Vanishing and Exploding Gradients

Lucas Torres, Miguel Amaral,
Morsinaldo Medeiros, Thaís Medeiros

AGENDA

Vanishing Gradients Problem

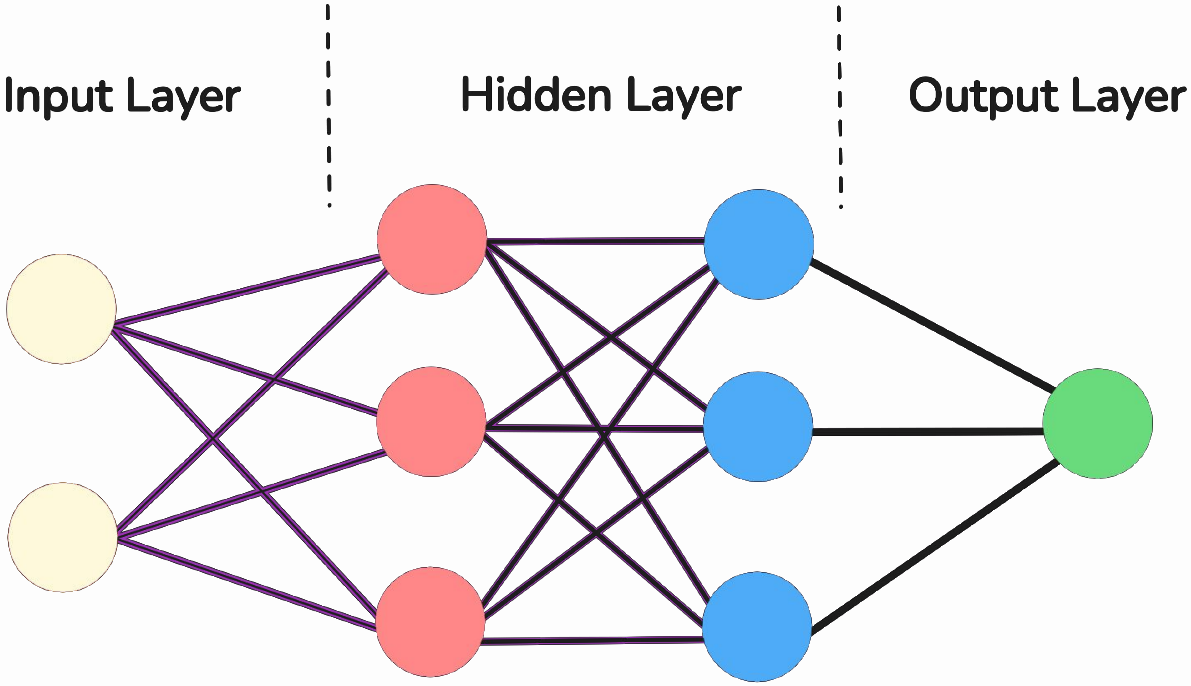
a. Visualization in deep models
b. Weight initialization techniques
c. Impact of initialization on gradients
d. Batch normalization as a solution

Exploding Gradients Problem

a. Understanding the issue
b. Techniques for gradient clipping
c. Value clipping vs. norm clipping
d. Backward hooks for gradient clipping
e. Clipping during vs. after backpropagation

Vanishing Gradients

The Problem



Vanishing Gradients

THE PROBLEM

In deep networks, the gradients become **smaller and smaller**

WHY IS TI SO BAD?

If the gradients are **close to zero**, the **weights will barely change**

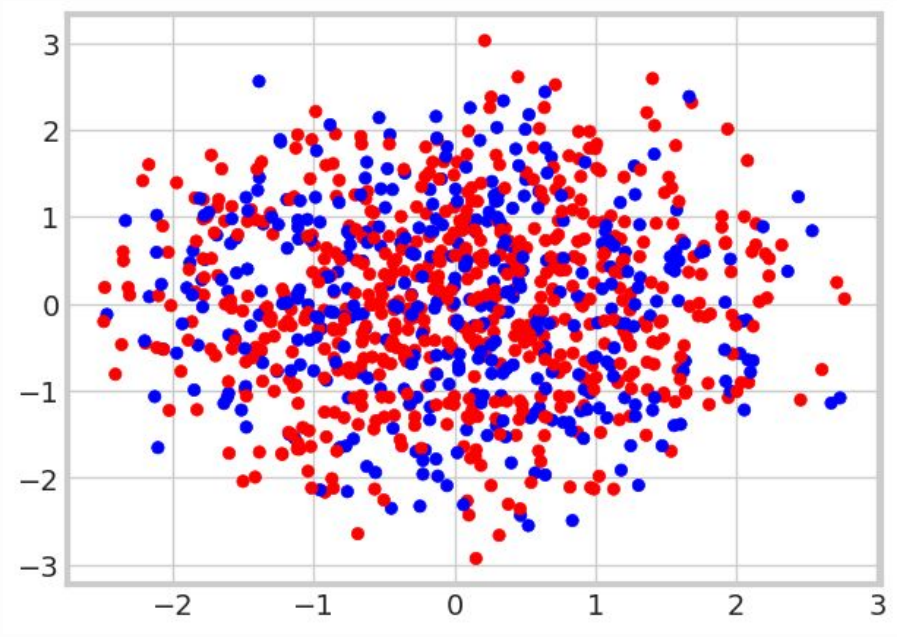
WHY DOES IT HAPPEN?

Because of the **internal covariate shift**

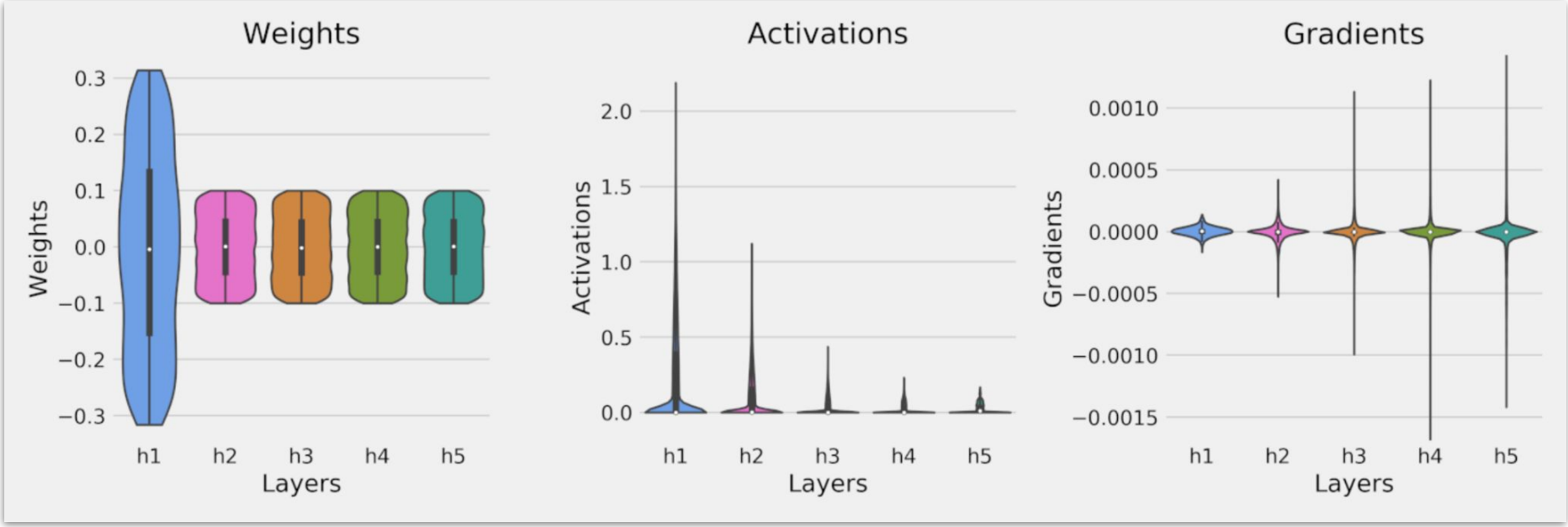
Vanishing Gradients
Model Configuration

```
torch.manual_seed(11)
n_features = X.shape[1]
n_layers = 5
hidden_units = 100
activation_fn = nn.ReLU
model = build_model(
    n_features, n_layers, hidden_units,
    activation_fn, use_bn=False
)
```

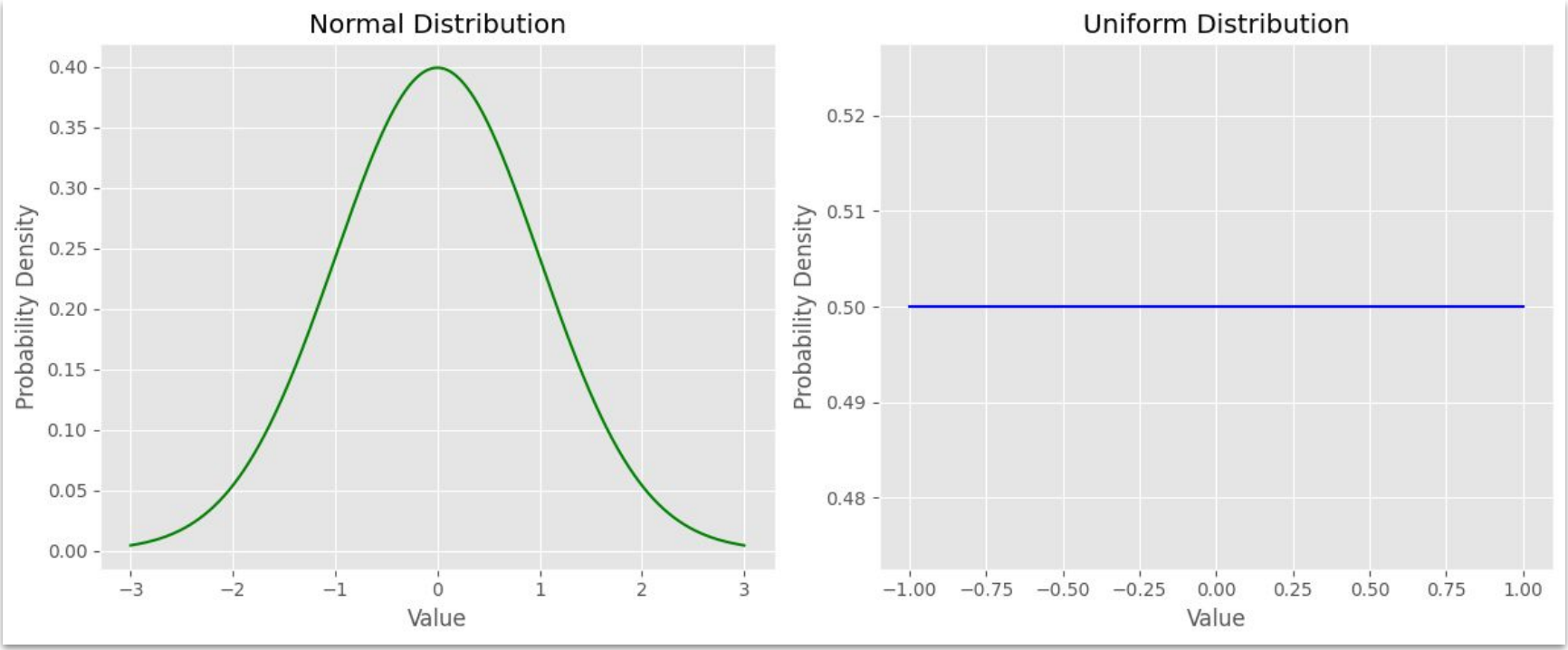
```
Sequential(
  (h1): Linear(in_features=10, out_features=100, bias=True)
  (a1): ReLU()
  (h2): Linear(in_features=100, out_features=100, bias=True)
  (a2): ReLU()
  (h3): Linear(in_features=100, out_features=100, bias=True)
  (a3): ReLU()
  (h4): Linear(in_features=100, out_features=100, bias=True)
  (a4): ReLU()
  (h5): Linear(in_features=100, out_features=100, bias=True)
  (a5): ReLU()
  (o): Linear(in_features=100, out_features=1, bias=True)
)
```



Vanishing Gradients



Vanishing Gradients
Initialization Schemes



PYTORCH'S DEFAULTS INITIALIZATION

Vanishing Gradients
Initialization Schemes

Each PyTorch layer has its **own default initialization** of the weights in the `reset_parameters()` method

```
# nn.Linear.reset_parameters()
def reset_parameters(self) -> None:
    init.kaiming_uniform_(self.weight, a=math.sqrt(5))
    if self.bias is not None:
        fan_in, _ = init._calculate_fan_in_and_fan_out(self.weight)
        bound = 1 / math.sqrt(fan_in)
        init.uniform_(self.bias, -bound, bound)
```

MANUAL INITIALIZATION OF WEIGHTS

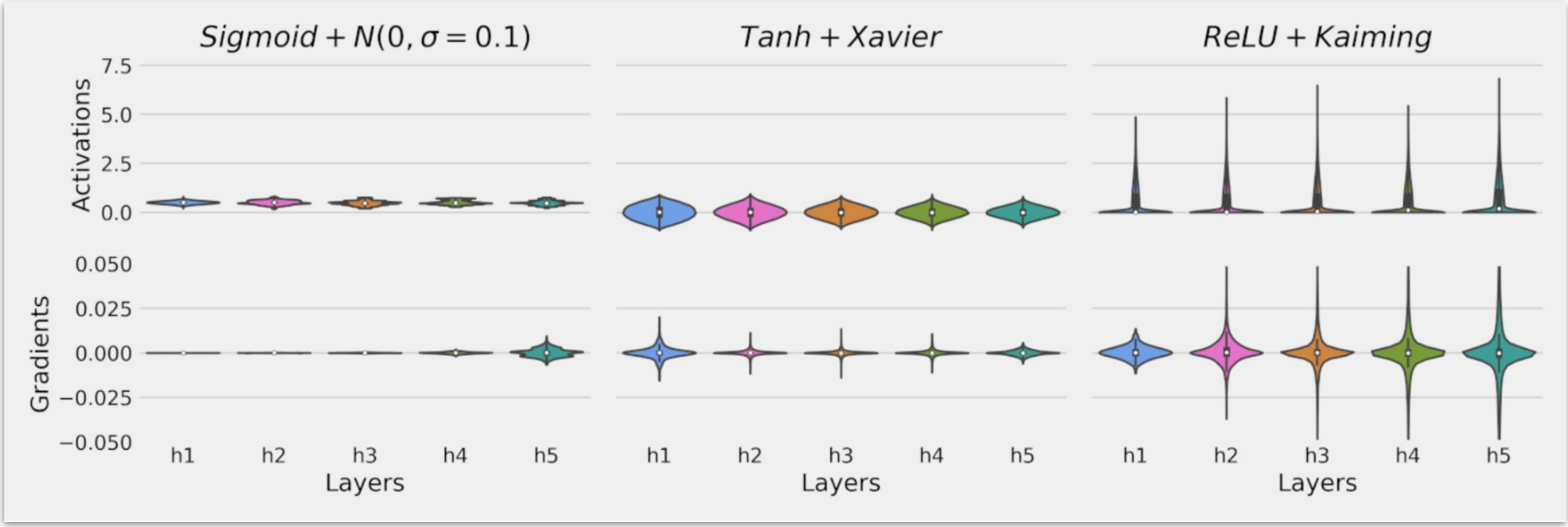
Vanishing Gradients
Initialization Schemes

To **manually initialize** the weights of your model, you can apply the function `weights_init()` and it will recursively apply to **all its internal layers**.

```python
def weights_init(m):
    if isinstance(m, nn.Linear):
        nn.init.kaiming_uniform_(m.weight, nonlinearity='relu')
        if m.bias is not None:
            nn.init.zeros_(m.bias)
```

```python
with torch.no_grad():
    model.apply(weights_init)
```

Vanishing Gradients
Initialization Schemes



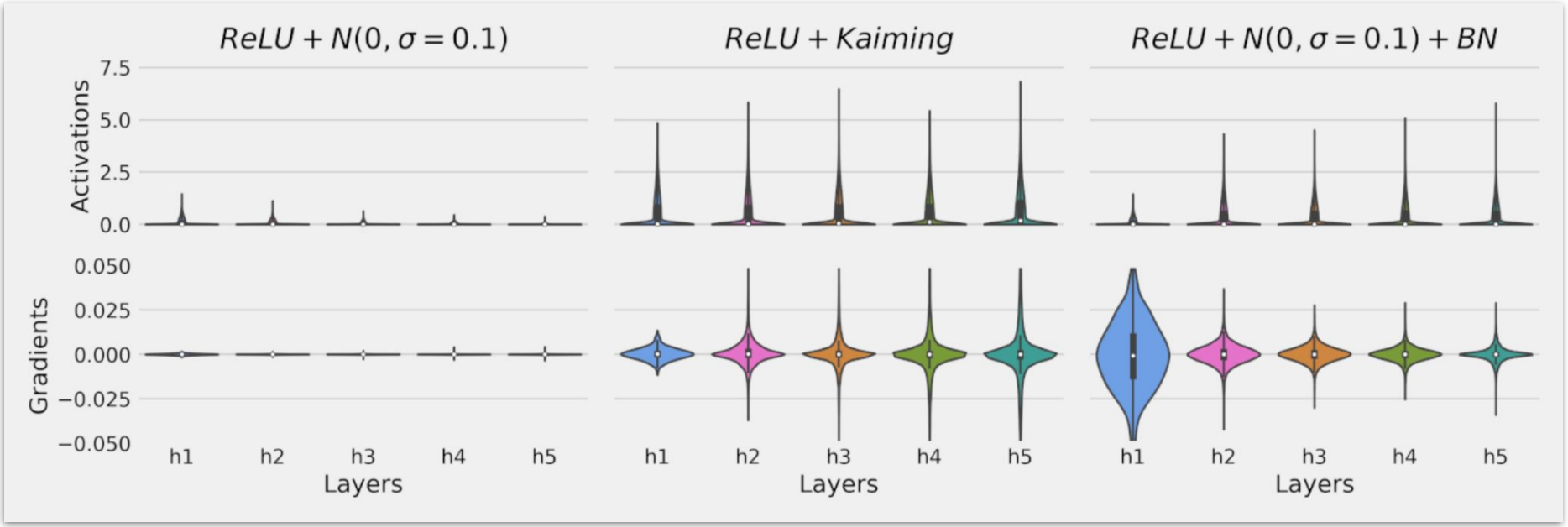
Vanishing Gradients
Batch Normalization

WHAT IS IT?

CAN WE GET AWAY WITH A BAD INITIALIZATION?

BATCH NORMALIZATION IS ALL YOU NEED!

Vanishing Gradients
Batch Normalization



Exploding Gradients

WHAT IS IT?

Gradients grow **excessively large** during backpropagation

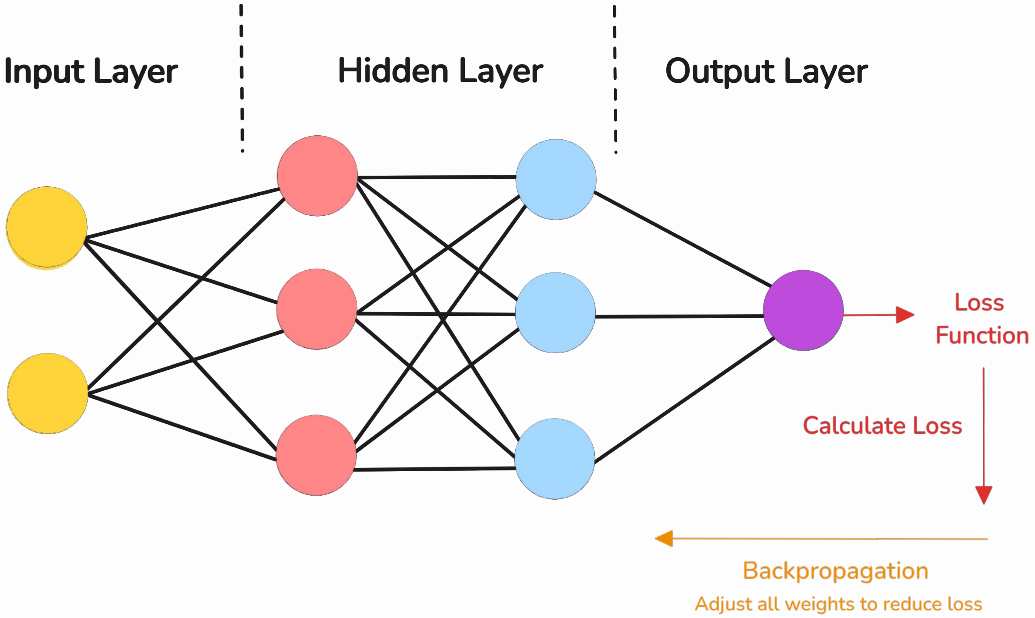
WHY DOES IT HAPPEN?

The **compounding effect of gradients** in deep networks, especially recurrent ones, or because of a **high learning rate** and a **target variable with a wide range**

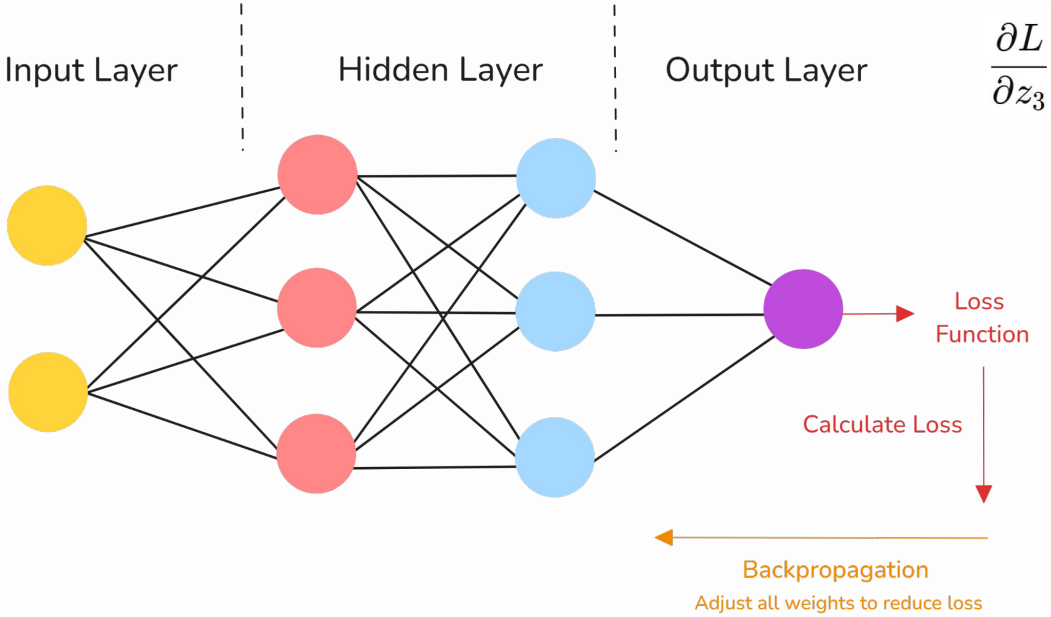
HOW TO FIX IT?

Using **gradient clipping**, **reducing the learning rate**, or **standardize the target value**

Exploding Gradients
The Problem



Exploding Gradients
The Problem



$$\frac{\partial L}{\partial z_3} = 5, \quad \frac{\partial z_3}{\partial z_2} = 4, \quad \frac{\partial z_2}{\partial z_1} = 3, \quad \frac{\partial z_1}{\partial W} = 2$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial z_3} \times \frac{\partial z_3}{\partial z_2} \times \frac{\partial z_2}{\partial z_1} \times \frac{\partial z_1}{\partial W}$$

$$\frac{\partial L}{\partial W} = 5 \times 4 \times 3 \times 2 = 120$$


MARVEL STUDIOS
WHAT IF...?

Exploding Gradients
The Problem

1. 50 Layers

2. Each derivative in each layer has an average value of 4

$$\frac{\partial L}{\partial W} = 4^{50}$$

NaN ⟵ Very Large!!!

$$\frac{\partial L}{\partial W} = 1.27 \times 10^{30}$$

Exploding Gradients

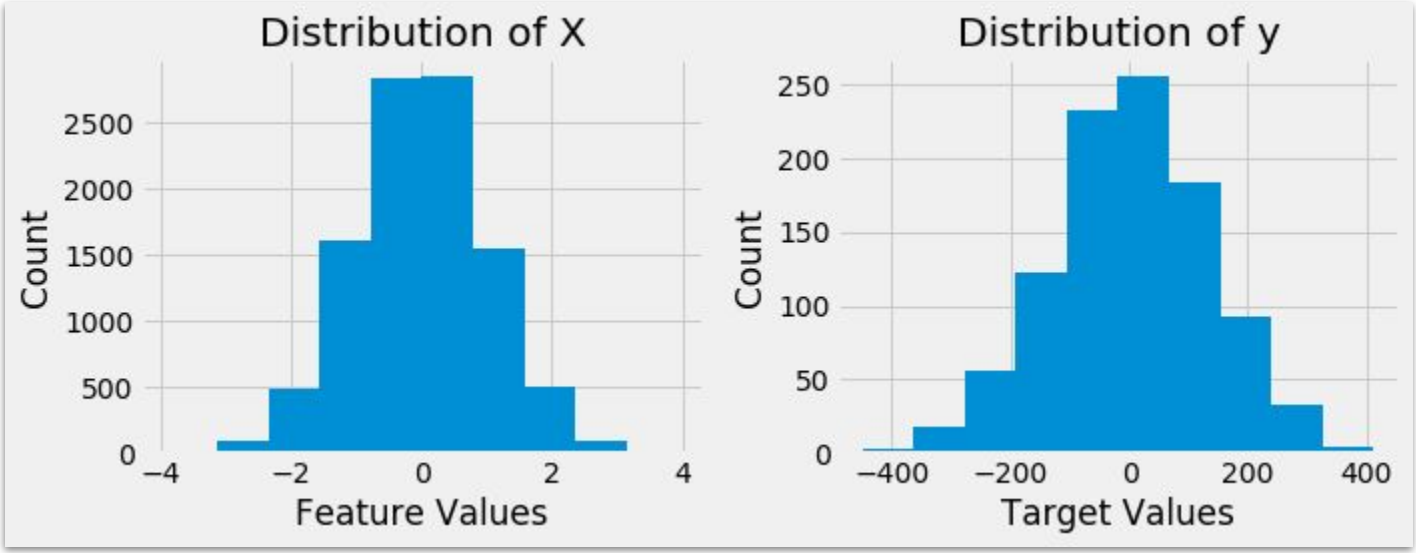
Data Generation & Preparation

```
X_reg, y_reg = make_regression(
    n_samples=1000, n_features=10, noise=0.1, random_state=42
)
X_reg = torch.as_tensor(X_reg).float()
y_reg = torch.as_tensor(y_reg).float().view(-1, 1)

dataset = TensorDataset(X_reg, y_reg)
train_loader = DataLoader(
    dataset=dataset, batch_size=32, shuffle=True
)
```

Exploding Gradients

Data Generation & Preparation



Exploding Gradients

Model Configuration & Training

```
torch.manual_seed(11)
model = nn.Sequential()
model.add_module('fc1', nn.Linear(10, 15))
model.add_module('act', nn.ReLU())
model.add_module('fc2', nn.Linear(15, 1))
optimizer = optim.SGD(model.parameters(), lr=0.01)
loss_fn = nn.MSELoss()
```

```
sbs_reg = StepByStep(model, loss_fn, optimizer)
sbs_reg.set_loaders(train_loader)
sbs_reg.capture_gradients(['fc1'])
sbs_reg.train(2)
```

Exploding Gradients
Model Configuration & Training

```
sbs_reg.losses
```

Output

```
[16985.014358520508, nan]
```

```
grads = np.array(sbs_reg._gradients['fc1']['weight'])
print(grads.mean(axis=(1, 2)))
```

Exploding Gradients

Model Configuration & Training

Output

```
[ 1.58988627e+00 -2.41313894e+00  1.61042006e+00  4.27530414e+00
  2.00876453e+01 -5.46269826e+01  4.76936617e+01 -6.68976169e+01
  4.89202255e+00 -5.48839445e+00 -8.80165010e+00  2.42120121e+01
 -1.95470126e+01 -5.61713082e+00  4.16399702e+01  2.09703794e-01
  9.78054642e+00  8.47080885e+00 -4.37233462e+01 -1.22754592e+01
 -1.05804357e+01  6.17669332e+00 -3.27032627e+00  3.43037068e+01
  6.90878444e+00  1.15130024e+01  8.64732616e+00 -3.04457552e+01
 -3.79791490e+01  1.57137556e+01  1.43945687e+01  8.90063342e-01
 -3.60141261e-01  9.18566430e+00 -7.91019879e+00  1.92959307e+00
 -6.55456380e+00 -1.66785977e+00 -4.13915831e+01  2.03403218e+01
 -5.39869087e+02 -2.33201361e+09  3.74779743e+26            nan
            nan             nan             nan            nan
            nan             nan             nan            nan
            nan             nan             nan            nan
            nan             nan             nan            nan
            nan             nan             nan            nan]
```

How do we fix it?

Exploding Gradients
Gradient Clipping

You **pick a value** and **clip gradients higher** (in absolute terms) than the value you picked

VALUE CLIPPING **VS** NORM CLIPPING

```
torch.manual_seed(42)
parm = nn.Parameter(torch.randn(2, 1))
fake_grads = torch.tensor([[2.5], [.8]])
```

VALUE CLIPPING

Exploding Gradients
Gradient Clipping

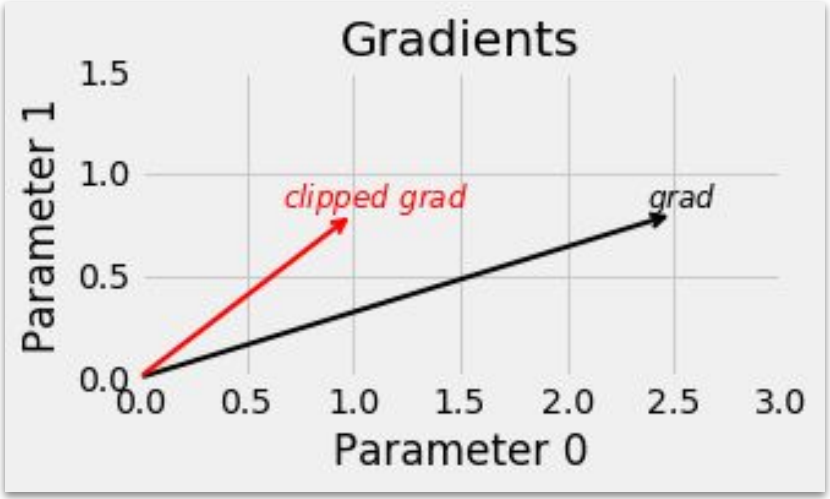
```
parm.grad = fake_grads.clone()
#Gradient Value Clipping
nn.utils.clip_grad_value_(parm, clip_value=1.0)
parm.grad.view(-1,)
```

Output

```
tensor([1.0000, 0.8000])
```

VALUE CLIPPING

Exploding Gradients
Gradient Clipping



NORM CLIPPING

Exploding Gradients
Gradient Clipping

Computes the norm for **all gradients together** as if they were concatenated into a single vector

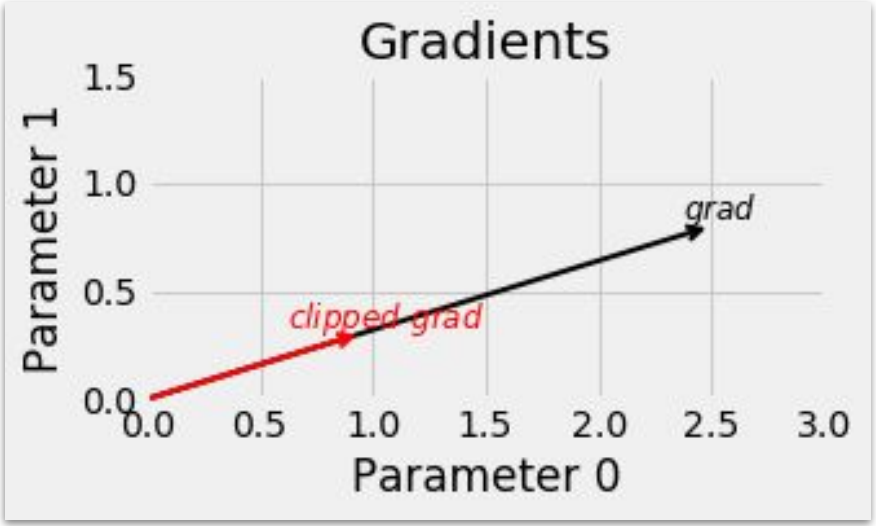
```
parm.grad = fake_grads.clone()
# Gradient Norm Clipping
nn.utils.clip_grad_norm_(parm, max_norm=1.0, norm_type=2)
fake_grads.norm(), parm.grad.view(-1,), parm.grad.norm()
```

Output

```
(tensor(2.6249), tensor([0.9524, 0.3048]), tensor(1.0000))
```

NORM CLIPPING

Exploding Gradients
Gradient Clipping



Which one is better?

Exploding Gradients
Gradient Clipping

NORM CLIPPING

A Maintains Balance

B Direction Preservation

VALUE CLIPPING

C Speed

D Limited Impact on Direction

Exploding Gradients
Gradient Clipping

WHICH CLIP VALUE SHOULD I USE?

TREAT IT AS A HYPERPARAMETER!

CLIP VALUE
PRACTICE

Exploding Gradients
Gradient Clipping

```
setattr(StepByStep, 'clipping', None)

def set_clip_grad_value(self, clip_value):
    self.clipping = lambda: nn.utils.clip_grad_value_(
        self.model.parameters(), clip_value=clip_value
    )

def set_clip_grad_norm(self, max_norm, norm_type=2):
    self.clipping = lambda: nn.utils.clip_grad_norm_(
        self.model.parameters(), max_norm, norm_type
    )

def remove_clip(self):
    self.clipping = None
```

CLIP VALUE
PRACTICE

Exploding Gradients
Gradient Clipping

```
def _make_train_step_fn(self):
    # This method does not need ARGS... it can refer to
    # the attributes: self.model, self.loss_fn, and self.optimizer

    # Builds function that performs a step in the train loop
    def perform_train_step_fn(x, y):
        # Sets model to TRAIN mode
        self.model.train()

        # Step 1 - Computes model's predicted output - forward pass
        yhat = self.model(x)
        # Step 2 - Computes the loss
        loss = self.loss_fn(yhat, y)
        # Step 3 - Computes gradients
        loss.backward()

        if callable(self.clipping):       ①
            self.clipping()               ①

        # Step 4 - Updates parameters
        self.optimizer.step()
        self.optimizer.zero_grad()

        # Returns the loss
        return loss.item()

    # Returns the function that will be called inside the train loop
    return perform_train_step_fn
```

① Gradient clipping after computing gradients and before updating parameters

```
setattr(StepByStep, 'set_clip_grad_value', set_clip_grad_value)
setattr(StepByStep, 'set_clip_grad_norm', set_clip_grad_norm)
setattr(StepByStep, 'remove_clip', remove_clip)
setattr(StepByStep, '_make_train_step_fn', _make_train_step_fn)
```

CLIP VALUE
PRACTICE

Exploding Gradients
Gradient Clipping

After we can **reset the weight and parameters** of our model, with the new adjustments we can use a **ten times higher learning rate**.

```
def weights_init(m):
    if isinstance(m, nn.Linear):
        nn.init.kaiming_uniform_(m.weight, nonlinearity='relu')
        if m.bias is not None:
            nn.init.zeros_(m.bias)
```

```
torch.manual_seed(42)
with torch.no_grad():
    model.apply(weights_init)

optimizer = optim.SGD(model.parameters(), lr=0.1)
```

CLIP VALUE
PRACTICE

Exploding Gradients
Gradient Clipping

Model Training

```
1 sbs_reg_clip = StepByStep(model, loss_fn, optimizer)
2 sbs_reg_clip.set_loaders(train_loader)
3 sbs_reg_clip.set_clip_grad_value(1.0)
4 sbs_reg_clip.capture_gradients(['fc1'])
5 sbs_reg_clip.train(10)
6 sbs_reg_clip.remove_clip()
7 sbs_reg_clip.remove_hooks()
```

CLIP VALUE
PRACTICE

Exploding Gradients
Gradient Clipping

Recurrent neural networks require gradients to be clipped **during backpropagation**. For that we use **backward hooks**.

```
def set_clip_backprop(self, clip_value):
    if self.clipping is None:
        self.clipping = []
    for p in self.model.parameters():
        if p.requires_grad:
          func = lambda grad: torch.clamp(grad,
                                          -clip_value,
                                          clip_value)
          handle = p.register_hook(func)
          self.clipping.append(handle)
```

```
def remove_clip(self):
    if isinstance(self.clipping, list):
        for handle in self.clipping:
            handle.remove()
    self.clipping = None

setattr(StepByStep, 'set_clip_backprop', set_clip_backprop)
setattr(StepByStep, 'remove_clip', remove_clip)
```

CLIP VALUE
PRACTICE

Exploding Gradients
Gradient Clipping

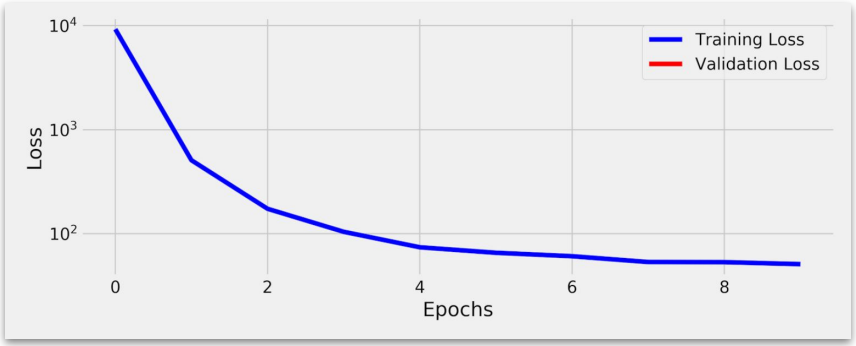
Model Training

```
1 sbs_reg_clip_hook = StepByStep(model, loss_fn, optimizer)
2 sbs_reg_clip_hook.set_loaders(train_loader)
3 sbs_reg_clip_hook.set_clip_backprop(1.0)
4 sbs_reg_clip_hook.capture_gradients(['fc1'])
5 sbs_reg_clip_hook.train(10)
6 sbs_reg_clip_hook.remove_clip()
7 sbs_reg_clip_hook.remove_hooks()
```

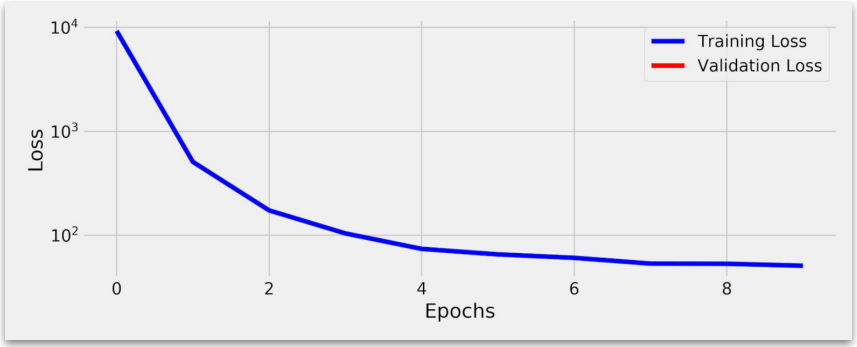
CLIP VALUE
PRACTICE

Exploding Gradients
Gradient Clipping

NORMAL CLIPPING AFTER BACKWARD



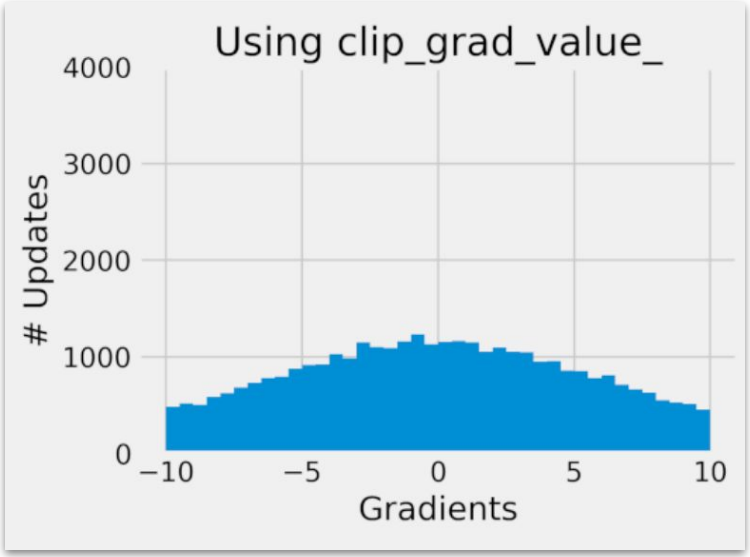
CLIPPING WITH BACKWARD HOOKS



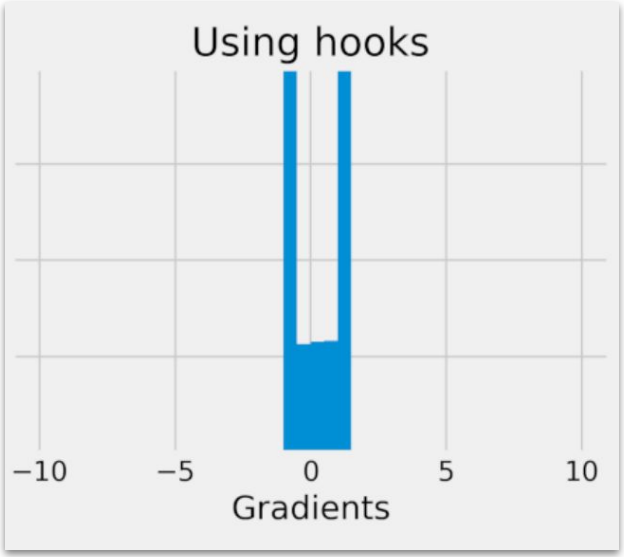
CLIP VALUE
PRACTICE

Exploding Gradients
Gradient Clipping

NORMAL CLIPPING AFTER BACKWARD



CLIPPING WITH BACKWARD HOOKS



Time to Recap

Recap

Vanishing and Exploding Gradients

- **Gradient Issues**:
  - Vanishing gradients in deeper models.
  - Exploding gradients.
- **Approaches to Vanishing Gradients**:
  - Weight initialization.
  - Initialization schemes effect.
  - Batch normalization for bad initialization.
- **Approaches to Exploding Gradients**:
  - Gradient clipping (2 methods).
  - Using hooks for clipping during backpropagation.
  - Value clipping vs Norm clipping.
  - Clipping before and during backpropagation.

Thank you
Any Questions?