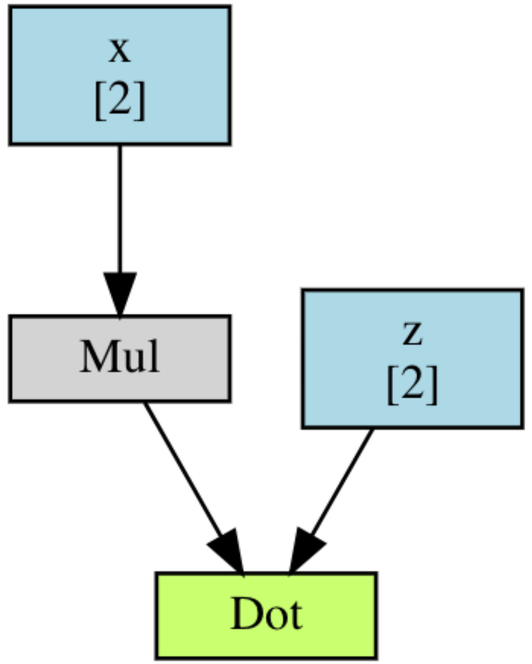
Introduction to Machine Learning with PyTorch

Tim Rocktäschel & Sebastian Riedel
COMP0087 Natural Language Processing

PyTorch Variables

```
# variable
x = torch.tensor([-1.5, 1.2], requires_grad=True)
# constant
y = torch.tensor([1.0, -1.3])
# variable
z = torch.tensor([-2.0, 0.2], requires_grad=True)
# output tensor value, but also computation graph
x * y @ z
> tensor(2.6880, grad_fn=<DotBackward>)
```



Gradients

```
x = torch.tensor([-1.5, 1.2], requires_grad=True)
y = torch.tensor([1.0, -1.3])
z = torch.tensor([-2.0, 0.2], requires_grad=True)
r = x * y @ z
r.backward()
x.grad
> tensor([-2.0000, -0.2600])
```

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \alpha \frac{\partial}{\partial \mathbf{x}_t} r_t$$

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial}{\partial \theta_t} L(X, Y; \theta_t)$$

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{x}} r &= \frac{\partial}{\partial \mathbf{x}} \left(\mathbf{x} \odot \mathbf{y}\right)^\top \mathbf{z} \\
&= \mathbf{z} \left[\frac{\partial}{\partial \mathbf{x}} \left(\mathbf{x} \odot \mathbf{y}\right)\right] + \left(\mathbf{x} \odot \mathbf{y}\right) \left[\frac{\partial}{\partial \mathbf{x}} \mathbf{z}\right] \\
&= \mathbf{z} \left[\mathbf{y} \left[\frac{\partial}{\partial \mathbf{x}} \mathbf{x}\right] + \mathbf{x} \left[\frac{\partial}{\partial \mathbf{x}} \mathbf{y}\right]\right] \\
&= \mathbf{z} \odot \mathbf{y} = \begin{bmatrix} -2.0 \\ 0.2 \end{bmatrix} \odot \begin{bmatrix} 1.0 \\ -1.3 \end{bmatrix} = \begin{bmatrix} -2.00 \\ -0.26 \end{bmatrix}
\end{aligned}$$

Why Autograd?

$$\mathbf{s}_t = \begin{bmatrix} \mathbf{x}_t \\ \mathbf{h}_{t-1} \end{bmatrix}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}^i \mathbf{s}_t + \mathbf{b}^i)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}^f \mathbf{s}_t + \mathbf{b}^f)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}^o \mathbf{s}_t + \mathbf{b}^o)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}^c \mathbf{s} + \mathbf{b}^c)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

$$\frac{\partial}{\partial \mathbf{W}^f} L(\mathbf{h}_t) ?$$

Backpropagation

= Efficient Application of Chain Rule

- Chain Rule:

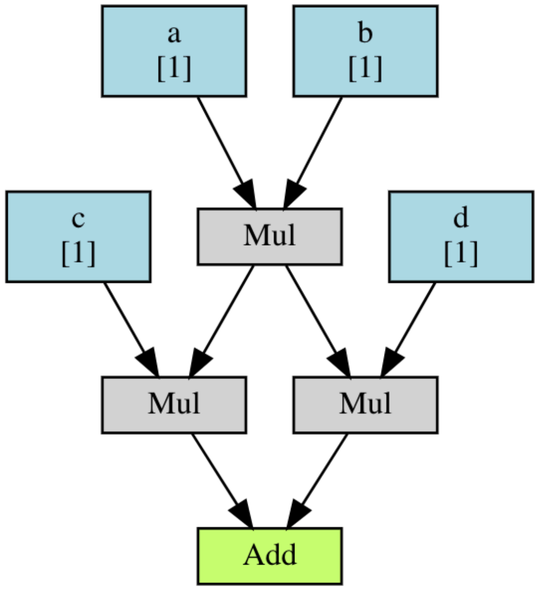
$$y = g(x)$$

$$z = f(y)$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x} = f'(g(x))g'(x)$$

- Backprop

```
a = torch.rand(1, requires_grad=True)
b = torch.rand(1, requires_grad=True)
c = torch.rand(1, requires_grad=True)
d = torch.rand(1, requires_grad=True)

e = a * b
# compare to: f = c * a * b + d * a * b
f = c * e + d * e

f
> tensor([0.0994], grad_fn=<AddBackward0>
```



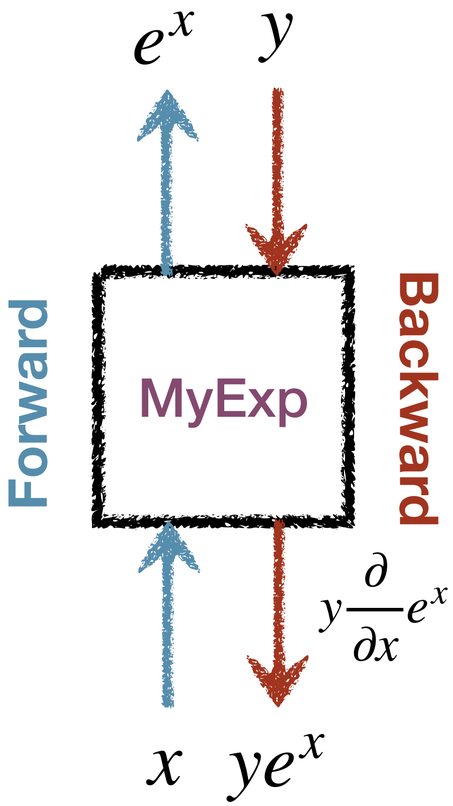
Naomi Saphra
@nsaphra
Following

What idiot called it "deep learning hype" and not "backpropaganda"

3:05 PM - 14 Apr 2016

PyTorch Autograd Function

```
from torch.autograd import Function

class MyExp(Function):
  @staticmethod
  def forward(ctx, i):
    result = i.exp()
    ctx.save_for_backward(result)
    return result

  @staticmethod
  def backward(ctx, grad_output):
    result, = ctx.saved_tensors
    return grad_output * result

x = torch.tensor([0.5], requires_grad=True)
y = MyExp.apply(x)
y.backward()
x.grad
> tensor([1.6487])
```

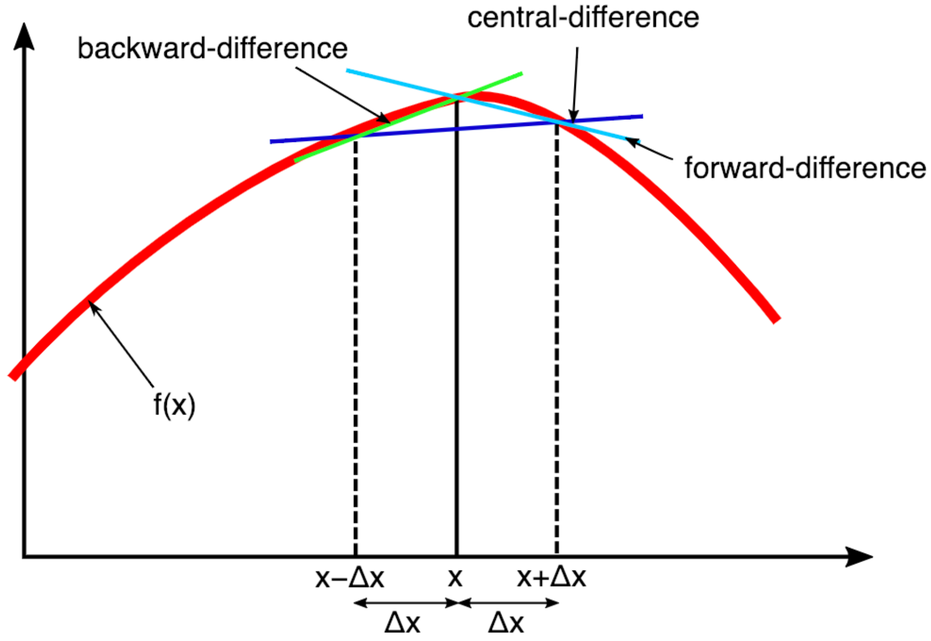


Gradient Checking

- How can we check our implementation of backward is correct w.r.t. forward? Finite-differences Approximation!

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \approx \frac{1}{2\epsilon}\left(f(\mathbf{x}+\epsilon) - f(\mathbf{x}-\epsilon)\right)$$

- Make sure to test for multiple **x**
- Benefit over $\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \approx \frac{f(\mathbf{x}+\epsilon)}{\epsilon}$?
- See `torch.autograd.gradcheck`



https://timvieira.github.io/blog/post/2017/04/21/how-to-test-gradient-implementations/
https://en.wikipedia.org/wiki/Finite_difference

PyTorch Modules

- All network components should inherit from nn.Module and override the forward method
- Using a module provides functionality:
  - Keeps track of trainable parameters
  - Allows you to easily swap between CPU and GPU (see .to(device))
- To register a variable tensor to the parameters of a module you need to wrap it using `nn.Parameter`

Linear Model Example

```
import torch.nn as nn

class LinearModule(torch.nn.Module):
    def __init__(self, x_dim, y_dim):
        super(LinearModule, self).__init__()
        self.W = nn.Parameter(torch.randn(y_dim, x_dim, requires_grad=True))
        self.b = nn.Parameter(torch.randn(y_dim))
    def forward(self, x):
        return self.W @ x + self.b

# Some random input and output data
x = torch.randn(5)
y = torch.randn(2)

model = LinearModule(5, 2)

for param in model.parameters():
    print(param.size())

model(x)
> torch.Size([2, 5])
> torch.Size([2])
> tensor([ 1.3681, -1.9464], grad_fn=<AddBackward0>)
```

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

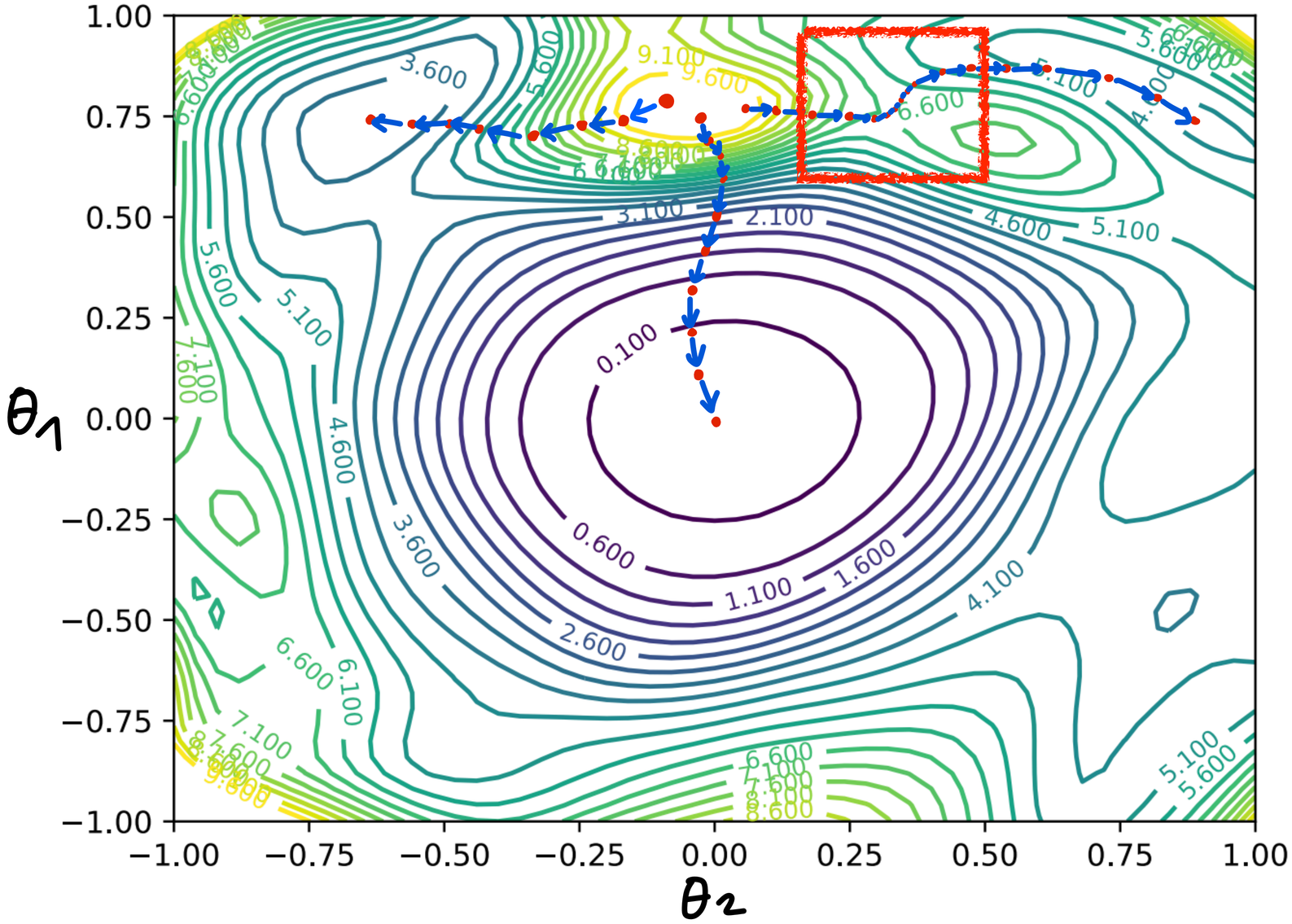
Loss Functions

- Least squares [`nn.MSELoss`] $L(f_\theta(x), y) = \frac{1}{2}\left(f_\theta(x) - y\right)^2$
- Logistic [`nn.SoftMarginLoss`] $L(f_\theta(x), y) = \log(1 + exp(-y f_\theta(x)))$
- Hinge loss [`nn.MultiMarginLoss` / `nn.MultiLabelMarginLoss`]
  $L(f_\theta(x), y) = \max(0,\ 1 - y f_\theta(x))$
- Cross-entropy [`nn.CrossEntropyLoss`]
  $L(f_\theta(x), y) = -\left[y \log(f_\theta(x)) - (1 - y)\log(1 - f_\theta(x))\right]$
- …and many more: https://pytorch.org/docs/stable/nn.html#loss-functions

Training Loop

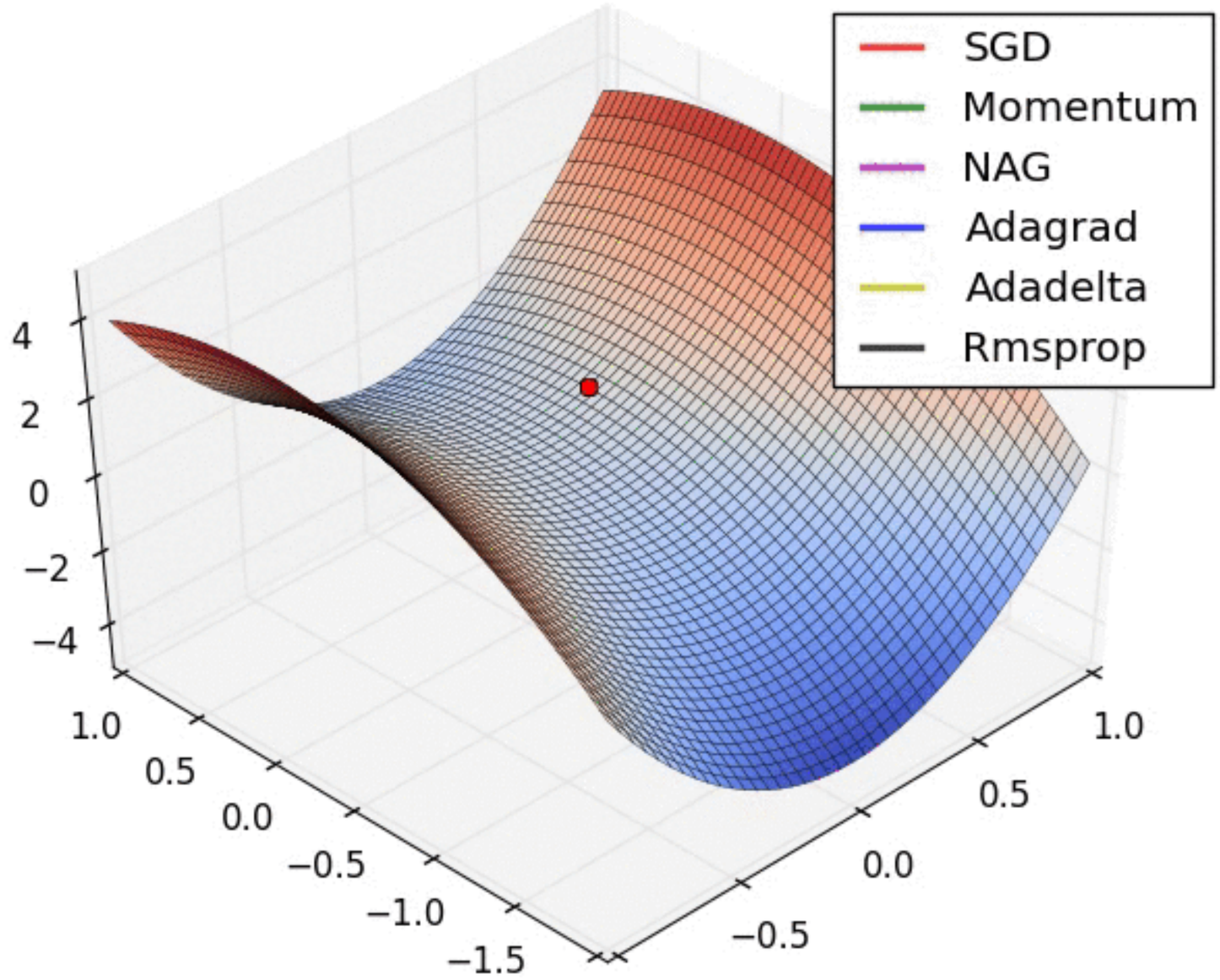
- Given model $\mathbf{f_\theta}$, initialize parameters $\boldsymbol{\theta}$ (e.g. randomly)
- For number of epochs:
  - For number of iterations (i.e. number of batches in data):
    - Sample batch of data **x, y ~ T**
      - **x**: input
      - **y**: target output
    - Run model forward $\mathbf{y^* = f_\theta(x)}$ and calculate loss **L(y*, y)**
    - Calculate gradient of loss w.r.t. parameters using backprop
    - Update parameters using optimizer, e.g., $\theta_{t+1} = \theta_t - \alpha \nabla \theta_t$

Stochastic Gradient Descent



Li, Hao, et al. "Visualizing the loss landscape of neural nets." *NeurIPS*. 2018.

Optimizers



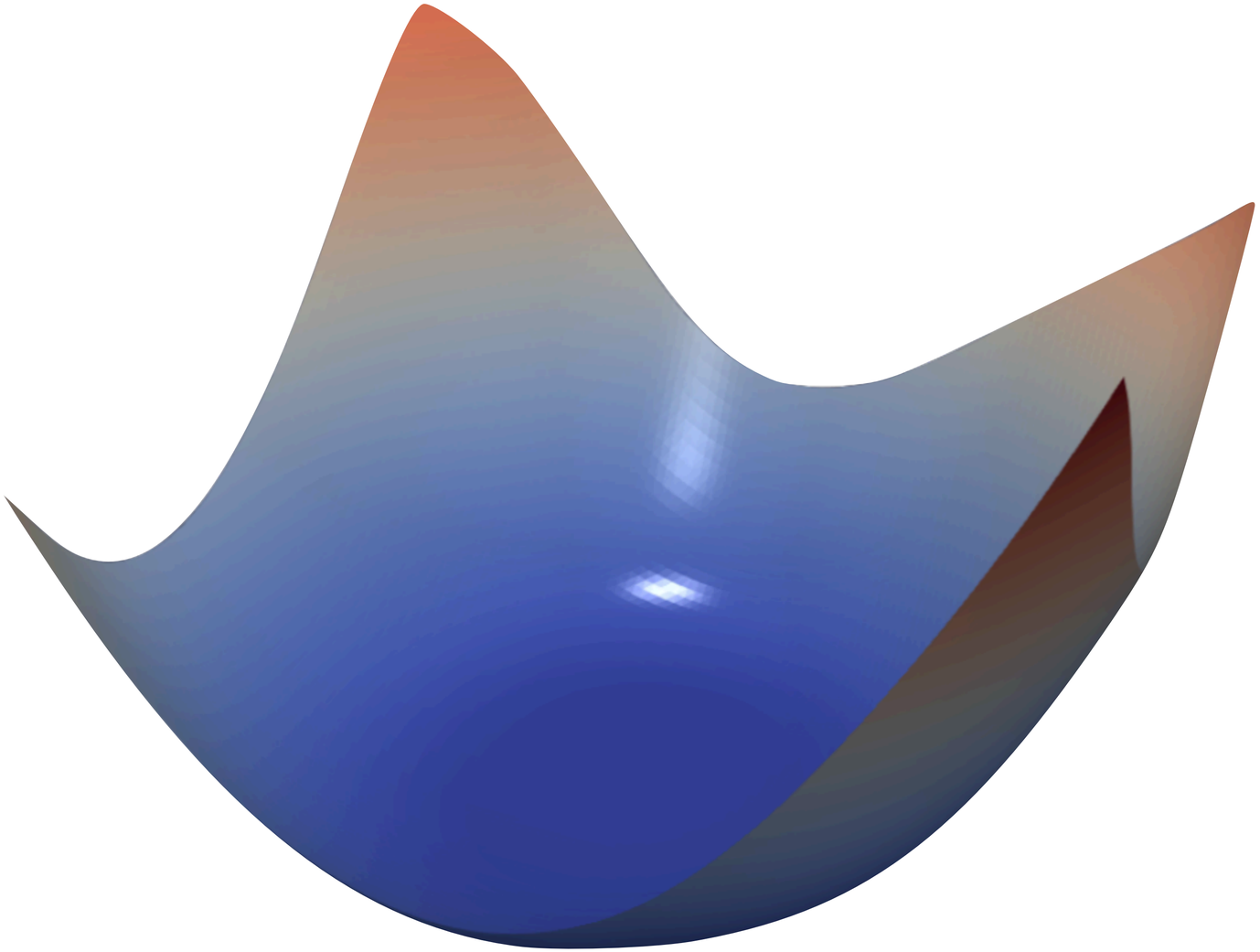
- `torch.optim.SGD`
- `torch.optim.Adagrad`
- `torch.optim.Adadelta`
- `torch.optim.Adam`
- `torch.optim.RMSprop`
- … and many more

http://ruder.io/optimizing-gradient-descent/

PyTorch Training Loop Scaffold

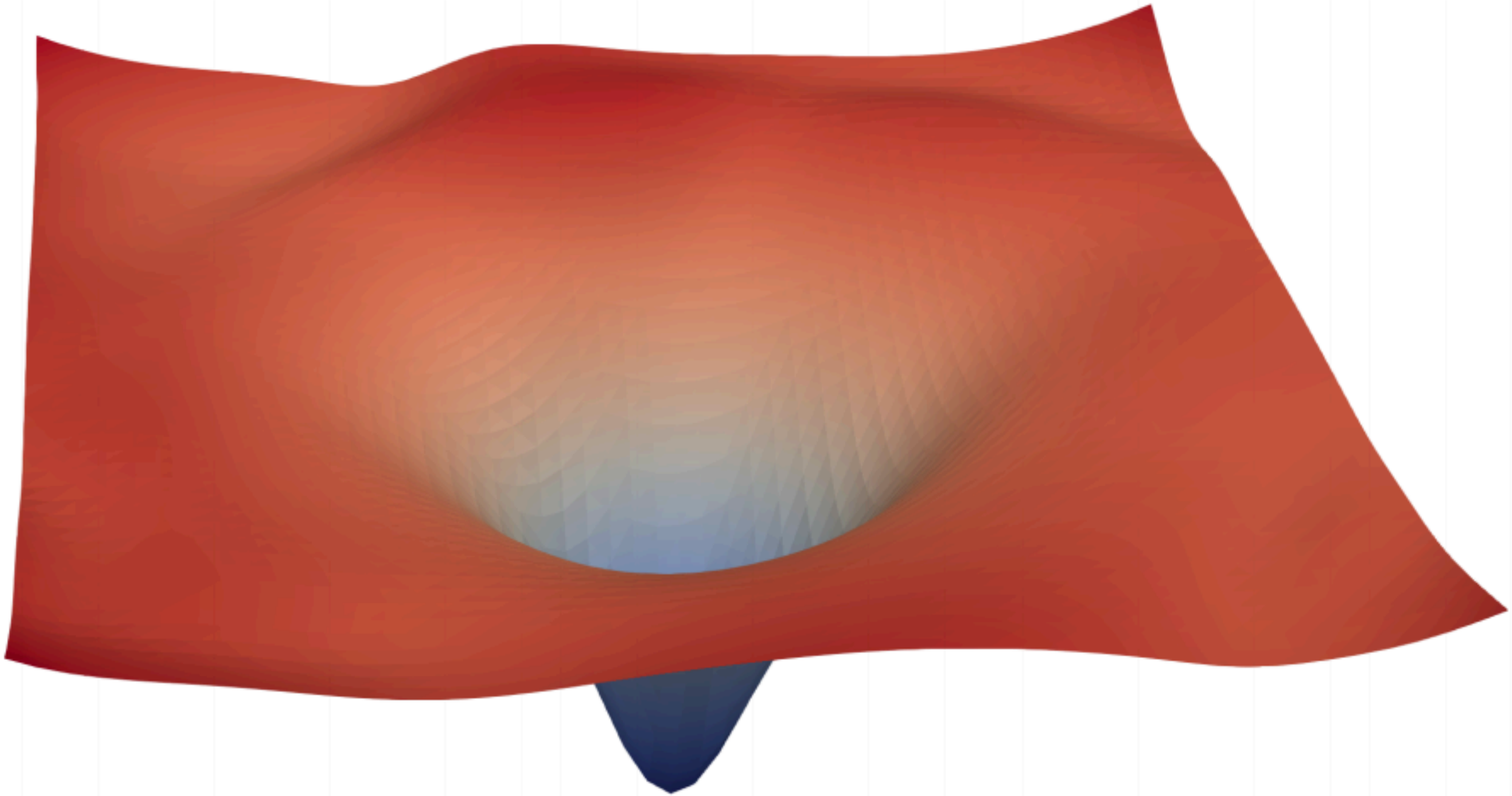
```
import torch.nn as nn
import torch.optim as optim

# Set a seed; your experiments should be reproducible!
torch.manual_seed(1)
# Load data
train, dev, test = ...
# Instantiate model
model = MyModel(...)
# Define loss function
loss_fn = nn.CrossEntropyLoss()
# Instantiate optimizer with a learning rate (lr)
optimizer = optim.SGD(model.parameters(), lr=0.1)

for epoch in range(10):  # 10 epochs in this example
  for i, batch in enumerate(train):  # assuming train is a generator that reshuffles data
    # Set gradients to zero
    optimizer.zero_grad()
    # Run forward
    y = model(batch)
    # Calculate loss
    loss = loss_fn(y, y_target)
    # Run backward to compute gradient of loss w.r.t. to model parameters
    loss.backward()
    # Perform one step of optimization
    optimizer.step()
    # Print diagnostics (e.g. loss or dev set performance)
    ...

# Evaluate on test
...
```

Loss Landscapes in the Wild



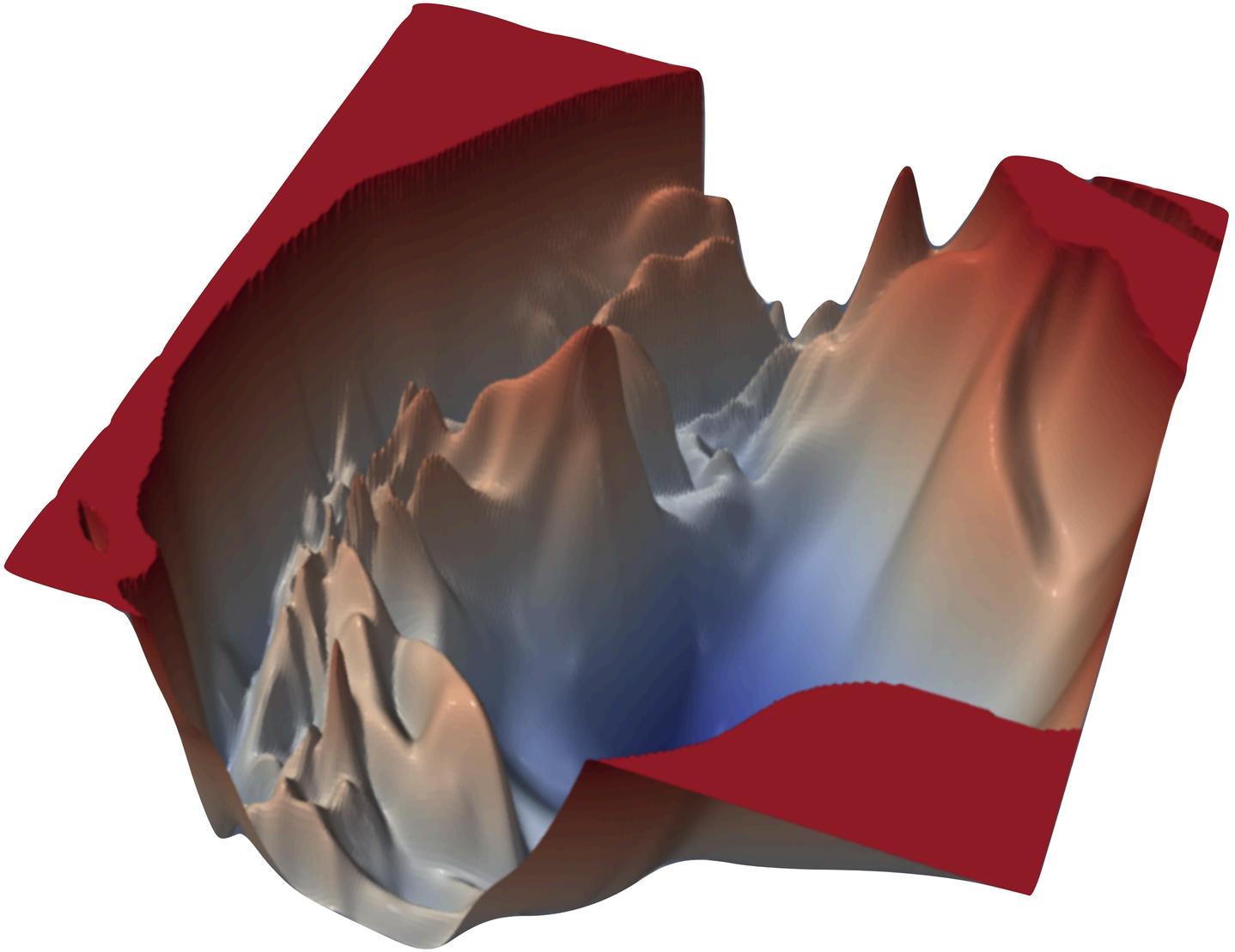
Li, Hao, et al. "Visualizing the loss landscape of neural nets." *NeurIPS*. 2018.

Loss Landscapes in the Wild



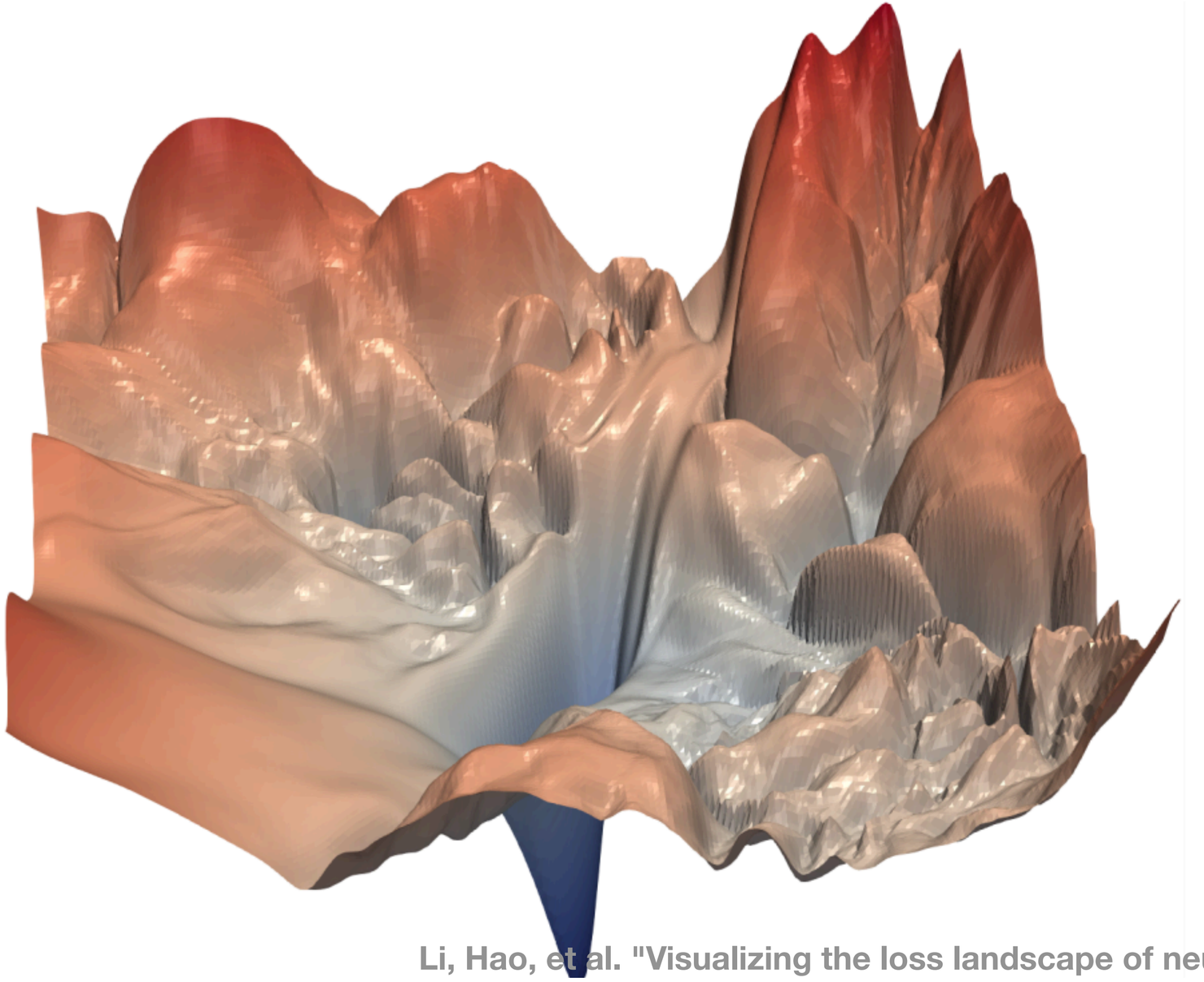
Li, Hao, et al. "Visualizing the loss landscape of neural nets." *NeurIPS*. 2018.

Loss Landscapes in the Wild



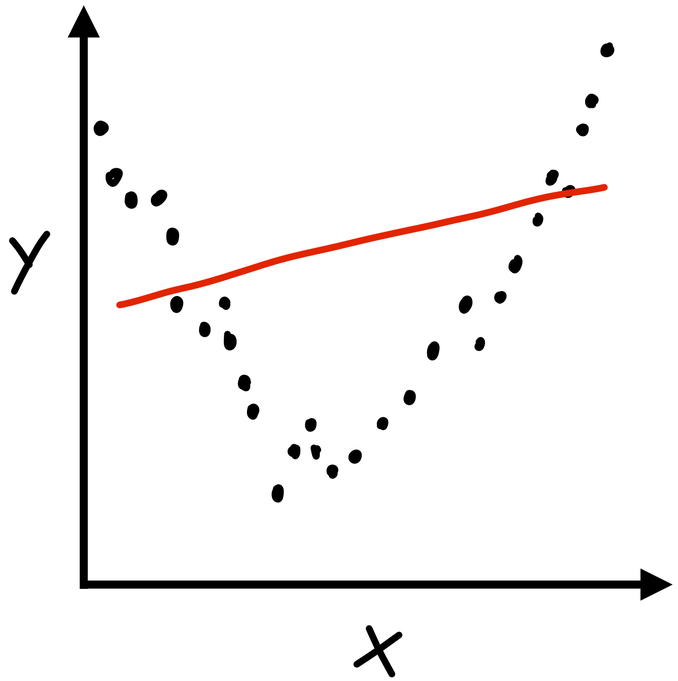
Li, Hao, et al. "Visualizing the loss landscape of neural nets." *NeurIPS*. 2018.

Loss Landscapes in the Wild

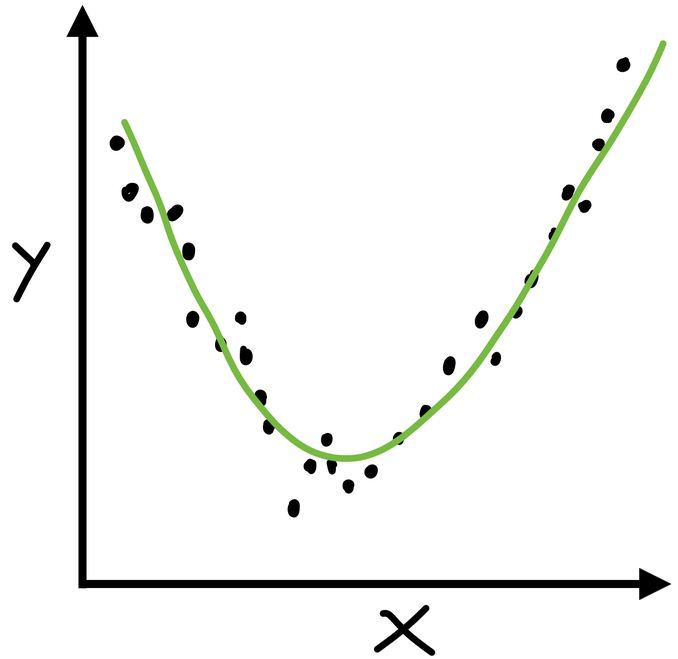


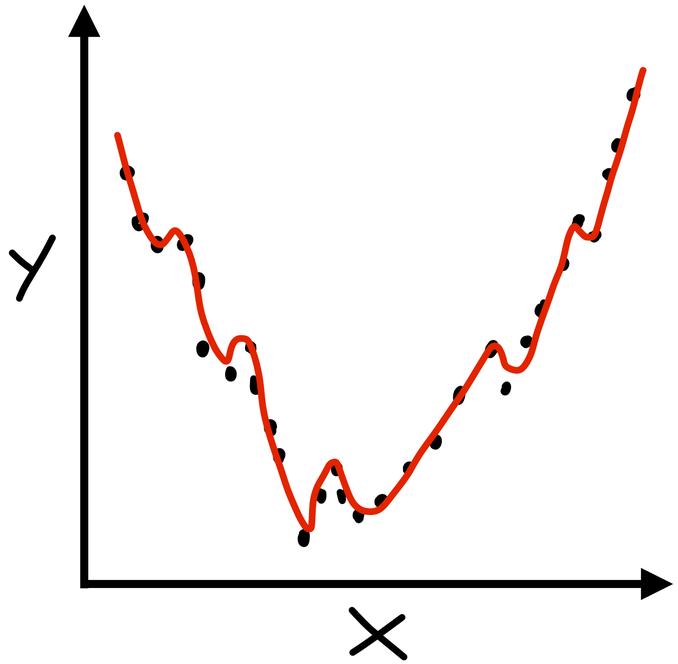
Li, Hao, et al. "Visualizing the loss landscape of neural nets." *NeurIPS*. 2018.

Underfitting and Overfitting



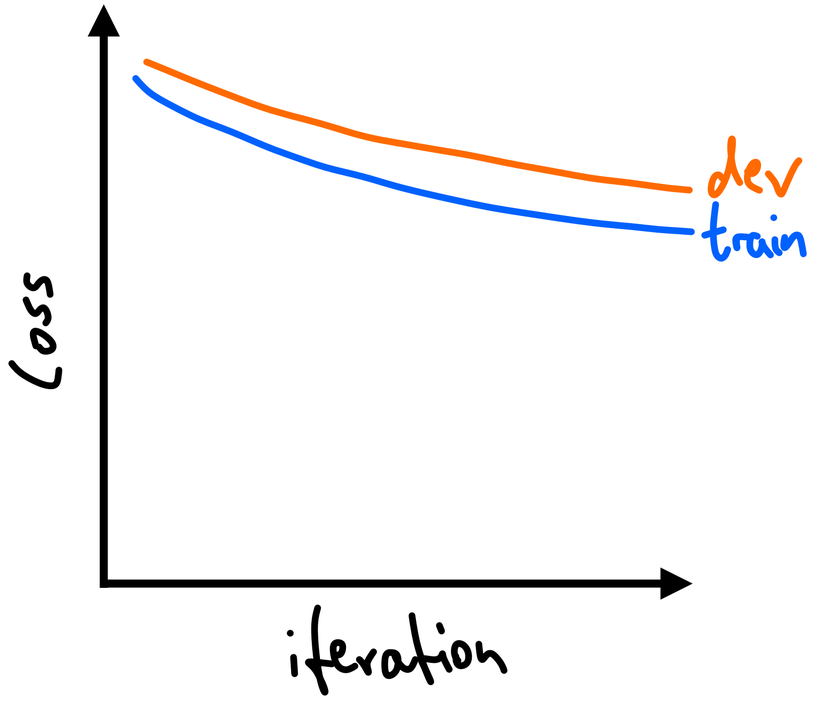
underfit



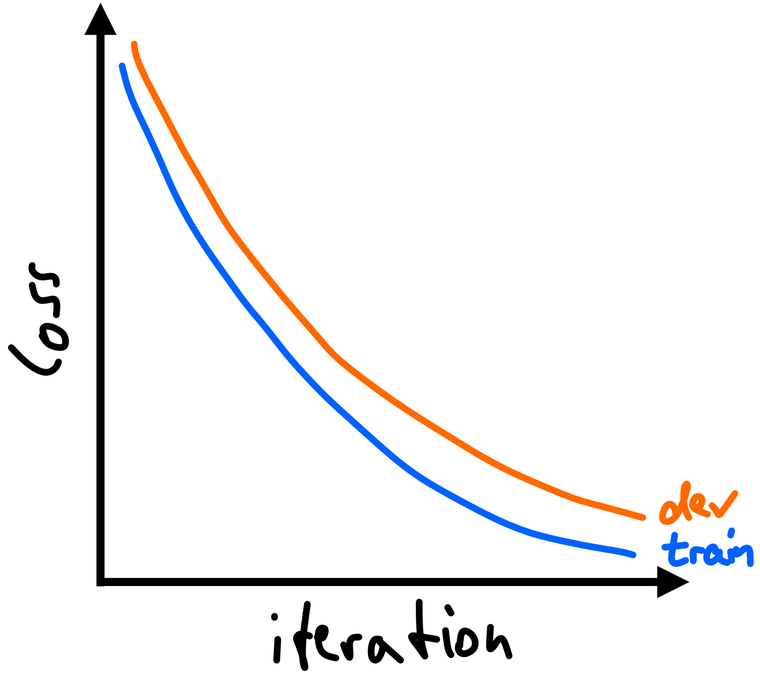


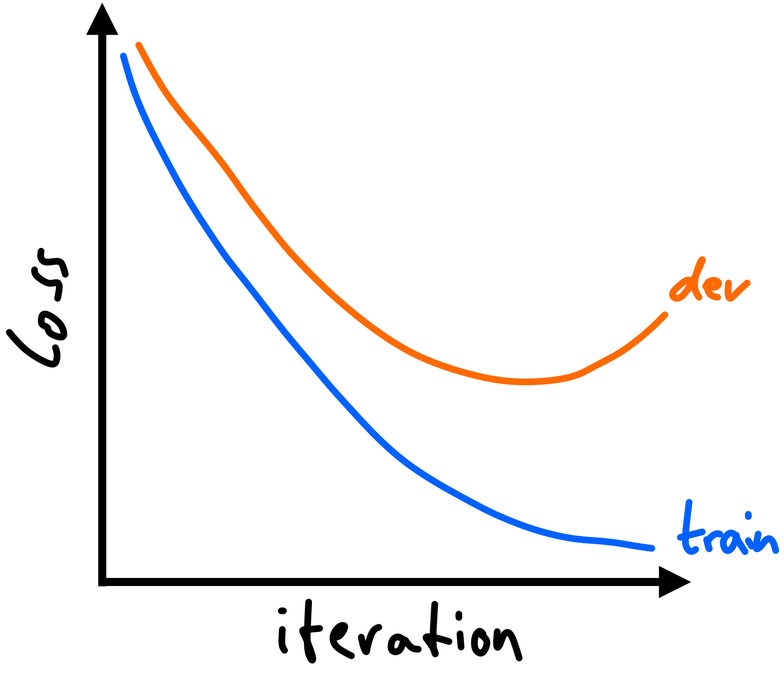
overfit

Underfitting and Overfitting



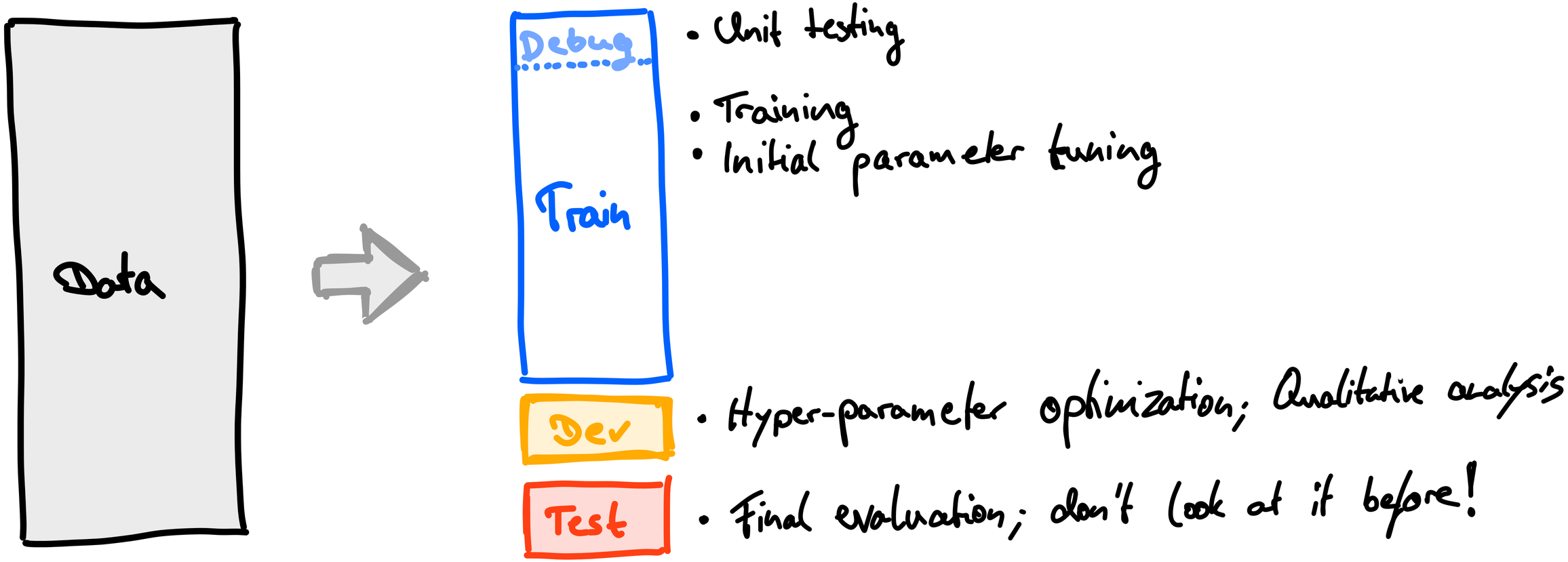
underfit



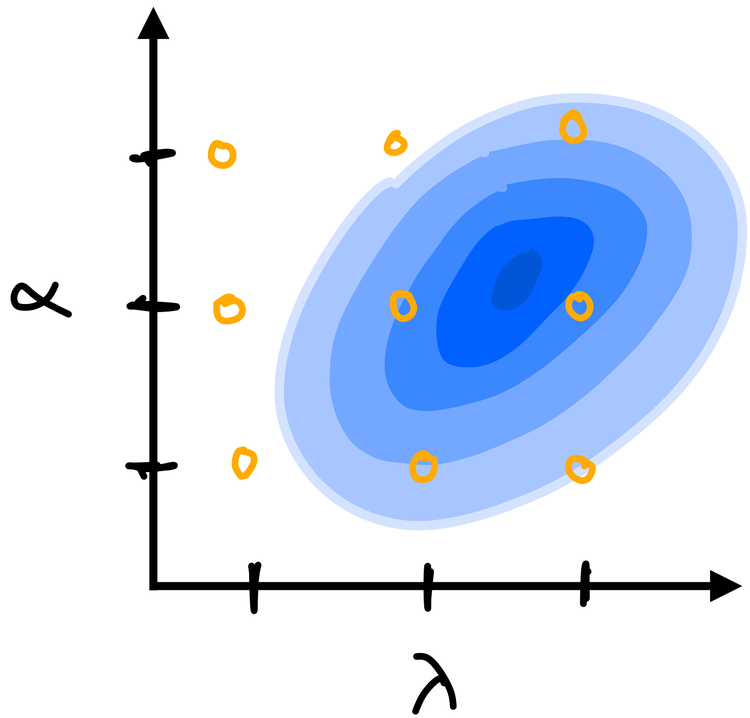


overfit

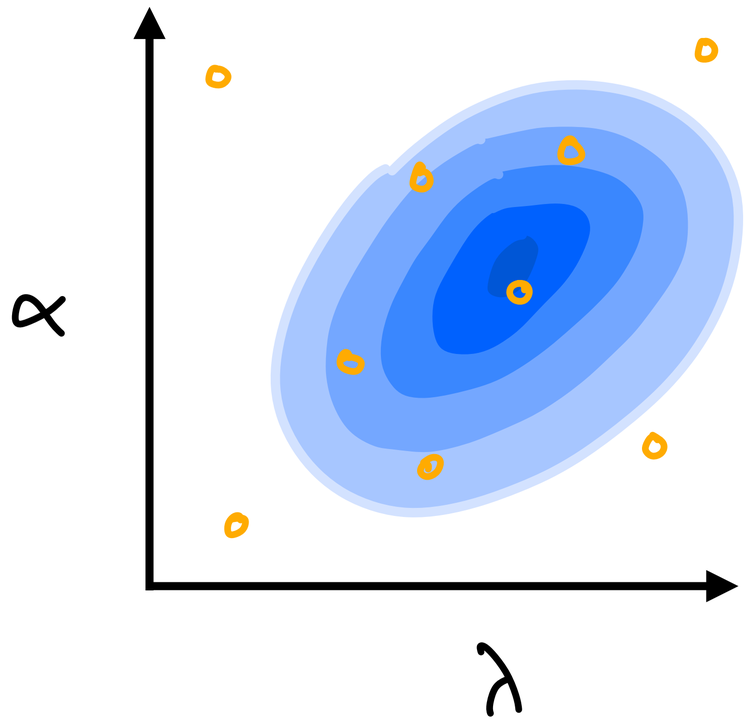
Train / Dev / Test Splits



Hyper-parameter Optimization



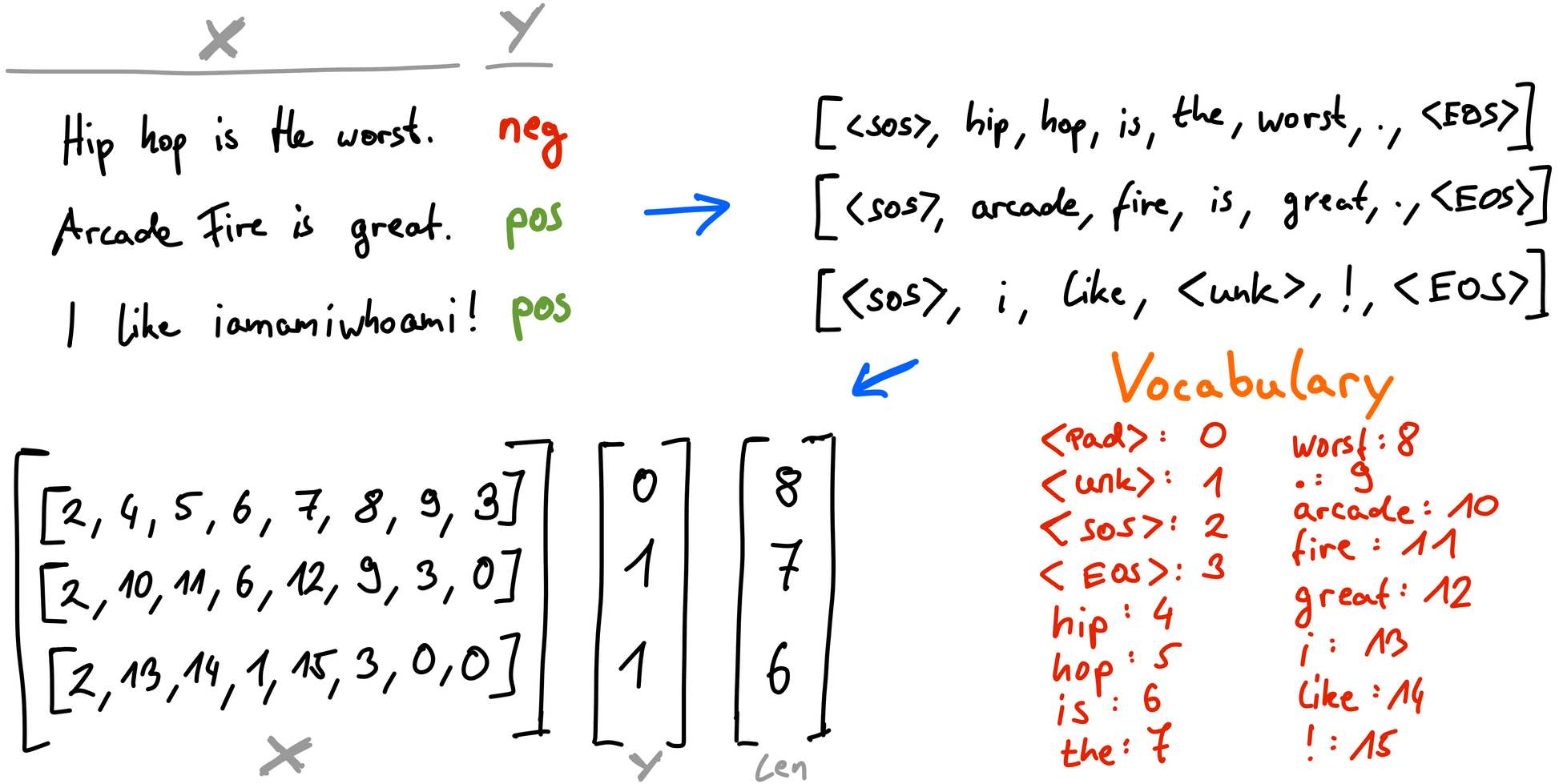
Grid search



Random search

AllenNLP

- Data processing utilities and popular datasets for natural language
- https://allennlp.org/



AllenNLP

- In your own time, go through an example here.