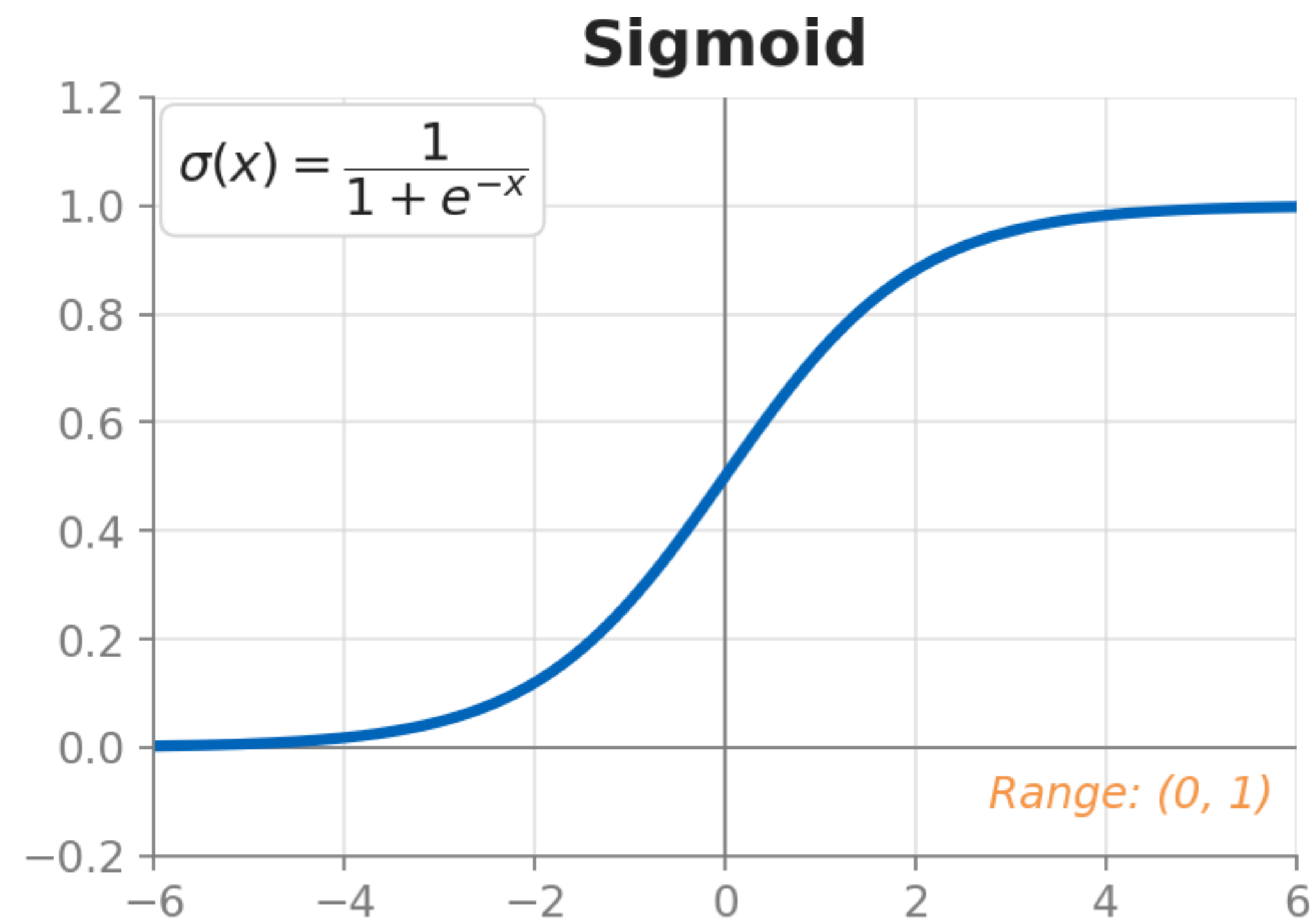
Introduction to Deep Learning (I2DL)

Tutorial 6: Hyperparameter Tuning

Today's Outline

- **Review: Solution of Exercise 5**
 - Sigmoid activation function — forward and backward pass
- **Introduction: Exercise 6**
 - Hyperparameter tuning on CIFAR-10

Activation Functions



Sigmoid: Forward Pass

Definition

- **Scalar definition** $\sigma : \mathbb{R} \rightarrow \mathbb{R}$

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

- **Derivative — a clean self-reference**

$$\sigma'(x) = \sigma(x) (1 - \sigma(x))$$

- **Element-wise on a vector**

$$\tilde{\sigma} : \mathbb{R}^n \rightarrow \mathbb{R}^n$$

$$\tilde{\sigma}(x)_i = \sigma(x_i)$$

Code

- Vectorised in two lines — NumPy broadcasts the scalar `1` over `x`.

```
out = 1 / (1 + np.exp(-x))
```

```
cache = out
```

- **Cache the output, not the input** — the backward pass only needs $\sigma(x)$, so caching `out` saves a recomputation.

Sigmoid: Backward Pass

From Jacobian to a vector

- Element-wise sigmoid maps $\mathbb{R}^n \rightarrow \mathbb{R}^n$, so its derivative is an $n \times n$ Jacobian.

- **Each output depends only on its own input** — the Jacobian is diagonal.

$$\frac{\partial \tilde{\sigma}_i}{\partial x_j} = \begin{cases} \sigma(x_i) (1 - \sigma(x_i)) & i = j \\ 0 & i \neq j \end{cases}$$

- Store only the diagonal — an n -vector. Matrix multiplication collapses to element-wise multiplication.

Code

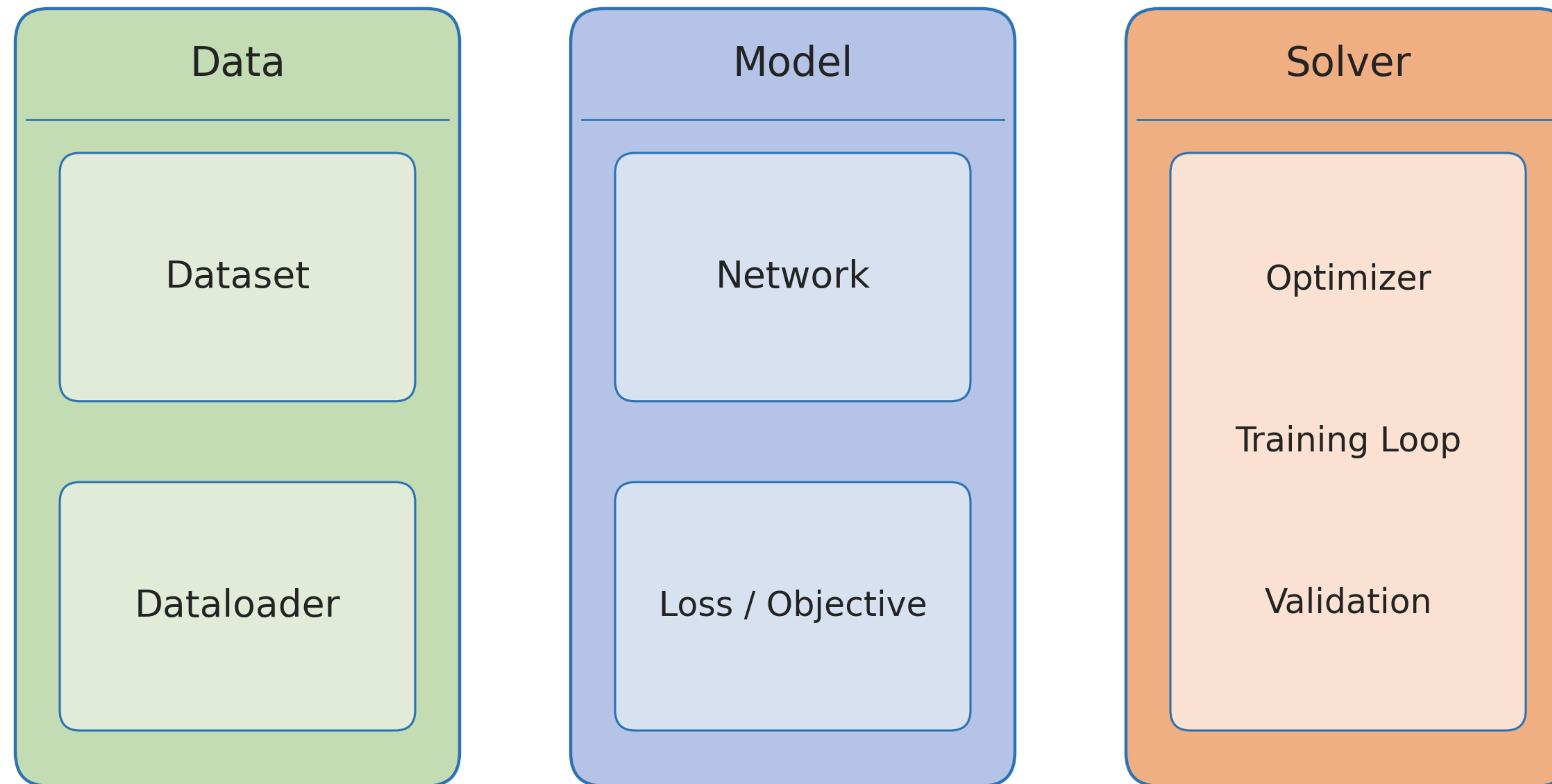
- Chain rule with the upstream gradient `dout`:

$$dx = dout * cache * (1 - cache)$$

- **One line, element-wise.** The `*` is NumPy element-wise multiplication — the diagonal-Jacobian shortcut, written out.

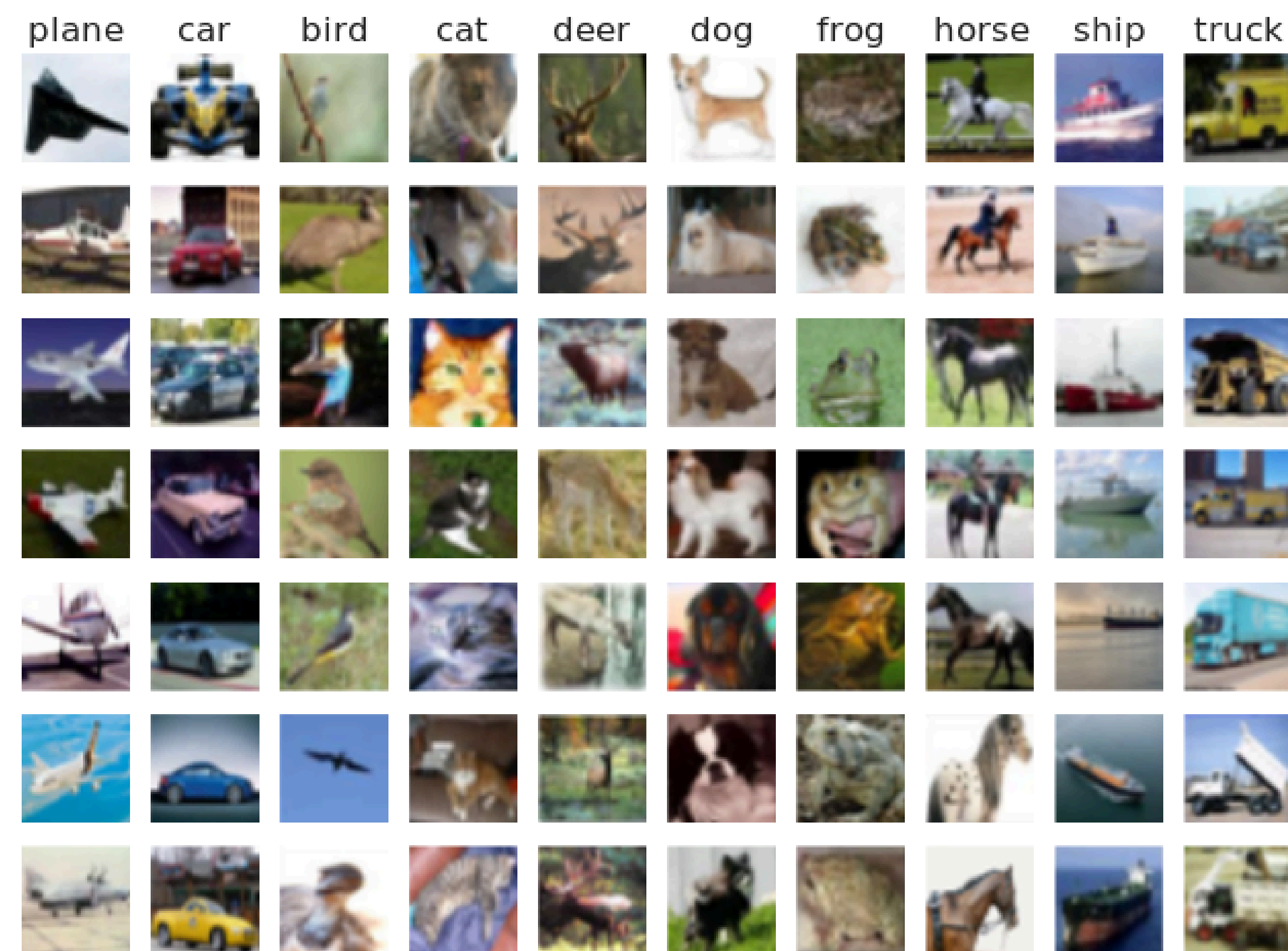
Exercise 6: Hyperparameter Tuning

Recap: The Pillars of Deep Learning



Goal: Exercise 6

Image classification on CIFAR-10



What's new

- **Reuse our reworked implementations** of Dataset, DataLoader, Solver, Network.
- **Focus on training and debugging**, not on rewriting the building blocks.
- **Hyperparameter search** — squeeze the last bit of performance out of the network.

Leaderboard

- **Your model's accuracy is all that counts.**
- At least **48 %** on the hidden test set to pass the submission.
- All submissions are ranked — there is a public leaderboard from this exercise on.
- Submit as often as you like and watch your standing move.

Previously: Dataset & DataLoader

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
```

DataLoader

```
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

Solver class

```
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): ...
```

How you call it

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

```
solver.train(epochs=epochs)
```

- Pick **SGD** or **Adam** as the optimizer — both are wired in.

Previously: Network & BCE Loss

ClassificationNet

```
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): ...
```

Binary Cross Entropy — used in Ex 4

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

- Two classes only — cheap or expensive, in / out, yes / no.
- CIFAR-10 has ten classes, so BCE no longer applies.

New: Multiclass Cross Entropy Loss

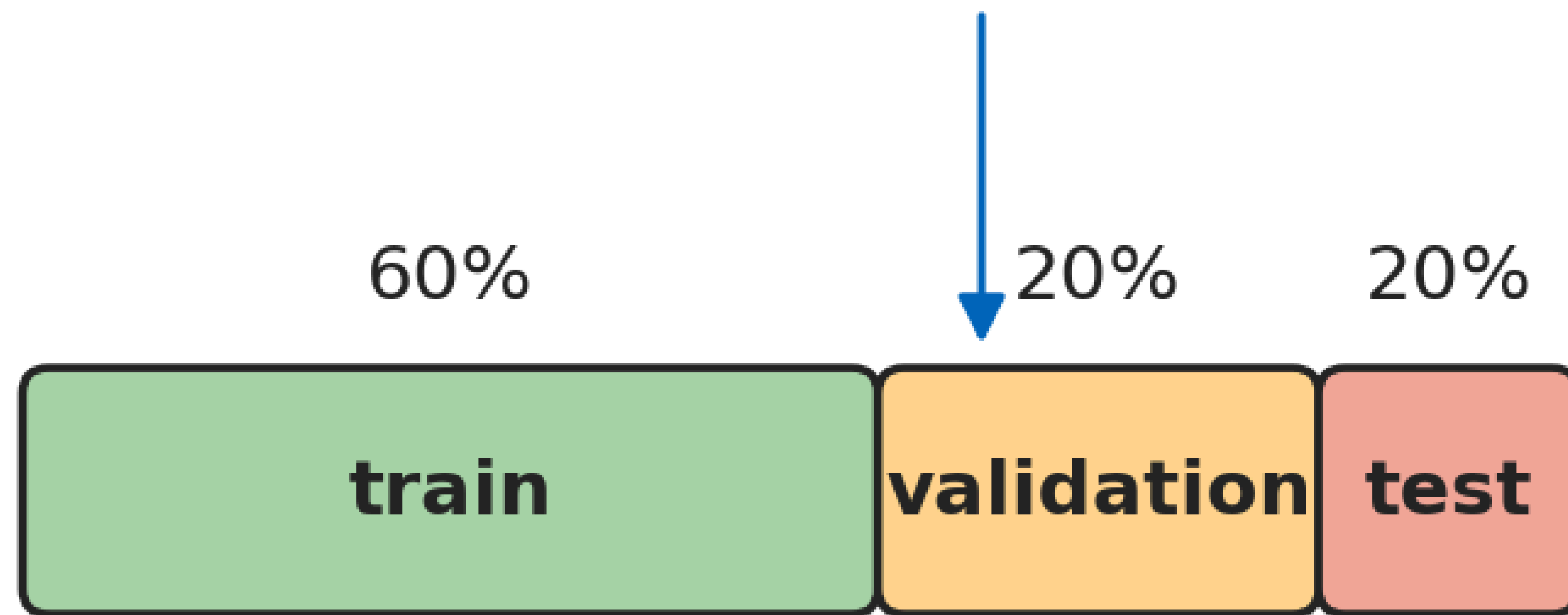
- The general form, summed over C classes per sample:

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^C y_{i,c} \log \hat{y}_{i,c}$$

- $y_{i,c}$ is the one-hot ground truth (1 for the correct class, 0 otherwise).
- $\hat{y}_{i,c}$ is the network's predicted probability for class c — a softmax over the ten output logits.
- **We implemented this for you.** Focus on the hyperparameter tuning.

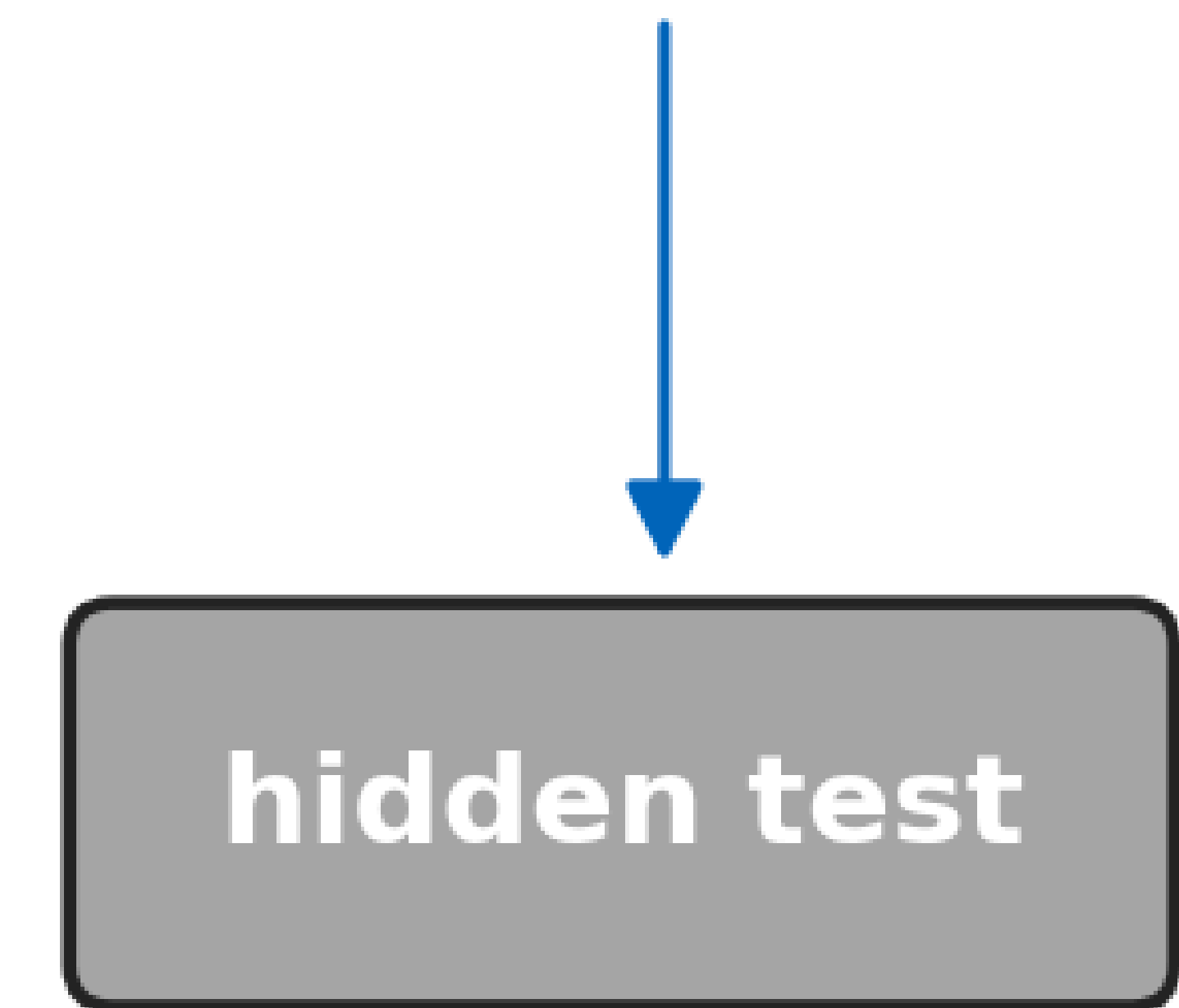
Basic Recipe: Split Your Data

Find your hyperparameters



Train a model, evaluate on validation, repeat with new hyperparameters.

Benchmarking



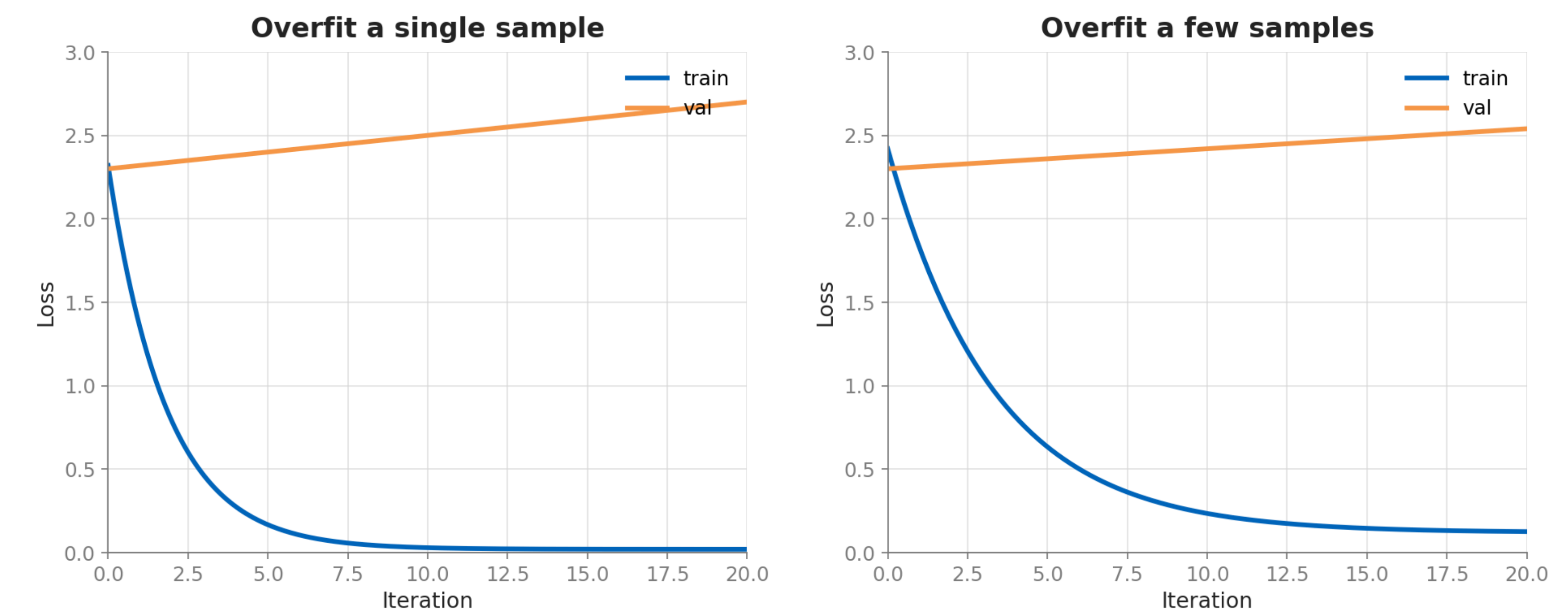
*On our submission server.
Only used once at the end!*

How to Start: Overfit Small First

Climb the overfit ladder

- **1. Overfit a single sample.**
 - Train loss should fall to ~ 0 ; train accuracy should hit 100 %.
 - Validation loss won't move — that's expected.
- **2. Overfit a handful of samples (10-15).**
 - Train loss still drops; train accuracy stays near 100 %.
- **3. Train on the full dataset.**
 - Now you want generalisation — validation loss should drop too.

Loss curves you should see



What is a Hyperparameter?

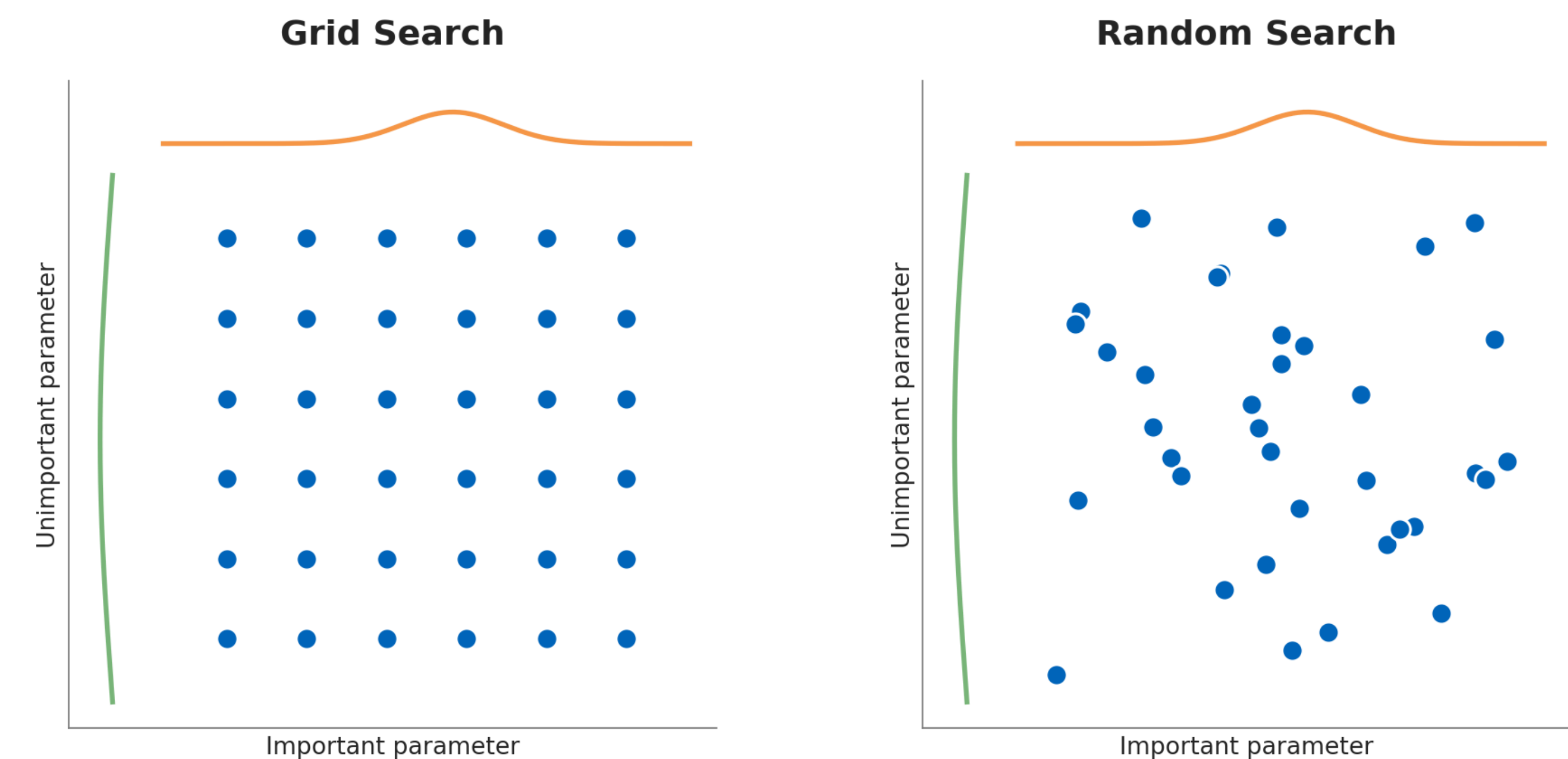
- **Anything set before training that isn't a learned weight.**
- **Architecture** — number of layers, hidden width, activation function.
- **Optimisation schedule** — number of iterations or epochs, learning rate, decay schedule.
- **Regularisation strength** — more on this in the next lecture.
- **Batch size** — bigger batches mean smoother gradients, smaller ones noisier.

How to Find Good Hyperparameters

Three approaches

- **Manual** — trial and error. Cheap, builds intuition, doesn't scale.
- **Grid search** — try every combination on a fixed lattice. Exhaustive but expensive.
- **Random search** — sample each hyperparameter independently from a range. Wins when only a few hyperparameters matter.
- **React to what you see:** overfitting → raise regularisation or shrink the model; underfitting → grow it or train longer.

Grid vs. Random



Exercise Plan: Recap and Outlook

NumPy (Reinvent the wheel)

- Ex 01: Organization
- Ex 02: Math Recap
- Ex 03: Datasets
- Ex 04: Linear Regression
- Ex 05: Neural Networks
- Ex 06: Hyperparameter Tuning ← you are here**

PyTorch / Tensorboard

- Ex 07: Introduction to PyTorch
- Ex 08: Autoencoder

Applications (Hands-off)

- Ex 09: Convolutional Networks
- Ex 10: Semantic Segmentation
- Ex 11: Transformers (1/2)
- Ex 12: Transformers (2/2)

Good luck
See you next week.